# *Code & Creative Higher Education*

## UNDERSTANDING COMPUTER PROGRAMMING PRACTICES IN PORTUGUESE ARTS AND DESIGN UNDERGRADUATE COURSES

EDUARDO MORAIS

# CODE AND CREATIVE HIGHER EDUCATION

## UNDERSTANDING COMPUTER PROGRAMMING PRACTICES IN PORTUGUESE ARTS AND DESIGN UNDERGRADUATE COURSES

**Eduardo José Botelho Batista Morais de Sousa**

DOCTORAL PROGRAM IN DIGITAL MEDIA

2021

Advisor:

**Carla Susana Lopes Morais**

Auxiliary Professor
Faculdade de Ciências da Universidade do Porto

Co-advisor:

**João Carlos de Matos Paiva**

Associate Professor
Faculdade de Ciências da Universidade do Porto

U.PORTO

NOVA
UNIVERSIDADE NOVA
DE LISBOA

UT AUSTIN | PORTUGAL × FCT SCHOLARSHIP

Eduardo Morais benefitted from a UT AUSTIN | PORTUGAL Doctoral Scholarship in Digital Media. Financial support was awarded by the European Social Fund under the Norte 2020 operational program and by the FCT – Portuguese Foundation of Science and Technology (scholarship code PD/BD/128416/2017).

# ACKNOWLEDGEMENTS

I must begin by thanking Carla Morais and João C. Paiva for their attentive supervision and mentorship. I am grateful for their support and generosity in making this long doctoral journey possible and for their frank and warm demand of commitment and rigor.

I must thank António Coelho, director of the Doctoral Program in Digital Media, and Marisa Silva, at the FEUP Department of Informatics Engineering, for their prompt and indefatigable support throughout my journey.

I am grateful to all my friends who have supported and heard me throughout my journey. In particular, I want to thank Tiago Assis, Hugo Brito, Constança Carvalho Homem, Cecília Durães, Mário Martins Antunes, Luciano Moreira, Sofia Pinto, Helena Restivo, Ana Luísa Sá, and Fátima São Simão for their academic insights.

This thesis would have not been possible without the help of the large cast that helped with my inquiries over the last several years. I am grateful for every letter of support, for every action made to forward my surveys or to host my studies, for every generous advice: Eduarda Abrantes, Luís Afonso, Luísa Aires, Heitor Alvelos, Valter Alves, Miguel Amado, Pedro Amado, Fernando Amaral da Cunha, Dimitris Andrikopoulos, Pedro Ângelo, José Azevedo, Solange Azevedo, Daniel Brandão, Pedro Cabral, Jorge Cardoso, Pedro Cardoso, Manuela Carneiro, Miguel Carvalhais, José Pedro Cavalheiro, Rui Coelho, Duarte Costa Pereira, Manuel Costa e Silva, Fred D. Davis, Paulo Fernandes, Célia Figueiredo, Pau Galbis, Alice Geirinhas, Patrícia Gouveia, Ricardo Guerreiro, Marco Heleno, Francisco Jesus, Joana Lessa, Maria José Marcelino, Carlos Melo Ferreira, Ana Isabel Moniz, Natacha Moniz, Joaquim Moreira de Sousa, Carlos Oliveira, Miguel Oliveira, Sofia Oliveira, Teresa Pereira, Rui Penha, Armando Rabaça, Tiago Ralha, Carlos Reis, José Miguel Rodrigues, Paula Rodrigues, Ricardo Rodrigues, Cândida Ruivo, Rui Santos, Miguel Serafim, Pedro Serapicos, Eduardo Serra, André Sier, António Sousa Dias, João Paulo Sousa, Sara Tranquada, Ricardo Torcato, Viswanath Venkatesh, Paula Viana, and S. Craig Watkins.*

I am also grateful to Gil Nata and to my dear friend Joana Cabral for their specialized advice concerning statistical analyses. I am also deeply indebted to Mariana Figueiredo for gifting me the time she took to proofread this thesis' chapters.

Finally, I am thankful to my mother and father for their help and support.

* My deep and sincere apologies if I left anyone out.

*(this page was intentionally left blank)*

ABSTRACT

After a long and contentious history, a preoccupation with the potential of the computer as an artistic medium began to gain traction in education over the last two decades, in simultaneity with a growing concern with students' digital literacies and computational thinking skills. As of 2018/19, 7,000 (or almost 40%) of the students enrolled in arts and design undergraduate courses at Portuguese public higher education institutions would have the occasion to learn computer programming during their studies. Still, there is a literature gap about introducing that subject in arts and design higher education courses. Hence, this inquiry's primary goal was to characterize how those educators and students accept and engage in teaching and learning to program. To that end, we undertook our research in two phases. In the preliminary stage, we conducted a quantitative survey of the acceptance of computer programming of 270 students attending and 80 professors assigned to relevant arts and design courses. The results informed the main stage of our research, in which we conducted a collective case study of six curricular units about programming, which were a part of distinct licentiate curricula in different higher education institutions.

Our findings merit articulation with the extant scholarship on higher education, arts education, and computing pedagogies. FIRST, the dichotomous perspectives of code as either a practical tool or as a medium shaped how professors taught programming. SECOND, the professors were often disconnected from the recommended pedagogies. Most saw themselves as facilitators striving to teach the students to be effective learners, yet the same professors were also likely to solve students' immediate problems rather than expecting them to engage in deep thinking. THIRD, a previous familiarity was a major predictor of students' interest in programming. Students were also less likely to commit to deep learning if the subject was introduced at the later stages of the curriculum. FOURTH, the students saw the planning and the algorithm design, not writing, as the most challenging aspects of computer programming. FIFTH, the professors regarded students' learning as fragile, and there were gaps in learning assurance that deserve serious attention by institutions, lest they deflate the value of certification.

Given how code came to permeate human endeavors and the unique position of artists and designers in its interpretation and continued development, the moment requires a more robust engagement with the relationship between code and the arts in the educational stage. We believe that our findings hold lessons that can be beneficial and thus merit the attention of policymakers, institutions, curriculum designers, educators, and future researchers.

***Keywords:*** *higher education, arts education, computer programming, pedagogies, case studies*

*(blank page)*

# RESUMO

Após uma história já longa e contenciosa, a preocupação com o potencial do computador enquanto meio artístico começou a ganhar força no campo da educação nas duas últimas décadas, em simultâneo com uma crescente atenção às literacias digitais e às capacidades de pensamento computacional dos estudantes. Em 2018/19, 7.000 (ou quase 40%) estudantes inscritos em cursos de primeiro ciclo em artes e design nas instituições públicas de ensino superior portuguesas teriam a ocasião de aprender programação de computadores durante os seus estudos. Ainda assim, existe uma lacuna na literatura sobre a introdução dessa matéria nos cursos de ensino superior em artes e design. Assim, o primeiro objectivo desta investigação foi caracterizar como é que esses educadores e estudantes encaram e se empenham no ensino e na aprendizagem de programação. Com esse fim, empreendemos a nossa pesquisa em duas fases. Na fase preliminar, efectuámos um inquérito quantitativo sobre a aceitação da programação de computadores a 270 estudantes inscritos e 80 professores com serviço distribuído em cursos de artes e design. Os resultados informaram a fase principal da nossa pesquisa, em que realizámos um estudo de caso colectivo que incidiu sobre seis unidades curriculares sobre programação, integradas em diferentes cursos de licenciatura de diferentes instituições de ensino superior.

Os resultados obtidos merecem uma articulação com o conhecimento existente sobre ensino superior, ensino artístico, e pedagogias da computação. PRIMEIRO, as visões dicotómicas do código, ora uma ferramenta prática, ora um *medium*, moldaram como os professores ensinaram programação. SEGUNDO, os professores mostraram um frequente desconhecimento de estratégias pedagógicas relevantes. A maioria reviu-se no papel de facilitador e expressou vontade de ensinar os estudantes a serem aprendizes eficazes. Porém, os mesmos professores demonstraram propensão para resolver as dificuldades imediatas dos estudantes ao invés de esperar pelo seu empenho numa reflexão profunda. TERCEIRO, uma familiaridade prévia foi o principal preditor do interesse dos estudantes na programação. Os estudantes também mostraram um menor empenho numa aprendizagem profunda quando a matéria era apresentada nos anos mais tardios do currículo. QUARTO, os estudantes viram o planeamento e a concepção de algoritmos, não a escrita, como os aspectos mais desafiantes da programação de computadores. QUINTO, os professores avaliaram os conhecimentos adquiridos como frágeis e verificaram-se lacunas na garantia de aprendizagem que merecem a atenção séria das instituições, dado o risco para o valor das certificações.

Face a como o código veio a permear a actividade humana e ao posicionamento único de artistas e designers para a sua interpretação e para o seu desenvolvimento, o momento presente requer um compromisso mais robusto entre o código e as artes no palco educativo. Acreditamos que os resultados desta investigação contêm lições e implicações que merecem a atenção de instituições, educadores, poder político e responsáveis pelo desenvolvimento curricular, bem como de futuros investigadores nesta área.

*(blank page)*

CONTENTS

Chapter 1
INTRODUCTION 1

1.1 My path and motivation 1

1.2 Purpose and audience 3

1.3 Thesis style 4

1.4 Thesis organization 5

PART ONE

Chapter 2
THE CONVERGENCE OF COMPUTING AND EDUCATION 9

2.1 Historical overview and recent trends 9

2.1.1 Crucial historical developments 10

2.1.2 Recent technology adoption trends 14

2.1.2.1 Computer and internet adoption trends in Portugal 14

2.2 Media convergence and new technological literacies 17

2.2.1 Media convergence 18

2.2.2 Higher Education in the convergence era 21

2.2.2.1 A critical engagement with technology 23

2.2.3 Computing literacies 24

2.2.3.1 Studies of computing and information literacies 27

2.3 Computer programming in education 29

2.3.1 Computing education research 30

2.3.1.1 Main questions in the field 31

2.3.2 Teaching objectives and pedagogical strategies 32

2.3.2.1 Learning objectives for novices 32

2.3.2.2 Pedagogical strategies 33

Chapter 3
THE ARTS, PROGRAMMING, AND HIGHER EDUCATION 35

3.1 Computers and the arts 35

3.1.1 Foundations of digital art 36

3.1.1.1 Early computer art 36

3.1.1.2 Computers and the art world 39

3.1.2 The infatuation with software tools 41

3.1.3 Programming and exploration of the medium 44

3.2 Creative computing and higher education 46

3.2.1 Characteristics of arts and design education 47

3.2.1.1 Contemporary challenges to the signature pedagogies 48

3.2.2 Perspectives on the encounter with digital media 49

3.2.3 Computer programming within arts and design education 51

3.2.3.1 Creative code pedagogy 52

3.2.3.2 The research gap 54

3.3 Programming languages for arts education 55

3.3.1 Criteria for educational programming languages 56

3.3.2 Common languages for novices and practitioners 57

3.3.2.1 General-purpose programming languages 57

3.3.2.2 Web programming 58

3.3.2.3 Processing and P5.js 59

3.3.2.4 Visual programming languages 61

3.3.2.5 Game creation systems 62

3.3.2.6 Hardware tools 64

3.3.2.7 Blocks-based Languages 65

Chapter 4
CODE WITHIN PORTUGUESE PUBLIC ARTS AND DESIGN UNDERGRADUATE EDUCATION—THE PRESENT PANORAMA 67

4.1 The Portuguese public Higher Education system 67

4.1.1 Terminology 68

4.1.2 A short chronology in facts and figures 69

4.1.2.1 The Bologna Process 71

4.1.2.2 Academic staff in recent decades 72

4.1.2.3 Students and diplomas in recent decades 73

4.1.3 Higher arts, design, and architecture education 75

4.1.3.1 Arts and design within the CNAEF categorization 75

4.1.3.2 The specific case of Architecture 77

4.1.3.3 Facts and figures 78

4.2 Panorama of programming education for arts and design 80
4.2.1 The courses 81
4.2.1.1 The students 84
4.2.1.2 A brief chronology 85
4.2.2 The curricular units 87
4.2.2.1 The syllabi 94
4.2.2.2 Session types and teaching methodologies 97
4.2.2.3 Evaluation methodologies 98

PART TWO

Chapter 5
RESEARCH DESIGN AND METHODS 103

5.1 The researcher's position and objectives 103
5.1.1 Biases and orientation 104
5.1.2 Research objectives and contribution 104
5.2 Theoretical framework(s) 106
5.2.1 Symbolic interactionism and the pragmatic paradigm 107
5.2.2 Overview of technology acceptance theories 108
5.2.2.1 From TAM to UTAUT: a short chronology 109
5.2.3 The Unified Theory of Acceptance and Use of Technology 111
5.2.3.1 UTAUT in literature 114
5.2.3.2 Criticisms and limitations 115
5.2.3.3 UTAUT's applicability to the present research 115
5.3 Research design 117
5.3.1 Characteristics of qualitative research 117
5.3.1.1 Mixed methods research 119
5.3.2 Our research design 119
5.3.2.1 Preliminary survey 120
5.3.2.2 Case studies 121
5.4 Quantitative research 122
5.4.1 The survey instruments 122
5.4.1.1 Characterization and programming use questions 125
5.4.1.2 Translation and pilot tests 126
5.4.2 Sample and data collection procedures 127
5.4.2.1 Preliminary survey 127
5.4.2.2 Case studies' surveys 128
5.4.3 Data analysis procedure and validation 129

5.4.3.1 Students' preliminary survey validation 129
5.4.3.2 Professors' preliminary survey validation 131
5.4.3.3 Case studies' surveys 132

5.5 Case study research 133

5.5.1 Sample and data collection procedures 133
5.5.1.1 Documentation and online content 135
5.5.1.2 Classroom observation 136
5.5.1.3 Students' projects and exercises 137
5.5.1.4 Interviews with the professors 138
5.5.1.5 Quantitative surveys of students 140

5.5.2 Qualitative data analysis 141
5.5.2.1 Intra-case analysis 143
5.5.2.2 Inter-case analysis 143

5.5.3 Quantitative surveys' validation and analysis 144
5.5.3.1 Statistical analyses 145

PART THREE

Chapter 6
A PRELIMINARY SURVEY OF STUDENTS AND PROFESSORS' ACCEPTANCE AND USE OF COMPUTER PROGRAMMING 149

6.1 Overview 149

6.1.1 Communications related to these surveys 150

6.2 Students' acceptance and use of programming 151

6.2.1 Participants 151
6.2.1.1 Participants' programming knowledge 153

6.2.2 Students' acceptance and intentions 154
6.2.2.1 Participants without programming knowledge 154
6.2.2.2 Participants learning programming for the first time 155
6.2.2.3 Participants with prior programming knowledge 155

6.2.3 Students' intentions and use behavior 156

6.3 Professors' acceptance and use of programming 158

6.3.1 Participants 158
6.3.1.1 Participants' practice and programming knowledge 159

6.3.2 Professors' acceptance and intentions 160
6.3.2.1 Participants without programming knowledge 160
6.3.2.2 Participants with prior programming knowledge 161

6.3.3 Professors' intentions and use behavior 161

6.3.4 Professors' views on teaching to code 162

Chapter 7
CASE STUDIES IN PROGRAMMING EDUCATION WITHIN UNDERGRADUATE ARTS AND DESIGN 165

7.1 Overview 165

7.2 PRIMER: multimedia programming projects 168

7.2.1 The course and its host institution 168

7.2.2 The curricular unit 169

7.2.3 Session observation 172

7.2.3.1 Classroom conditions 172
7.2.3.2 Project supervision 174
7.2.3.3 Professors' behavior 176
7.2.3.4 Students' behavior 177

7.2.4 Online learning management and educational resources 178

7.2.5 Students' projects 180

7.2.6 Students' acceptance and use of computer programming 183

7.2.6.1 Students' acceptance and behavior 184

7.2.7 Interviews with the professors 185

7.2.8 Successes and limitations faced in this case study 192

7.3 GRAPHIC: computer visuals and animations 193

7.3.1 The course and its host institution 193

7.3.2 The curricular unit 194

7.3.3 Session observation 196

7.3.3.1 Classroom conditions 196
7.3.3.2 The lectures 197
7.3.3.3 Tutoring 200

7.3.4 Students' work 201

7.3.5 Interview with the professor 204

7.3.6 Successes and limitations faced in this case study 207

7.4 SKETCH: basics of interactive design 208

7.4.1 The course and its host institution 208

7.4.2 The curricular unit 209

7.4.3 Session observation 212

7.4.3.1 Classroom conditions 212
7.4.3.2 The lectures 214
7.4.3.3 Project supervision 218

7.4.4 Online learning management and educational resources 219

7.4.5 Students' projects 221

7.4.6 Students' acceptance and use of computer programming 222
7.4.6.1 Students' acceptance and behavior 223
7.4.6.2 Acceptance and behavior at the end of the semester 224
7.4.7 Interview with the professor 225
7.4.8 Successes and limitations faced in this case study 229
7.5 WEAVE: programming for the web 230
7.5.1 The course and its host institution 230
7.5.2 The curricular unit 231
7.5.3 Session observation 233
7.5.3.1 Classroom conditions 234
7.5.3.2 The lectures 236
7.5.3.3 Practical sessions 239
7.5.4 Online educational resources 241
7.5.5 Students' acceptance and use of computer programming 241
7.5.5.1 Students' acceptance and behavior 243
7.5.5.2 Acceptance and behavior at the end of the semester 244
7.5.6 Interviews with the professors 244
7.5.7 Successes and limitations faced in this case study 246
7.6 TOUCH: graphics and tangible interfaces 248
7.6.1 The course and its host institution 248
7.6.2 The curricular unit 249
7.6.3 Session observation 251
7.6.3.1 Classroom conditions 252
7.6.3.2 Project supervision session 253
7.6.4 Students' project exhibition and presentation 255
7.6.4.1 Public exhibition 255
7.6.4.2 Project presentation 255
7.6.5 Students' acceptance and use of computer programming 257
7.6.6 Interview with the professor 259
7.6.7 Successes and limitations faced in this case study 261
7.7 FLOW: visual programming for musical composition 262
7.7.1 The course and its host institution 262
7.7.2 The curricular unit 263
7.7.3 Session observation 265
7.7.3.1 Classroom conditions 266
7.7.3.2 The lectures 267
7.7.3.3 Project supervision 271

7.7.4 Online learning educational resources 273
7.7.5 Students' acceptance and use of computer programming 273
7.7.6 Interview with the professor 274
7.7.7 Successes and limitations faced in this case study 278
7.8 Inter-case comparison 279
7.8.1 Units' positioning and characteristics 279
7.8.1.1 The value of programming 279
7.8.1.2 Relationship with the courses 280
7.8.1.3 Objectives and contents 281
7.8.1.4 Evaluation methods 281
7.8.2 Conditions and resources 282
7.8.2.1 Class size and session schedule 282
7.8.2.2 Spatial conditions, hardware, and software 282
7.8.2.3 Online support and resources 285
7.8.3 Professors' strategies and behavior 286
7.8.3.1 Motivational strategies 286
7.8.3.2 Theoretical lectures 287
7.8.3.3 Practical and tutoring sessions 290
7.8.4 Students' characteristics and classroom behavior 293
7.8.4.1 Behavior in theoretical lectures 293
7.8.4.2 Behavior in practical sessions 294
7.8.5 Students' acceptance and use of computer programming 295
7.8.6 Projects and learning outcomes 298
7.8.7 Limitations faced across the studies 300

PART FOUR

Chapter **8**
DISCUSSION OF OUR FINDINGS 305
8.1 Methodological summary and implications 305
8.1.1 Summary of research goals, design, and methods 306
8.1.2 Challenges and implications 306
8.1.2.1 Institutional resistances 307
8.1.2.2 Quantitative findings in the context of UTAUT scholarship 307
8.2 Programming within arts and design curricula 309
8.2.1 Rationales for teaching programming 309
8.2.2 Learning objectives 310

8.2.3 The studio 311
8.2.4 Programming language choice 311
8.3 Professors' role and teaching strategies 313
8.3.1 Roles and duties 313
8.3.1.1 Encouraging deep learning 314
8.3.1.2 Establishing rapport 314
8.3.1.3 Responsibility and demand 315
8.3.2 Teaching strategies 315
8.3.2.1 Lectures and live coding demonstrations 316
8.3.2.2 Consolidating knowledge 317
8.3.2.3 Practical projects and their supervision 318
8.3.2.4 Unexplored possibilities 319
8.3.3 Use of online resources 320
8.4 Acceptance, learning, and outcomes 322
8.4.1 Students' characteristics and acceptance of computer programming 322
8.4.1.1 Early contact and the effect of seniority 323
8.4.1.2 Gender differences 323
8.4.1.3 A note about culturally responsive computing 325
8.4.2 Autonomy and self-efficacy 325
8.4.2.1 Difficulties and reactions 326
8.4.2.2 Self-efficacy 327
8.4.3 Learning outcomes 327
8.4.3.1 Learning fragility 328

Chapter 9
CONCLUSIONS 331
9.1 Summary of findings 331
9.1.1 The value of computer programming 331
9.1.2 Working conditions 332
9.1.3 Pedagogical approaches 332
9.1.3.1 Online resources 332
9.1.3.2 Tutoring 333
9.1.4 Students and their acceptance of computer programming 333
9.1.5 Learning outcomes 333
9.2 Recommendations 335
9.2.1 For policymakers and higher education institutions 335
9.2.2 For professors 336
9.3 Future research directions 338

REFERENCES 341

LIST OF ANNEXES ON THE CD 359

*(blank page)*

# LIST OF FIGURES

## Chapter 2


2.1.1a Evolution of computer processors' efficiency 10
2.1.1b Evolution of component cost: transistors, DRAM, storage 10
2.1.2a *Acorn User Magazine, 25* (1984), magazine cover 12
2.1.2b *A Scuola con il Commodore 64* (1984), book cover 12
2.1.3a Evolution of the share of individuals using the internet 14
2.1.3b Evolution of mobile subscriptions per 100 people 14
2.1.4 Evolution of computer and internet users in Portugal, by age and gender 15
2.1.5 Share of Portuguese households with a computer and internet service 15


## Chapter 3


3.1.1a George Nees, *23 Ecke* (1964) 38
3.1.1b Frieder Nake, *Matrix Multiplication* (1967) 38
3.1.2a Lillian Schwartz & Ken Knowlton, *Pixilation* (1970), still detail 38
3.1.2b Charles Csuri, *Sine Wave Man* (1967) 38
3.1.3 Gordon Pask, *A Colloquy of Mobiles* at Cybernetic Serendipity (1968) 38
3.1.4 Sol LeWitt, *Wall Drawing #392* (1983) 40
3.1.5 Andy Warhol, Debbie Harry, and the Commodore Amiga computer (1985) 41
3.1.6 Zaha Hadid Architects, Antwerp Port Authority Building (2016) 43
3.3.1 The Processing 3.5 development environment 59
3.3.2 The P5.js online development environment 60
3.3.3 The Max 8 user interface 61
3.3.4 The Godot game creation system user interface 63
3.3.5 An Arduino board connected to a LCD screen 64
3.3.6 The Snap! online blocks editor 65


## Chapter 4


4.1.1 Municipal coverage of the Portuguese public higher education system 68
4.1.2 Evolution of academic staff at public higher education, by age and gender 71
4.1.3 Evolution of students at public universities, polytechnics, and private institutions 73
4.1.4 Evolution of higher education students, by degree level 74
4.1.5 Evolution of higher education diplomas, by degree level 75
4.2.1 Territorial distribution of public arts and design undergraduate courses with curricular units about computer programming (2018/2019) 84

## Chapter 5


5.2.1 Investigation foci and research domains, theories, and methods 106
5.2.2 The Theory of Reasoned Action (as in Madden, Ellen, & Ajzen, 1992) 109
5.2.3 Early version of the Technology Acceptance Model (Davis, 1986) 109
5.2.4 Parsimonious Technology Acceptance Model (Davis, 1989) 110
5.2.5 Unified Theory of Acceptance and Use of Technology (Venkatesh et al., 2003) 113
5.3.1 Our two-phase research design 120
5.4.1 Survey research presentation and privacy policy webpage 128
5.4.2 Our qualitative research model of students' acceptance and use of computer programming, after UTAUT (Venkatesh et al., 2003) 130
5.4.3 Our qualitative research model of professors' acceptance and use of computer programming, after UTAUT (Venkatesh et al., 2003) 132


## Chapter 6


6.2.1 Mean computer programming acceptances and *Behavioral Intention* of male and female students 155
6.2.2 Students' reported programming use frequency and intensity 157
6.3.1 Professors' reported programming use frequency and intensity 162


## Chapter 7


7.2.1 A view of the studio environment (PRIMER case study) 173
7.2.2 A view of the studio environment (PRIMER) 173
7.3.1 Students' view of the classroom environment (GRAPHIC case study) 196
7.3.2 Professor's view of the classroom environment (GRAPHIC) 197
7.3.3 A selection of image captures of the students' first exercises (GRAPHIC) 202
7.3.4 A selection of image captures of the students' second exercises (GRAPHIC) 203
7.4.1 Professor's view of the classroom environment (SKETCH case study) 213
7.4.2 Students' view of the classroom environment (SKETCH) 213
7.5.1 A view of the computer studio environment (WEAVE case study) 234
7.5.2 Students' view of the auditorium environment (WEAVE) 235
7.5.3 Professor's view of the auditorium environment (WEAVE) 235
7.6.1 A view of the studio environment (TOUCH case study) 252
7.6.2 A view of the studio environment (TOUCH) 252
7.7.1 A view of the classroom environment (FLOW case study) 266
7.7.2 View of a Max loop recorder source diagram (FLOW) 269

# LIST OF TABLES


Chapter **2**

2.3.1 Computer programming learning demands, after Robins et al. (2003) 32

Chapter **4**

4.2.1 Portuguese public arts and design undergraduate courses with curricular units about computer programming (2018/2019) 82

4.2.2 Curricular units about computer programming in Portuguese public arts and design undergraduate courses (2018/2019) 88

4.2.3 Bloom and Krathwohl's cognitive levels of the learning objectives of the curricular units about computer programming 93

4.2.4 Programming languages taught in higher arts and design education (2018/2019) 96

Chapter **5**

5.4.1 Acceptance and intention questionnaire items (indicative mood) —students with a prior programming experience and first-time learners 123

5.4.2 Acceptance and intention questionnaire items (conditional mood) —professors without any knowledge of computer programming 124

5.4.3 Participant characterization and computer programming usage questions 126

5.4.4 Students' questionnaire item groups reliability and consistency 129

5.4.5 Professors' questionnaire item groups reliability and consistency 131

5.5.1 Overview of our case studies 134

5.5.2 Overview of the data collected in each case study 135

5.5.3 Overview of students' survey participation in our case studies 141

5.5.4 Qualitative analyses main coding categories and subcategories 142

Chapter **6**

6.2.1 Student participants' institutions and courses 152

6.2.2 Students' characteristics and familiarity with programming 153

6.2.3 Students' programming acceptance, grouped by familiarity 154

6.2.4 Pearson correlations between students' acceptance and *Behavioral Intention* 156

6.2.5 Spearman correlations between students' programming use and their regard of the *Facilitating Conditions* and *Behavioral Intention* 157

6.3.1 Professors' characteristics and familiarity with programming 158

6.3.2 Professors' programming acceptance, grouped by familiarity 160

6.3.3 Pearson correlations between professors' acceptance and *Behavioral Intention* 161

## Chapter 7


7.1.1 Overview of the contact with computer programming across our case studies 166
7.1.2 Overview of the professors participating in our case studies 167
7.2.1 Summary of the observed sessions of the PRIMER curricular unit 172
7.2.2 Students' characteristics and familiarity with programming (PRIMER, week 4) 183
7.2.3 Students' programming acceptance, grouped by familiarity (PRIMER, week 4) 184
7.2.4 Correlations between students' acceptance and *Behavioral Intention* (PRIMER) 185
7.2.5 Successes and limitations (PRIMER case study) 192
7.3.1 Successes and limitations (GRAPHIC case study) 207
7.4.1 Summary of the observed sessions of the SKETCH curricular unit 212
7.4.2 Students' characteristics and familiarity with programming (SKETCH, week 2) 223
7.4.3 Students' programming acceptance, grouped by familiarity (SKETCH, week 2) 223
7.4.4 Correlations between students' acceptance and *Behavioral Intention* (SKETCH) 224
7.4.5 Successes and limitations (SKETCH case study) 229
7.5.1 Students' characteristics and familiarity with programming (WEAVE, week 3) 242
7.5.2 Students' programming acceptance, grouped by familiarity (WEAVE, week 2) 243
7.5.3 Correlations between students' acceptance and *Behavioral Intention* (WEAVE) 244
7.6.1 Comparison between the programming acceptance of students familiar with programming (TOUCH, week 6) and third-year preliminary survey participants 258
7.6.2 Successes and limitations (TOUCH case study) 261
7.7.1 Summary of the observed sessions of the FLOW curricular unit 266
7.7.2 Successes and limitations (FLOW case study) 278
7.8.1 Overview of the data collected in each case study 285
7.8.2 Comparison of students' programming use across different case studies 297

# ACRONYMS

| | |
|---|---|
| A3ES | *Agência de Avaliação e Acreditação do Ensino Superior*—Higher Education Assessment and Accreditation Agency |
| AMBER | Codename for a higher education institution |
| AMP | Audiovisual and Media Production |
| BLUE | Codename for a higher education institution |
| CAI | Computer-assisted instruction |
| CAD | Computer-aided design |
| CIL | Computer and information literacy |
| CLT | Cognitive Load Theory |
| CNAEF | *Classificação Nacional das Áreas de Educação e Formação*—National Classification of Education and Training Fields |
| CS | Computer Science |
| CSS | Cascading Style Sheets |
| CT | Computational thinking |
| CUP | Curricular unit plan |
| CTESP | *Curso técnico superior professional*—a technical-vocational course |
| DGEEC | *Direção Geral de Estatísticas em Educação e Ciência*—Education and Science Statistics General Directorate |
| DGES | *Direção Geral do Ensino Superior*—Higher Education General Directorate |
| FLOW | Codename for a curricular unit / case study |
| GDPR | General Data Protection Regulation |
| GRAPHIC | Codename for a curricular unit / case study |
| GUI | Graphical user interface |
| HTML | Hypertext Markup Language |
| ICILS | International Computer and Information Literacy Study |
| IDE | Integrated development environment |
| ISCED | International Standard Classification of Education |

| | |
|---|---|
| IT | Information Technology |
| ICT | Information and Communication Technology |
| LIME | Codename for a higher education institution |
| LMS | Learning Management System |
| MOOC | Massive Open Online Course |
| PCA | Principal component analysis |
| PHP | PHP Hypertext Preprocessor, a programming language |
| PRIMER | Codename for a curricular unit / case study |
| RED | Codename for a higher education institution |
| ROSE | Codename for a higher education institution |
| RPG | Role-playing game |
| SKETCH | Codename for a curricular unit / case study |
| STEM | Science, technology, engineering, and mathematics |
| TAM | Technology Acceptance Model |
| TRA | Theory of Reasoned Action |
| TOUCH | Codename for a curricular unit / case study |
| UTAUT | Unified Theory of Acceptance and Use of Technology |
| WEAVE | Codename for a curricular unit / case study |

*(blank page)*

*(blank page)*

# 1

# *Introduction*

> Computers aren't the thing; they're the thing that gets us to the thing.
> —*Fictional character Joe McMillan in* Halt and Catch Fire*, season 1, ep. 1.*

## 1 MY PATH AND MOTIVATION

In the summer of 1986, I remember spending one Sunday afternoon in the company of my older cousin, typing alchemical BASIC instructions in a Timex TC-2068 hooked up to the living room television. Step after step, we managed to get the computer to display the national flags of Argentina and West Germany. My father recorded a few seconds to VHS videotape; later in the evening, he would use the rest of the tape to record the live broadcast of the Mexico World Cup final. At the age of seven, that was my first contact with creative code. It felt good.

⁂

I had a front-row seat to the critical four last decades of the process of media convergence (Jenkins, 2006). I entered primary school in a purely analog world; the thing closest to a computer in the home was my father's calculator, a sturdy machine with red LEDs which consumed a lot of batteries—not a toy!, as I was often cautioned. The Timex computer entered the home during the second grade, and despite a few creative exploits—which mainly consisted of copying program listings from books and magazines and tweaking them to make suitable introductions to videotape recordings—gaming was the primary use. I spent entire afternoons playing Chuckie Egg, Jet Set Willy, or Bomb Jack, or otherwise tuning the cassette tape recorder when games did not load. After getting a Commodore Amiga A600 for Christmas while in the eighth grade, I felt I had a 'proper' computer—one with a disk drive, an operating system with a graphical user interface (GUI), and software tools such as Deluxe Paint for drawing and Soundtracker for making music. There were also a lot of games for the Amiga. With secondary school came the procession of Windows PCs that lasts to this day, after the initial 'excuse' of needing to keep up with the school's IT classes about Microsoft Office and Corel Draw; still, without an internet connection neither at home nor at school, the Microsoft Encarta CD-ROM felt like a wonder of the world.

After a first unsuccessful stint at college, I became interested on the internet while looking for an interesting subject to learn. Even before having an internet connection at home, I started learning HTML from magazine articles and designed a webpage about the game Sensible Soccer, which I uploaded to GeoCities from a local cybercafé. I then enrolled at a short vocational multimedia course to pursue web design and worked as a freelancer for a couple of years. In contrast with today's complexity, a single book like *Webmaster in a Nutshell*[*] could include almost everything a budding web designer at the turn of the millennium needed to know. One chapter introduced the PHP programming language, and I found it easy to learn. After all those years away from code, under the spell of the Amiga Workbench and the Windows GUIs, I again saw a vast expanse of possibilities before me, the same potential held by that blinking **K** in the Timex BASIC command prompt. I may have dedicated this newfound alchemical knowledge to create things like a CD catalog and or a weblog backend, but I was proud, nonetheless.

In 2001, a parallel passion for film led me to enroll in the Film and Video licentiate course at the Escola Superior Artística do Porto (ESAP), a private polytechnic college. Upon graduation in 2005, the course coordinator invited me to remain as a visiting instructor and teach a new curricular unit about non-linear video editing; the matter had been absent from the Film and Video curriculum, and I had learned it while working on the course's short films and documentary projects, apparently well enough to teach it. I accepted the invitation and first faced the question that will later surface throughout this thesis: *"how do you teach something you have learned by yourself?"* Moreover: how would I teach? At the age of twenty-six and not much older than the students, the first year proved difficult. Still, I persevered and would continue leading the video editing unit all over my ten-year stint at ESAP. Meanwhile, I worked as a freelance videographer, but also took to further my academic qualifications.

While taking the FEUP Master in Multimedia (2008-2010), I sought to improve my computer programming skills. Coursework projects offered opportunities to learn and employ ActionScript and Processing. I was also able to use my PHP experience to develop a digital portfolios platform intended for arts education, which was the theme of my master's thesis. After completing the master's, I was invited to teach a theoretical unit about digital arts to ESAP's Multimedia and Communication Design students. After a couple of years, that course's coordinator asked me whether I would rather teach an introduction to computer programming with Processing. I accepted, again thinking, *"how do you teach something you have learned by yourself?"* After one single year, curricular changes implied a return to the previous theoretical syllabus I taught to the Design students. Still, I was asked to keep teaching an introduction to Processing to students enrolled in the Visual Arts and Photography course, which I did during my two final years teaching at ESAP.

In the meantime, I applied to an opening to teach an introduction to programming in the Communication Design course at the Fine Arts Faculty of the University of Porto (FBAUP). After obtaining the assignment, I met with the course coordinator, who offered me advice about the large group of students—over sixty divided into multiple classes—I would be meeting. I was told that the students would be of two minds about

[*] Spainhour & Eckstein (1999). *Webmaster in a Nutshell*, 2nd edition. O'Reilly.

computer programming: they would either be enthusiastic about it or would reject the matter. Though the ESAP classes I had experienced teaching to were much smaller—around ten or twelve students—I agreed with the coordinator's assessment. I told him I would do my best. I believe I did, though I was teaching something I had learned myself. Moreover, I had learned to teach all by myself.

⁂

Why do I tell these stories? By the time I started at FBAUP, I was already enrolled in the Doctoral Program in Digital Media. After my previous work on digital portfolios, I began with a research focus on e-learning platforms and Massive Open Online Courses (MOOCs) for arts education, which slowly crystallized into a more specific interest in specialized educational software. Still, progress was slow; my doctoral studies took a backseat to my teaching and my freelance videography practice. Yet, decisions had to be made at some point. After leaving ESAP, I decided it was high time to commit to the Doctoral Program. I applied for a scholarship that would imply a full-time commitment to research. After receiving the grant, I bade farewell, for the time being, to my teaching engagements.

At the time, I had a sprawling collection of notes and, worse, a fluid collection of ideas about my research possibilities. Over the long meetings with my advisors that ensued, I had to engage in a process akin to renovating and cleaning up a home: which research questions brought me joy? I realized the pertinent questions had been written on billboards along my teaching path.

## 2 PURPOSE AND AUDIENCE

> Probably the greatest reason that technology has taken over from art is because art was always technology anyway. (Levine, 1971/2011, p. 55)

Watching a YouTube video of Douglas Engelbart's *Mother of All Demos*,[*] the historic presentation at the San Francisco Civic Auditorium in December 1968 looks and sounds like a transmission from an alternate, more ideal universe. This researcher is reminded of Alfred North Whitehead's observation[†] that most Western philosophy is just a set of footnotes to Plato; analogically, the past fifty years of digital media development might be footnotes to Engelbart's live demonstration.

Yet, computers are not the thing. Mass affordability, accessibility, and ubiquity wield out digital media as what Timothy Morton (2013) names an 'hyperobject,' an inconceivable infrastructural layer upon human endeavors. Artists and designers may have their hand in its development, but the artists are also those who can meet it with 'impunity,' as McLuhan (1964/2008) stated, and best interpret the 'hyperobject.' Here lies the importance of learning to code—to develop new technological refinements, sure, but also to grant those who can make the most sense of the 'matrix' the necessary literacies to decode the 'matrix.'

---

[*] youtu.be/B6rKUf9DWRI

[†] Found in Whitehead, A. N. (1929). *Process and Reality*.

We settled on a simple main research question that will propel this thesis forward: *how* is computer programming taught and learned by artists and designers? As we shall examine in detail in chapter **5**, this researcher's prior teaching experience brings an 'insideness' (cf. Sikes & Potts, 2008) that, while unarguably aiding navigation of the field, may bring along unconscious biases. Yet, the researcher believes his experience in video documentary and reporting may have contributed to the needed distancing; one ideal version of this inquiry would exist, at least partially, as a documentary series.

⁂

Though the scope of the inquiry may appear narrow—as it should—we believe this thesis will appeal to a broader audience.

To policymakers, institutions, and curriculum designers, we will contribute findings and recommendations relating to the curricular placement of computer programming instruction, the working conditions inside the computer arts studio, and the interplay between class size and face-to-face time, and learning assurance.

To the educators committed to teaching computer programming to arts and design classes, we will offer findings and recommendations relating to syllabi, pedagogical strategies, evaluation methodologies, and learning management.

To educational researchers, we will contribute findings from a two-phase mixed-methods research plan, merging quantitative technology acceptance surveys with a collective case study of distinct curricular units.

To computer scientists, we will offer pertinent findings about how computing is taught in other disciplines, the tools used, and the needs of the arts and design field.

Finally, to uncommitted artists and designers, we will portray how computing is made relevant and taught to their peers.

## 3 THESIS STYLE

In producing this thesis, we followed the recommendations and guidelines presented in the *Publication Manual of the American Psychological Association,* 7th edition (2020) and Robert Bringhurst's *The Elements of Typographic Style*, version 3.2 (2008). We pondered over those situations where the guidelines differed and made a few minor compromises to ease our use of Microsoft Word and shorten the time we would spend in typesetting. Regardless, we aimed to produce a readable, good-looking, and 'user-friendly' document to the best of our abilities.

Throughout this thesis, we will set chapter references in bold type (e.g., chapter **2** will follow); yet, when referring to divisions within the same chapter, we will employ the section symbol (e.g., you are reading subsection §3). We will also use three numbers to identify tables and figures, where the first two respectively refer to their chapter and section (e.g., Figure 5.3.1 is located in **5**.3).

## 4 THESIS ORGANIZATION

Including this introduction, we divided this thesis into nine chapters. The remaining eight chapters correspond to four main parts:

- PART ONE We will contextualize our research and review the pertinent literature. We invite our kind reader to imagine a two-set Venn diagram where the fields of computing and higher education intersect. Chapter **2** will focus on that intersection, reviewing historical foundations, recent trends, pedagogical practices, and current debates. We will then add a third set to our Venn diagram. Chapter **3** will address how the arts [*] intersect with computing and higher education, reviewing the specific foundations, practices, and debates. We then change our lens and zoom in far into the center of our diagram until the edges are invisible and we face a panorama. Chapter **4** will surveil the territory of the Portuguese public higher education system and review the inclusion of computer programming into arts and design undergraduate curricula.

- PART TWO We will present the strategies and the tools we employed to explore beyond the gaps we identified in our knowledge. Chapter **5** will address our research goals and describe our theoretical frameworks, our research design, and our research methods and procedures.

- PART THREE We will then report our findings. Chapter **6** will present the results of a preliminary survey of students enrolled in arts and design courses that include computer programming in their curricula, as well as a parallel survey of professors holding teaching assignments at those courses. These findings informed our approach to the main stage of our research. Chapter **7** will describe and present our findings from a collective case study of six units about computer programming that were a part of six distinct licentiates in arts and design fields.

- PART FOUR Finally, we will engage with our findings and bring this inquiry to a conclusion. Chapter **8** will place our research results underneath the light of our literature review and discuss the most reactive aspects. Chapter **9** will summarize our most salient findings and present our recommendations for stakeholders and our suggestions for future research.

[*] Here understood broadly, i.e., from fine arts to applied arts (e.g., design or architecture).

*(blank page)*

PART ONE

# *ECHO ("CONTEXT");*

*(blank page)*

# 2

# *The convergence of computing and education*

★ SYNOPSIS We present a historical overview and the present state of discussions surrounding the interrelation of computing and educational practices, with a focus on higher education. In the first section, we present significant historical developments in that regard and summarize the recent trends in technology adoption which are salient to education. In section §2, we focus on critical understandings of the convergence of digital media and education, as well as of the new literacies borne out of computer technology. Finally, in section §3, we address the present state of computing education research and practices.

## 1 HISTORICAL OVERVIEW AND RECENT TRENDS

> Computing is one of the most ahistorical disciplines in the sciences [... a] series of ongoing failed lessons, where instead of building on knowledge of successes and failures past, we just trace the same steps in a kind of random walk with whatever is contemporary and thrust upon us by virtue of habit or some mighty deal that gets done at your institution. — Ian Bogost, interview (Marzoni, 2020)

On the surface, the history of computer technology may appear one of logarithmic progress in computing power, affordability, and ubiquity—from multi-million-dollar mainframe computers at a few elite institutions to the mobile phones (with several mainframes' computing power) in the pockets of a large part of the population. Such a history is often neatly summarized in charts such as those presented in Figures 2.1.1 (next page), which trace the last few decades' progress in the efficiency of computing power and the affordability of computing components. Yet, as media and computing scholar Ian Bogost had stated in that interview, juxtaposed with the progress in silicon circuit etching, a more profound history of computing applications appears to run in cycles, oscillating between periods of amnesia and rediscovery.

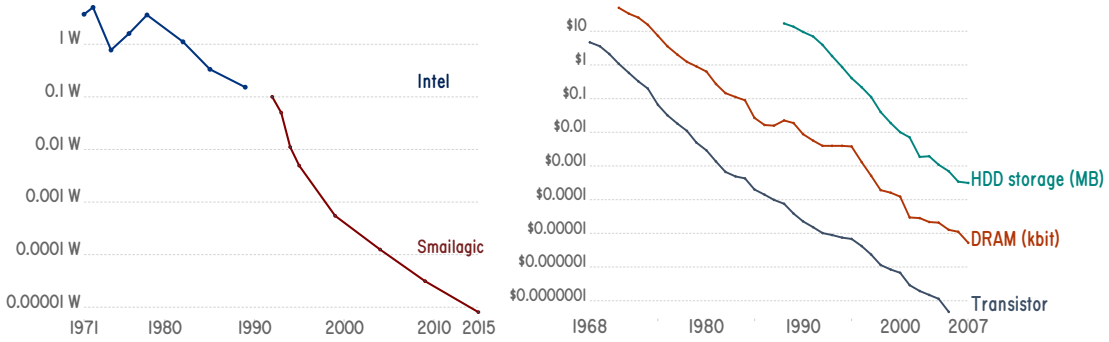



***Figure 2.1.1a*** *(left). Computer processors' efficiency, in watts per million instructions per second (Kurzweil, 2015).*
***Figure 2.1.1b*** *(right). Computer components' cost (in US dollars): transistors, DRAM memory per kilobit, hard disk storage per megabyte (Roser & Ritchie, 2013).*

The history of computing has been anything but linear. On the one hand, simple applications like text editors appear to devolve, and keyboards anecdotally seem less responsive than they used to be at the dawn of the personal computing era. On the other hand, internet connectivity made the progress in the cost and performance of individual machines portrayed in Figures 2.1.1 appear quaint—we routinely leverage computing capabilities orders of magnitude superior to that of the machines in our desktops, merely to search for cooking recipes on the internet.

⁂

Computer history is a paradoxical history of fantastic progress and wasted potential, a history that is both celebratory of technological achievement and insinuating that a better societal relationship with computing would have been possible, if only some historical details had been different: what if Ted Nelson had released a working Xanadu system before Tim Berners-Lee released the WorldWideWeb browser? What if, as the field crystallized, institutions and the industry had perceived computer science as the province of women rather than men? What if LOGO had been accepted as integral to basic schooling curricula worldwide? What if...?

In the following subsection, we shall summarize key aspects of the history of computing in its relationship with education, with a particular focus on critical developments that evince its still-unexplored potential. In subsection §1.2, we shall then take a look at some of the recent trends in technological adoption.

## 1.1 *Crucial historical developments*

Heimbürger and Multisilta (2004) situate a *time of mainframes* from the mid-1940s to the mid-1970s. The authors note events with educational implications, which include Vannevar Bush's ideation of a hypermedia system—the Memex (Bush, 1945)—the earliest computer-assisted instruction systems in the 1960s, Ted Nelson's invention of the word 'hypertext' and specification of the Xanadu system (Nelson, 1974/1987), Douglas Engelbart's demonstration of the oNLine System (cf. Engelbart, 1995), and the invention of the LOGO programming language for children (cf. Papert, 1985).

Though mainframe computers cost hundreds of thousands or millions of dollars and required reinforced floors and specialized air conditioning at their sites, these researchers all understood what was soon to follow. Though writing prior to the availability of the first home computers, Ted Nelson stated, *"EVERYBODY SHOULD UNDERSTAND COMPUTERS"* in his introduction to the *Computer Lib* volume, and then proceeded to describe computer illiteracy as *"a silly and dangerous ignorance"* (Nelson, 1974/1987). Nelson, it is worth noting, really did mean *everybody* when he wrote those capitalized words. In *Dream Machines*, Nelson noted the inevitability of the personal computer but also how crucial the shape of its hardware and software was going to be:

> My special concern, all too tightly framed here, is the use of computers to help people write, think and show. But I think presentation by computer is a branch of show biz and writing, not of psychology, engineering or pedagogy. This would be idle disputation if it did not have far-reaching consequences for the designs of the systems we are all going to have to live with. At worst, I fear these may lock us in; at best, I hope they can further the individualistic traditions of literature, film and scholarship. But we must create our brave new worlds with art, zest, intelligence, and the highest possible ideals. (Nelson, 1974/1987)

During the mid-1970s, that transitional time described by Heimbürger and Multisilta (2004) as a *time of workstations* (or minicomputers), Alan Kay and Adele Goldberg worked at Xerox's Palo Alto Research Center to realize precisely such a system of *"presentation by computer,"* the Dynabook. Kay and Goldberg prototyped graphical user interface (GUI) software programs underpinned by Smalltalk, a language they devised, all meant to be usable by school-age children. The duo noted:

> If such a machine were designed in a way that *any* owner could mold and channel its power to its own needs, then a new kind of medium would have been created: a metamedium, whose content would be a wide range of already-existing and not-yet-invented media. (Kay & Goldberg, 1977/2003)

Designing a high-level programming language easy enough for children was not a new idea at the time. Ever since John Mauchly and William Schmitt designed Short Code in the late 1940s, expanding the accessibility of computer programming had been one of the main objectives of language designers. In 1955, Grace Hopper led the development of FLOW-MATIC as the first language to employ English words and expressions; in 1957, John Backus designed FORTRAN to meet the needs of scientists and mathematicians without specialized knowledge of computing; likewise, COBOL, first released in 1959, expanded on Grace Hopper's work in order to meet the needs of business users. In 1964, John Kemeny and Thomas E. Kurtz released BASIC, a general-purpose language with ease of use in mind, as the name implies. A language designed for children thus appears as the inevitable outcome.

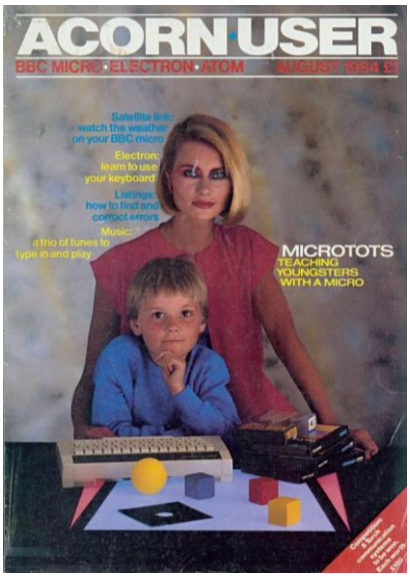
As narrated in *Tools for Thought* (1985), Howard Rheingold's account of the first three decades of computer technology, Seymour Papert spent five years working with Jean Piaget in Switzerland and, after the latter's notion that children are 'scientists,' came to believe computers would be an appropriate tool for children to conduct their research (Rheingold, 1985/2000, p. 239). In 1967, Papert, alongside Wally Feurzeig and

Cynthia Solomon, designed a programming language designed for children's exploration of computer graphics—LOGO. The event possibly marks the first articulation of the development of computational thinking (a notion we shall examine in the next subsection) as an educational objective. As Papert stated:

> The child programs the computer. And in teaching the computer how to think, children embark on an exploration about how they themselves think. The experience can be heady: Thinking about thinking turns the child into an epistemologist, an experience not even shared by most adults. (Papert, 1980, p. 19)

This development is especially noteworthy as the 1960s also saw the birth of the first computer-assisted instruction (CAI) systems, such as PLATO at the University of Illinois, which at first was a platform for simple 'drill and practice' educational software, antipodal to the constructionist pedagogy underpinning LOGO. Ted Nelson stated, *"computer-assisted instruction, applied thoughtlessly and imitatively, threatens to extend the worst features of education as it is now"* (Nelson, 1974/1987). Though nowadays students are able to autonomously explore a hypermedia system—the internet—as Nelson idealized, it is striking how much contemporary educational resources and software still borrows from the constrained sequential approach Nelson so decried in the computer-assisted education systems of the early 1970s.

Heimbürger and Multisilta (2004) place a *time of personal computers* from the early 1980s, at the dawn of the general availability of personal computers, such as the eponymous IBM PC and assorted 'beige box' clones which populated offices worldwide, but also affordable home computers such as the Apple II, the Commodore C64, or the Sinclair ZX Spectrum which sold by the millions in Western Europe, North America, and Japan. Despite these machines' frequent use as videogame consoles, their educational potential was accompanied by a growing industry in computing manuals and magazines aimed at children, teenagers, or their parents:



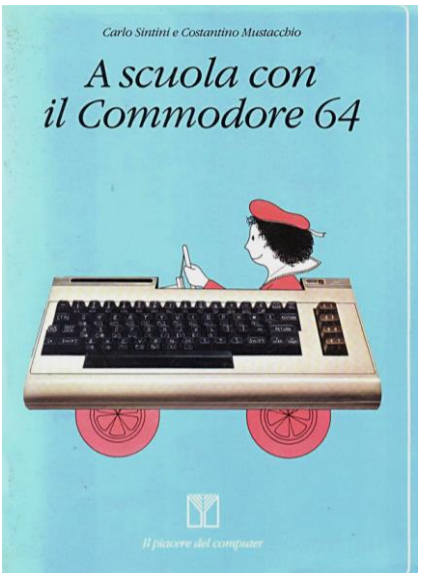




***Figure 2.1.2a*** *(left). Quinn [Ed.] (1984).* Acorn User Magazine, *25. Redwood Publishing.*
***Figure 2.1.2b*** *(right). Sintini & Mustacchio (1984).* A Scuola con il Commodore 64. *Franco Muzzio Editore.*

One emblematic initiative was the British BBC Computer Literacy Project which coordinated the production of television programs about using computer programming in educational contexts with the release of the Acorn BBC Micro family of affordable microcomputers (see Figure 2.1.2a), which were bought in large numbers by schools and families alike (cf. Selwyn, 2002). Heimbürger and Multisilta (2004) emphasize some of the developments in educational hypermedia of the time, such as the release of the Apple HyperCard (an influential end-user programming software) in 1987 and the growing concerns with usability research. These new developments were made possible by the availability of a new generation of machines with multimedia capabilities and GUI operating systems from the mid-1980s onward—e.g., the Apple Macintosh, the Commodore Amiga, or the Atari ST, alongside the Windows operating system for standardized PCs.

The broad dissemination of graphical user interfaces was perhaps instrumental in Tim Berners-Lee's release of the first web browser and HTML authoring software in 1990 after his earlier specifications for the World Wide Web. Heimbürger and Multisilta (2004) situate the beginning of a new era, lasting until the early 2000s, in that event. Writing in 2004, the authors situated their present time as the *time of the mobile Web and semantic Web*. We posit the authors lacked historical distancing, as neither the mobile Web nor the semantic Web materialized as they had envisioned. Still, we may find some notable technological events that serve as starting points to the latter eras of the history of computing and education.

The semantic and interoperable Web turned out to be a mirage. In the 2000s, the Web grew centralized around a few publishing platforms just as broadband connectivity was made more accessible to large parts of Western countries' populations. This brought the rise of weblogs (e.g., Blogger or Tumblr) and platforms for sharing photos (Flickr), videos (Vimeo and YouTube), and audio (Soundcloud). The social networking sites (e.g., Facebook) that begun as aggregators did eventually cannibalize the functionalities of the extant platforms. A latter era started in the turn of the 2010s, with the launch of Apple's iPhone App Store and the popularization of mobile devices running the Android operating system. The mobile Web became a secondary functionality, as the ubiquitous internet access in every pocket was centered on 'walled' applications. Web platforms made the transition to a focus on mobile use. Some floundered (weblogs), some were beaten by newer competitors (e.g., Flickr vs. Instagram), some grew to incredible power and influence (Facebook). What is apparent from the last fifteen to twenty years is a move away from the potential of hypertext, notwithstanding the commendable project that is Wikipedia. The movement toward a reenactment of older media forms over digital technology,[*] which we shall address in §2.1, coincided with a dramatic increase in connectivity and technology adoption.

[*] It is telling that one of the great novelties of the pandemic year has been the return of email newsletter publishing via platforms such as Substack.

## 1.2 *Recent technology adoption trends*

Using the website *Our World in Data*, we produced the following figures, which compare the proportions of individuals using the internet and subscribing to mobile phone networks since 1990 in Portugal, the entirety of the European Union, the United States, and a group of countries classified by the United Nations as the least developed.

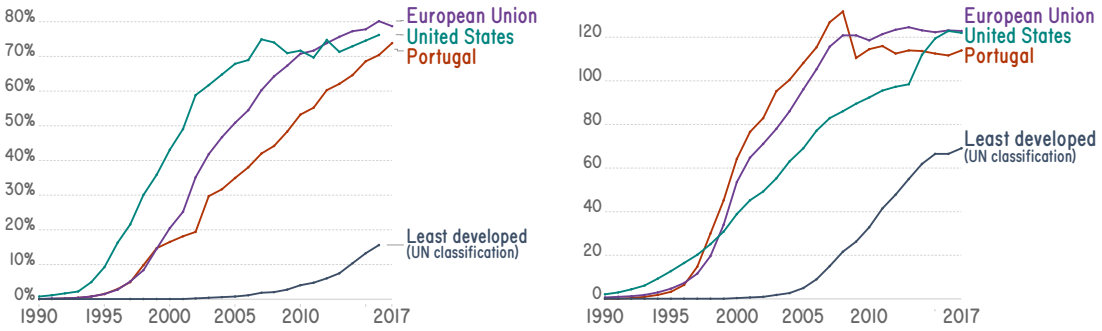



***Figure 2.1.3a*** *(left). Share of individuals using the internet* (Ritchie & Roser, 2017).
***Figure 2.1.3b*** *(right). Cellular subscriptions per 100 people* (Ritchie & Roser, 2017).

The share of the population using the internet in Portugal—73.8% in 2017—was converging with the United States' (76.2% in 2016) and the European Union's average (78.8% in 2017). As of 2016, only 15.7% of the population of the least developed countries had used the internet in the last three months (Ritchie & Roser, 2017).

Greater worldwide convergence is found in the mobile subscriber figures. In 2017, there were 69 such subscriptions per 100 people in the least developed countries; the worldwide average already exceeded one subscription per person (104/100). In Portugal, there were 114 such subscriptions per 100 residents; there were 122 in the United States and 123 across the European Union (Ritchie & Roser, 2017). We note there was a peak of 132 mobile subscriptions per 100 Portuguese people in 2008, which was possibly due to endogenous market factors.[*]

### 1.2.1 COMPUTER AND INTERNET ADOPTION TRENDS IN PORTUGAL

Since the turn of the millennium, 16 to 24-year-old have been the greatest adopters of computers[†] and the internet in Portugal, as shown in Figure 2.1.4 (PORDATA, 2019c). In 2002, 55.2% of residents in that age group used a computer, and 42.8% had used the internet,[‡] yet adoption grew dramatically through the 2000s. By 2009, 92.2% of people in that cohort used a computer, and 88.1% used the internet. Internet use in that group exceeded 99% in 2015 and rested at 99.5% in 2019.

[*] E.g., the prices of calls made to phones outside one's provider.

[†] The authors of the surveys defined computers as either desktop PCs, workstations, laptops, or computer terminals (i.e., excluding game consoles, palmtops, smartphones, and tablets).

[‡] By comparison, among 25 to 34-year-olds, only 40.2% used a computer and 30.2% connected to the internet in 2002.

We note the growing influence of mobile phones: computer use among 16 to 24-year-olds peaked in 2015 (98.1%) and contracted to 95.0% in 2017.*

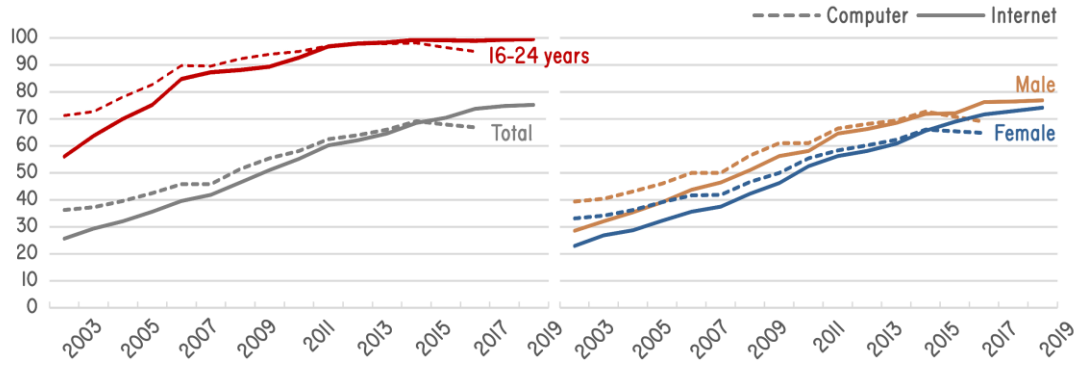



***Figure 2.1.4.*** *Computer and internet users in Portugal (%), by age (left) and by gender (right), from 2002 to 2019* (PORDATA, 2019c).

As shown above, a slightly greater percentage of men have adopted computers and the internet. We have not been able to cross-compare adoption across age groups, though we note it is likely that difference reflects women's greater life expectancy.†

Computer and internet use patterns closely match their availability in Portuguese households, charted in Figure 2.1.5:

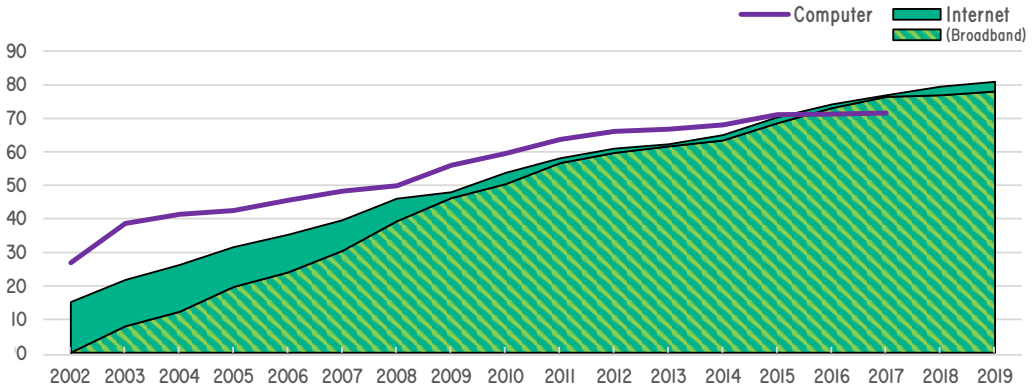



***Figure 2.1.5.*** *Households (%) with a computer and internet service* (PORDATA, 2019a).

Whereas in 2002 only a little over 40,500 (51.1% of the total) computers in Portuguese basic and secondary schools had internet connectivity, by 2010 over 622,000 (90.6%) computers in those schools were connected to the internet, which translated to one such computer for every 2.2 students. A decline followed, and by 2019 there were just over 246,000 computers with an internet connection in Portuguese schools: one for every 4.9 students. This decrease was the most expressive in 1st cycle (i.e., primary) schools: from one computer for every 1.1 students in 2010 to one for every 6.4 students in 2019. Across establishments dedicated to the 2nd cycle (5th and 6th year), 3rd cycle (7th to 9th year), and secondary schooling, the number of students per computer remained stable throughout the decade (PORDATA, 2019b).

---

* Internet use also surpassed PC use across the other age groups. As of 2017, 66.8% of all residents in Portugal used a computer, but 73.8% used the internet.

† According to the 2011 Portuguese Census, there were 132 women over the age of sixty for every 100 men (INE, 2011).

It may be the case that the austerity policies after 2011 led to fewer investments in computing equipment, under greater wear and tear in primary classrooms. It may also be the case that a more significant portion of those students began to take their own devices to school. And finally, it may also be the case that the idea of placing one computer in each desk in primary school classrooms was no more than a manifestation of the excessive technological optimism of the 2000s, as symbolized by the One Laptop Per Child initiative (backed by, among others, Seymour Papert and Alan Kay) or Intel's Classmate PC (manufactured in Portugal under the name 'Magalhães').*

* In *The Diamond Age* (1995), science-fiction author Neal Stephenson tells the story of how a poor young girl finds a sort of educational tablet computer intended for the children of the elite, and how, tutored by the device, the young protagonist eventually leads a revolution. Though the author of this thesis enjoyed this sci-fi fairy-tale, he is skeptical about the transferability of its premise to the real world.

> If the formative power in the media are the media themselves, that raises [...] that technological media are staples or natural resources, exactly as are coal and cotton and oil. Anybody will concede that a society whose economy is dependent upon one or two major staples [...] is going to have some obvious social patterns of organization as a result. (McLuhan, 1964/2008, p. 22)

Marshall McLuhan's adaptation of technological determinism to media, positing the introduction of new media technologies as leading a chain of causality, was criticized early on. Raymond Williams, using the example of television, argued that technologies have a social history, and so do the uses of that technology (Williams, 1972/2003). Their contemporary, John Culkin, elegantly summarized the need to recognize the interplay between the social and the technological with his oft-quoted observation that *"we shape our tools, and thereafter our tools shape us"* (cf. Hurme & Jouhki, 2017). Perkins (1985) reached a similar conclusion in his discussion of the interrelations between educational technologies and students' thinking.

Nonetheless, the specter of technological determinism still haunts our present, most often in the naïve belief that technologies can be devised in ways as to steer their consequences—what Evgeny Morozov termed as 'solutionism' (cf. Morozov, 2011). Rather than a pool cue that sets the billiard balls in motion in more or less precise angles, Bruce and Hogan emphasize an organic nature to media technologies:

> Literacy technologies, such as pen and paper, index cards, computer databases, word processors, networks, e-mail, and hypertext, are also ideological tools; they are designed, accessed, interpreted, and used to further purposes that embody social values. More than mechanistic, they are organic, because they merge with our social, physical, and psychological beings. (Bruce & Hogan, 1998)

After Bruno Latour's observation that *"technology is society made durable"* (1991), Bruce and Hogan call for a new literacy that includes *"reading' the world, and the technological artifacts within it"* (1998). We will review the subject of new literacies later, in subsection §2.3. But first, we shall explore the notion of media convergence, which was nonetheless foreshadowed by McLuhan:

> Energy and production now tend to fuse with information and learning. Marketing and consumption tend to become one with learning, enlightenment, and the intake of information. This is all part of the electric *implosion* that now follows or succeeds centuries of *explosion* and increasing specialism. (McLuhan, 1964/2008, p. 383)

Despite the mechanical neatness of McLuhan's technological determinism, his allusions of explosive processes betray more chaotic, turbulent dynamics. We shall address that *"electric implosion,"* the media convergence phenomenon, in the following subsection; then, in §2.2, we shall turn our focus to how the process of media convergence has been expressed in the field of higher education.

## 2.1 *Media convergence*

Out of a belief the unopposed ascendancy of Western capitalism in the early 1990s was the inevitable result of a deterministic dynamic, political commentators such as Francis Fukuyama decreed *The End of History* (the title of his 1992 book). The 'digital revolution' had yielded a 'global village.' The present had given way to the 'Future,' even if, after the famous aphorism of William Gibson, that 'Future' was not yet evenly distributed. That unevenness turned out to be key, the Forever Technocracy was a mirage, and History was far from over.* Indeed, some saw the paradoxes between the lines of glossy WIRED magazines and colorful music videos for Peter Gabriel CD-ROM releases:

> The global village, as the site of the culture of capitalist postmodernity, is a thoroughly paradoxical place, unified yet multiple, totalized yet deeply unstable, closed and open-ended at the same time. [...] a chaotic system, where uncertainty is a built-in feature. (Ang, 1994, p. 194)

Though anathema to the early 1990s triumphalist ethos, *uncertainty* was the actual way of the future. In less than a decade, Donald Rumsfeld—the United States' Secretary of Defense responsible for the 2003 invasion of Iraq—would put the expression 'unknown unknowns' into the popular lexicon. In few years, History had reawakened as a concept, and chaotic uncertainty was perhaps the only 'known known.' The *implosion* McLuhan had alluded to had revealed its turbulent nature. In his 2006 book *Convergence Culture*, Henry Jenkins refers to a phenomenon of *media convergence*:

> [...] where old and new media collide, where grassroots and corporate media intersect, where the power of the media producer and the power of the media consumer interact in unpredictable ways. (Jenkins, 2006)

Jenkins was more directly inspired by the work of Ithiel de Sola Pool, who in the early 1980s had described this possible outcome of digitization in the early 1980s by looking at media concentration in the conglomerates of the time. As Jenkins states it, Sola Pool predicted: *"a period of prolonged transition, during which the various media systems competed and collaborated, searching for the stability that would always elude them"* (Jenkins, 2006, p. 11).

Indeed, the three decades since have resembled Sola Pool's prolonged transition rather than a swift coup of 'new media' over 'old media.' Though examples abound fitting the 'digital revolution' narrative—e.g., Blockbuster (video rentals) going bankrupt after Netflix launched its streaming service—there are far more interesting examples of the unpredictable nature of media convergence: for instance, the recent (December 2020) announcement that HBO would release all Warner Brothers' movies in its streaming service simultaneous with their theatrical release is a case of two 'old media' companies† agreeing to a business move that may definitely torpedo the movie theatre industry.

---

* One ought to appreciate the irony of how Microsoft's apotheotic release of the Windows 95 operating system was driven by a marketing campaign centered on the *Start* menu, Rolling Stones' *Start Me Up* included.

† HBO, which started as a TV channel, is owned by AT&T, the direct descendant of the Bell Telephone Company.

It can be argued that, regardless of detours, movement toward media convergence points to a deterministic outcome. After all, we know how an implosion ends, regardless of how it happens. Still, Jenkins admonished against the 'black box fallacy,' which contends a single device will crystalize the end result of media convergence. Though the smartphone appears to come close to such a deterministic vision, Jenkins is careful to distinguish between *content* and *platforms*, citing a 2002 media industry report pointing towards *content convergence* alongside *platform divergence*—a statement that perfectly captures the media consumption landscape of 2020. Hence, Jenkins is more interested in Lisa Gitelman's (2006) notion of media as including protocols—after all, the protocol of going to the movies theatre remained the same despite the replacement of 35mm film by video projection over the last decade:

> If media include what I am calling protocols, they include a vast clutter of normative rules and default conditions, which gather and adhere like a nebulous array around a technological nucleus. Protocols express a huge variety of social, economic, and material relationships. So telephony includes the salutation "Hello?" (for English speakers, at least), the monthly billing cycle, and the wires and cables that materially connect our phones. (Gitelman, 2006)

We read Jenkins as observing such protocols have their own momentum and inertia. Hence, the case is not so much of Netflix 'disrupting' Blockbuster's business, but instead of Netflix improving the existing protocol of television channel-surfing and, in doing so, capturing the attention of enough additional customers as to render the video rental business unprofitable. Of course, this has societal and cultural consequences, for the video store was an affinity space that decisively shaped conversations and practices about filmmaking. Jenkins was indeed more concerned about these consequences than the business disruption. Moreover, a salient feature of convergence is the eroding of the boundaries between media producers and media consumers, which we have witnessed in the last decades, and which has accelerated after the democratization of the smartphone and ubiquitous internet access.

Predicting our present debates on the regulation of social networks, Jenkins wrote of an uneasy tension between a world without gatekeepers and one in which gatekeepers would have unprecedented power (Jenkins, 2006, p. 18). Other contemporary critics were concerned about what an unmoderated participatory culture could entail. In 2005, Nicholas Carr already foreshadowed the issues that became salient with the post-2016 cycle of populist politics and became near-catastrophic during the COVID-19 pandemic:

> All the things Web 2.0 represents—participation, collectivism, virtual communities, amateurism—become unarguably good things. [… Web 2.0 promoters] venerate the amateur and distrust the professional. We see it in their praise of Wikipedia, and we see it in their worship of open-source software and their promotion of blogging as an alternative to 'mainstream media.' (Carr, 2005)

Carr's position of how unmoderated participation leads to cultural impoverishment (above paradoxically expressed in a blog post) would be the basis for his 2010 book *The Shallows*. A similar argument is expressed by philosophers Hubert Dreyfus and Sean

Dorrance Kelly in their 2011 book *All Things Shining*, with the authors placing an additional emphasis on how technology impairs skill development:

> But the improvements of technology are impoverishments as well. The GPS covers over the meaningful distinctions that the art of skilled navigation revealed. To the extent that technology strips away the need for skill, it strips away the possibility of meaning as well. To have a skill is to know what counts or is worthwhile in a certain domain. Skills reveal meaningful differences to us and cultivate in us a sense of responsibility to bring these out at their best. To the extent that it takes away the need for skill, technology flattens out human life. (Dreyfus & Kelly, 2011)

Going back to Gitelman's (2006) notion of media as including historical and social protocols, one particular salient feature of the process of media convergence, and which we alluded to in the previous section, is how digital media applications have tended to refashion existing media forms—phenomena which Jay David Bolter and Richard Grusin (2000) have termed as 'remediation.' *"Software is eating the world"* (Andreessen, 2011), but the digested world appears not much different from before as long as one does not take a look under the surface. Computing technologies have either been massively consequential or just a little more convenient. Starting from the decades-old positions of Ted Nelson or Alan Kay (see §1.1), Omar Rizwan argues we are wasting the potential of technology:

> Computers have mostly been used to duplicate existing media so far: 'TV, but more flexible'; 'Books, but easier to distribute.' The only works which really depend on computing technology today are video games, but making games demands serious engineering effort—imagine if an author had to design fonts, typeset, bind, and print their books by hand. (Rizwan, 2015)

Setting aside that even the author's point about videogames can be arguable, Rizwan illustrates a state of affairs in which dreams of computing that truly 'augments human intellect' (cf. Engelbart, 1995) remain distant. That the distinction between 'old' and 'new media' became meaningless, as Sarah Kember and Joanna Zylinska argued in *Life After New Media* (2012), testifies to the hegemony of remediation. In the field of education, we see how 'e-learning' lost any specific meaning (Paiva, Morais, Costa, & Pinheiro, 2015). In the arts, we are now well in the *Post-Internet* era (Quaranta, 2015). The chaotic process of media convergence, like history, does not end as long as true innovation is possible. Nonetheless, we appear to be at a time of consolidation.

Kember and Zylinska (2012) note the process of convergence was articulated with the *"incorporation of the language of creativity by the international business-politics nexus."* They cite John Hartley, a defender of this paradigm, on how this incorporation included *"the conceptual and practical convergence of the Creative Arts (individual talent) with Cultural Industries (mass scale), in the context of New Media Technologies (ICTs) within a New Knowledge Economy, for the use of newly Interactive Citizen-Consumers"* [*] (S. Kember & Zylinska, 2012, p. 174). As we shall see, this 'buzzword' paradigm wrought significant consequences in the field of higher education.

[*] Hartley, John (2005). *Creative Industries*. Oxford: Blackwell Publishing.

## 2.2 *Higher Education in the convergence era*

> In the minds of many academics, the educational application of digital technology is an inherently forward-looking endeavor where critical analyses of the present and/or past are simply not relevant. This stems, at least in part, from the fundamental desire amongst most educational technologists to improve education through the implementation of digital technology. For many academics, then, technology and education is approached as an inherently "positive project." (Selwyn, 2015)

In the year 2000, higher education students in most developed countries were likely to get on the internet at their institutions' libraries and computer rooms. By 2010, they were likely to be able to have broadband internet at home in addition to being able to connect to the internet wirelessly at a number of commercial establishments; moreover, those better off were already likely to own a smartphone with ubiquitous internet access. Given the dramatic changes and the still unstable outlook regarding the applications and implications, many commentators either celebrated the deployment of digital technologies in education as an unarguably positive development (as noted by Neil Selwyn in the above quotation) or felt technical mediation presented an existential threat to Higher Education institutions (cf. Barber, Donnelly, & Rizvi, 2013; Christensen & Eyring, 2011; Davidson & Goldberg, 2010). Possible 'killer apps' were the Massive Open Online Courses (MOOCs), which were also celebrated by some. Writing about the emergence of MOOCs, Clay Shirky (2012) was possibly right to question the average experience of higher education, moreover when accounting for the extreme inflation of tuition fees in the United States; nonetheless, Shirky did so at the expense of dismissing the pivotal role of the proximity between students and educators:

> Every college provides access to a huge collection of potential readings, and to a tiny collection of potential lectures. We ask students to read the best works we can find, whoever produced them and where, but we only ask them to listen to the best lecture a local employee can produce that morning. (Shirky, 2012)

Convenience and minimization of human interaction appear to have been the main drivers of technological 'disruption' (e.g., Amazon deliveries, Uber Eats) in the years since Shirky wrote his blog post. Nonetheless, the author presented no evidence that telegenic video lecturers would lead students to better outcomes than a 'local employee.' Nor did Shirky address how Netflix stars taking on professorial roles, charismatically lecturing on Calculus as cued by a teleprompter, would address the motivational issues that define the higher education experience of the average student. Even if lectures were the main pedagogical method in most disciplines, it is not apparent a lack of 'production values' is the main problem of face-to-face lessons. For all its usefulness, distance education is, after all, distant. Cooler heads had been skeptical of technological mediation. Instead, they emphasized how physical proximity and social interactions are crucial elements of the higher educational experience (Agre, 2000; Noll, 2000).[*]

[*] Their point stands even if one takes a cynical view: Ivy League universities are especially attractive for the social networking opportunities. Closer to home, meeting people that want to make movies is the best reason to attend film school.

Nonetheless, Shirky made the vital point that *"the possibility MOOCs hold out is that the educational parts of education can be unbundled"* (2012) and that such unbundling could take higher education to a population previously shut out from the system. This idea found its way into think tanks aiming to shape public policy (e.g., Barber et al., 2013), and indeed the optimist could look at data about worldwide internet access and see the doors of higher education open to everyone as millions jump across the digital divide. Still, we ought to take a look at Sousa Santos' (2008) criticism of how the trends in Western universities toward professors losing their central role endangers academic freedom, but also at Nicholas Carr's criticism of unbundling. In *The Big Switch* (2009), Carr gives the example of newspapers in order to explain how individual webpage tracking unravels *"an intricate and, until now, largely invisible system of subsidization."* As *"classified ads, for instance, can no longer help to underwrite the salaries of investigative journalists or overseas correspondents,"* we may suspect that unbundling courses may rot universities from their core. That older blog post by Carr on *The Amorality of Web 2.0* (Carr, 2005) is prescient: humanities departments are either deemed unprofitable and incompatible with the institutions' 'mission statements' (Fish, 2010), or else the Arts and the Letters are presented as a luxury offering, a kind of conspicuous educational consumption.

Though Jenkins (2006) expressed some concerns about the changes wrought by media convergence, Sonia Livingstone (2014) sees Jenkins' position as mostly celebratory of the challenges posed by digital media to education, which she contrasts with the positioning of Buckingham and colleagues,[*] *"for whom digital media extend the reach of educational institutions into the home, 'curricularising' leisure and expanding the profitable marketplace of edutainment and informal learning products and services"* (Livingstone, 2014). Helena Cabeleira (2018) criticizes how these developments shaped the passage of education from 'a right' into 'a duty,' particularly evident in the calls for 'lifelong learning' and in educational policy recommendations that emphasize 'learning to learn.' As the author sees it, such proposals amount to a defense that lifelong learning can be underpinned by *"a void of knowledge"* (Cabeleira, 2018).

Though noting his refusal of a dichotomic perspective, António Nóvoa defends that universities ought to *"place the short tempo of 'modernization' within the long tempo of pedagogical and scientific work"* (Nóvoa, 2018). The author criticizes the specialization that accompanies emphases on employability and calls for a movement toward a liberal education that brings together scientific and humanistic knowledge; moreover, instead of a homogenizing notion of 'excellency' based on quantitative rankings and metrics, the author also calls for universities to revalue pedagogy and *"understand the importance of cross-fertilization between teaching and researching"* (Nóvoa, 2018).

Indeed, Gary King and Maya Sen (2013) suggest that higher education institutions are, as producers of educational research, in a privileged position to address the challenges posed by social and technological changes. Institutions ought to assert their independence, as William Dutton had stated one decade earlier:

[*] Buckingham, D., Scanlon, M., & Sefton-Green, J. (2001). Selling the digital dream. In Ellis, Loveless (Eds.), *ICT, Pedagogy and the Curriculum* (pp. 20–40). Routledge Falmer.

> Colleges and universities should not simply follow the calls for 'student-centered' learning, but play a role in the invention of new institutional and learning paradigms through critical research that informs the production and consumption of ICTs in higher education and learning. (Dutton, 2002)

The value of technological innovations in education appears to oscillate between optimism toward their potential and disappointment with the humdrum reality of adoption (Latchem, 2014). A recent survey of 1,650 undergraduate students found that most saw the benefits of digital technologies in the *"logistics of university study rather than matters related directly to learning per se"* (Henderson, Selwyn, & Aston, 2017). Students widely adopt unimaginative but pragmatic uses of technology (e.g., organizing study materials, working off-campus, saving time, communication, online research); yet the authors note those *"are not the most expansive, expressive, empowering, enlightening or even exciting ways that digital technologies COULD be used"* [their emphasis] (Henderson et al., 2017). Studies of the adoption of digital media in Portuguese higher education have also shown the central role of the teaching practices of individual professors in shaping adoption (Hasan & Laaser, 2010; Morais, Morais, & Paiva, 2014); institutions assert themselves only in facilitating platforms and software tools.

As in all debates about technological and scientific progress, there is a taut line between educational dystopias and utopias. It is only appropriate the editors of the 2016 *Wiley Handbook of Learning Technology* raised the debate. Neil Selwyn presents a dystopian vision in which the teacher is displaced and deprofessionalized, learners are disengaged, knowledge is 'dumbed down' and devalued, and the educational system itself becomes a surveillance apparatus (Selwyn, 2016). On the other hand, Marcus Childress presents a vision of education that is learner-centric and engaging, collaborative, and informal; *"anywhere, anytime, blending virtual and real, combining sources/providers"* (Childress, 2016). To the researcher, the latter author's *"breathless adoration of all things digital"* (Selwyn, 2015) seems to meet the adage from science fiction that every utopia hides a dystopia.

### 2.2.1 A CRITICAL ENGAGEMENT WITH TECHNOLOGY

Geert Lovink argues for a return to cybernetic studies aimed at, besides concerns with social media, *"inserting self-reflexivity in code"* (Lovink, 2012). He calls for *"research programs run by technologically informed theorists"* and denounces the *"submissive attitude in the arts and the humanities toward the hard sciences and industries."* Kember and Zylinska shared Lovink's call for self-reflexivity but across all creative processes, and quote Gilles Deleuze in that *"there is no creation without experiment"* (S. Kember & Zylinska, 2012, p. 187). The authors call for a new critical paradigm that recognizes what they portray as the false problems and divisions, "*between production and consumption, text and image, and language and materiality,"* that characterize conventional media analysis, and that embraces instead critical creation and invention (S. Kember & Zylinska, 2012, p. 203); a point which, as we shall see in the next chapter, is particularly salient to how artistic disciplines engage with technologies:

> Critics of the world, create! (S. Kember & Zylinska, 2012, p. 205)

Over the years, several authors have argued for sustained critical analysis as a vital component of using technology in education (cf. Sassen, 2002; Selwyn, 2015); more so in times when questioning the positivity of technological interventions may go against popular opinion or academic self-interest, as Selwyn (2015) notes. Said critical analyses may look inward at the ongoing adoption practices or outward at the future innovations. More crucially, the critical studies may also look forward, as suggested by some of the aforementioned authors, toward the possibilities for higher education to shape the exploration of the potential of technology:

> Instead of struggling with the over-hyped, pre-configured digital products and practices that are being imported continually into university settings, a genuine grassroots interest needs to be developed in the co-creation of alternative educational technologies. In short, mass participation is needed in the development of 'digital technology *for* university educators *by* university educators.' Only then might the 'disruptive' and democratizing possibilities of digital technology in higher education that are so often promised stand a chance of actually being realized in practice. (Selwyn, 2013)

## 2.3 *Computing literacies*

Writing the third volume of his *Critique of Everyday Life* in 1981, Henri Lefebvre saw *"the substitution of the 'user,' figure of daily life, for the political figure of the 'citizen,'"* after *"a massive operation of recuperation, a skillful defusing, an appropriation carried out by state-political power over a fairly long period"* (Lefebvre, 1981/2014, pt. I, ch.8). Though Lefebvre's notion of the 'user' at the time, figuring *"in social practice as one party, invariably absent and represented, in contractual terms and conditions,"* closely aligned with what became of the 'consumer' in the popular lexicon, we also find the notion prescient with regard to the process of convergence and 'defusing' of information technology 'users' in recent decades.

> Should the notion of users and the practice that corresponds to it be destroyed? No! But use must be connected up with citizenship, as opposed to separating the two. (Lefebvre, 1981/2014, pt. I, ch.8)

In *Technopoly* (1992), Neil Postman shares a similar assessment to Lefebvre about the sovereignty of technology over life and culture. Postman also shares a vision of "*romantic resistance*" partly underpinned by a renewed emphasis on the humanities in education, in particular the disciplines of history, semiotics, and epistemology (pp. 166–171). This does not imply that education should embrace Luddism; on the contrary. If employed appropriately, computing may help *"engineering a new epistemology"* (diSessa, 1985), following the proposal of Papert (cf. 1980). Indeed, the whole project of 'emancipatory computing,' as most famously articulated by Ted Nelson in *Computer Lib/Dream Machines* (1974), has been precisely one of *"romantic resistance."*

Hence, a fundamental issue concerns the new types of literacy that involve the use of computers and digital media. Whether digital literacy concerns the training of skillful users-consumers or instead concerns education in epistemological thinking closely matches the debates about the contemporary positioning of higher education.

According to Meyers and colleagues (2013), the notion of digital literacy has evolved since Gilster (1997) first framed the concept as referring to a set of internet use skills salient in formal education contexts. More recent understandings of digital literacy are widely variable but generally encompass skills ranging from internet research and information evaluation to digital content creation and sharing, articulated with technical proficiency (Meyers, Erickson, & Small, 2013). This changing notion of digital literacy cannot be separated from the dramatic increase in connectivity since the turn of the millennium and the increasing role of informal contexts—in the home or through mobile devices—for learning and using digital media (Meyers et al., 2013).

More recently, the DigComp 2.0 European Commission's Digital Competencies Framework (Vuorikari, Punie, Carretero, & Van Den Brande, 2016) described five main components: information and data literacy, communication and collaboration, content creation, safety, and problem-solving. Therein, technical proficiency is just one of the skills of the digital literate. The authors of the framework place computer programming as one of the competencies for digital content creation, yet it is argued that developing the associated computational thinking skills benefits all main areas, problem-solving most of all (Fraillon, Ainley, Schulz, Friedman, & Duckworth, 2020).

Annette Vee (2013) argued that writing and computer programming have followed similar historical trajectories and attempted to wrangle competing definitions and positions toward computational literacy. The lexicon is complicated and in flux. Donald Leu and colleagues separate 'uppercase Literacies' from 'lowercase literacies' (Leu, Kinzer, Coiro, Castek, & Henry, 2017), whereas Vee distinguishes between 'literacies' proper and—after Andrea diSessa—'material intelligences,' noting that though people benefit from the former, *"they need literacies to negotiate their world"* (Vee, 2013, p. 45).

To the question of whether computer programming meets the urgency of a proper Literacy, Vee believes that to be the case, stating, *"as code becomes more infrastructural and as more people learn to write it, computer programming is looking more and more like a literacy"* (Vee, 2013, p. 46). The author distinguishes several terms used in literature to refer to literacies related to computer programming, such as *computational thinking*, *procedural literacy*, and *computational literacy*.

Computational thinking, as proposed by Jeanette Wing (2006) and summarized by Paulo Blikstein, refers to the notion that programmers' *"ways of thinking, heuristics, and problem-solving strategies are universally important for both applying computing ideas to do work in other disciplines, and for applying computing ideas in everyday life"* (Blikstein, 2018, p. 8). Vee (2013) criticizes how this idea of computational thinking places the discipline of computer science as its focal point, stating, *"as programming moves beyond CS, we must broaden the conceptual apparatus we use to understand its functions in the world"* (Vee, 2013, p. 46).

Procedural literacy *"entails the ability to reconfigure basic concepts and rules to understand and solve problems, not just on the computer, but in general,"* according to Ian Bogost (2005). Recognizing the role of play, Michael Mateas (2005) emphasizes *"the craft skill of programming"* as a fundamental component of procedural literacy, whereas for Bogost (2005), the acquisition of procedural literacy comes with *"any activity that encourages active experimentation with basic building blocks in new combinations."* Bogost favors learning through recombination of higher-level elements with inherent

meanings—criticizing, as an example, Lego building blocks for being too elementary in comparison to Playmobil's (2005). Hence, Bogost argues that procedural literacy may be underpinned by exploratory authorship but also by play and consumption.* In contrast, Vee (2013) defends the centrality of writing to any notion of literacy and thus disagrees with Bogost's lesser emphasis on computer programming (p. 46).

Annette Vee (2013) admits to a closer alignment with Andrea diSessa's notion of computational literacy (e.g., diSessa, 2018), and offers her definition as encompassing *"the constellation of abilities to break a complex process down into small procedures and then express—or "write"—those procedures using the technology of code that may be "read" by a non-human entity such as a computer"* (Vee, 2013, p. 47).

⁂

As Curzon and colleagues (2019) note, computation dates back to Euclid (circa 300 BCE), if not earlier. Hence, Warren Sack (2019) positions computing as an art in the sense of 'arts and crafts,' arguing against computing as purely a science or software development as purely a form of engineering. Still, Sack does problematize computational thinking as a concept that is in turn either promoted as *"the queen of sciences"* in place of philosophy or mathematics (e.g., Stephen Wolfram's *A New Kind of Science*, published in 2002) or just as a vague buzzword *"supported by largely corporate concerns"* (Sack, 2019). The position of education on computing remains as an open question.

Testifying to cyclical nature of these debates, Seymour Papert had recognized the hazards of a digital divide and of social fragmentation borne out of computer networks (Papert, 1980, pp. 25–26). In doing so, Papert preempted some of the contemporary criticism of computational thinking articulated by authors such as Nicholas Carr (cf. Carr, 2005):

> By deliberately learning to imitate mechanical thinking, the learner becomes able to articulate what mechanical thinking is and what it is not. The exercise can lead to greater confidence about the ability to choose a cognitive style that suits the problem. Analysis of 'mechanical thinking' and how it is different from other kinds and practice with problem analysis can result in a new degree of intellectual sophistication. (Papert, 1980, p. 27)

Papert defends an early contact with computers, which he conceives as *"the best possible antidote"* against learners being induced a sole mode of thinking (p. 27). Likewise, Luke (2000) appealed for educators to lead the development of new media pedagogies, or else *"corporate experts will be the ones to determine how people will learn, what they learn, and what constitutes literacy"* (p. 69). Nonetheless, this has often not been the case. Indeed, though children may now learn to operate smartphones even before they learn to walk, actual educational contact with computing is often delayed, sometimes until higher education.

* Indeed, over the last decade, digital games and platforms that bridge construction and consumption such as Minecraft or Roblox became popular educational tools (e.g., Nebel, Schneider, & Rey, 2016; Overby & Jones, 2015).

### 2.3.1 STUDIES OF COMPUTING AND INFORMATION LITERACIES

In 2013, the International Association for the Evaluation of Educational Achievement (IEA) carried out a first International Computer and Information Literacy Study (ICILS) of 60,000 8th-grade students and 35,000 teachers in 21 countries. The authors of the study employed a skills-based definition of computer and information literacy (CIL): *"an individual's ability to use computers to investigate, create, and communicate in order to participate effectively at home, at school, in the workplace, and in society"* (Fraillon, Ainley, Schulz, Friedman, & Gebhardt, 2014, p. 15). Given the nature of the assessment, the authors of the study found participants' computer and information literacy correlated with their basic self-efficacy in using information technologies (IT), though not with advanced skills such as programming (p. 253). Other ICILS 2013 findings are worth emphasizing. Against a backdrop of highly variable CIL by students and teachers, the authors still found higher socioeconomic status was associated with students' higher proficiency in IT use (Fraillon et al., 2014, p. 20), as was greater students' regularity using a computer in the home (p. 21). Moreover, the study also challenged a 'pervasive belief' in male students' greater proficiency in the use of information technologies (Fraillon et al., 2014; Gebhardt, Thomson, Ainley, & Hillman, 2019), as they found *"female students performed relatively better on tasks that involved communication, design, and creativity"* (Gebhardt et al., 2019, p. 69). However, the original report does note that male students showed higher self-efficacy in advanced IT competencies, which include proficiency in computer programming (Fraillon et al., 2014, p. 252).

Though some participating teachers in the ICILS 2013 believed IT *"use could have some detrimental aspects, such as adversely affecting students' development of writing, calculation, and estimation skills,"* most recognized the positive aspects of information technologies (Fraillon et al., 2014, pp. 253–254). The study also found teachers most frequently used technologies for simple tasks (e.g., presentations, word processing, web browsing) and asked their students for equally simple uses (p. 254). The authors of the study thus called for more formal computing education in the schools:

> Results suggest that the knowledge, skills, and understandings described in the CIL [computer and information literacy] scale can and should be taught. To some extent, this conclusion challenges perspectives of young people as digital natives with a self-developed capacity to use digital technology. (Fraillon et al., 2014, p. 24)

A 2016 review of existing studies evinced similarities in how Portuguese higher education students used computers and the internet. If we believe their use of technology in the 8th grade was similar to those of participants in the ICILS study, then not much changed during their additional years at school. Students' computer proficiency, creation of online content, and adoption of the internet for learning were highly heterogenous and associated with gender, educational subsystem (i.e., public or private), and field of study (Santos & Azevedo, 2016).

The second ICILS study, carried out in 2018, surveyed over 46,500 students attending the 8th grade and over 26,500 teachers in 12 countries, which now included Portugal (Fraillon et al., 2020). In addition to the previous study's concern with IT

proficiency and literacy, the ICILS 2018 also surveyed the development of computational thinking skills in basic schooling. According to the authors, though most countries included aspects of computer and information literacy in their 8th-grade curricula, content related to computational thinking (e.g., computer programming) was less commonly taught (p. xx). Still, the study found a larger variation in eighth-graders computational thinking skills within each country than across participating counties, as well as a strong positive relationship (.82) between students' computational thinking and CIL scores (p. 110). The study also found teachers' valuing of these skills was highly heterogenous and dependent on their own disciplines; nonetheless, teachers' valuation of both CIL and computational thinking had a positive relationship with their own IT self-efficacy (Fraillon et al., 2020, p. 247).

In comparison to their earlier 2013 study, the authors of the ICILS 2018 found no differences in the way computers and other digital technologies were used in 8th grade classrooms. Female students had significantly higher computer and information literacy achievements than their male peers, who, in contrast, showed higher computational thinking skills, though at a lesser level of significance (Fraillon et al., 2020, p. xx). In both measures, however, the authors found evidence of a digital divide: higher socioeconomic status was associated with higher CIL and computational thinking skills (p. 244).

In both the 2013 and 2018 studies, the authors criticized how technologies were often used for little more than having 'digital textbooks' (Fraillon et al., 2020, p. 264, 2014, p. 257). In their conclusions to the 2018 study, they thus noted, *"for policymakers this raises a number of questions about the contrast between the messages that are frequently provided around the use of ICT in teaching and learning and the reality of classroom practice"* (Fraillon et al., 2020, p. 247).

> If the history of literacy is any model [...] then the development of more accessible programming languages might increase rather than decrease pressures on computational literacy. More sophisticated and more widely distributed writing technologies actually seem to have put more pressure on individual literacy, ratcheting up the level of skill needed for one to be considered 'literate.' (Vee, 2013, p. 49)

Teaching computer programming does not necessarily map to the training of computer programmers. Guzdial (2019) notes two main categories of motivations for teaching the subject to a broader variety of students: using programming as a tool to think with, and as a tool for doing things. Blikstein (2018) proposes four main reasons for pre-college programming education: first, to meet the demands of the job market; second, to develop computational thinking skills—after Wing (2006); third, to develop a computational literacy— after diSessa (2018) and Vee (2013); and fourth, as to ensure equal and diverse participation.

We may associate an education for meeting the demands of the job market with Guzdial's (2019) notion of programming as a tool for doing. In clerical work, skills such as proficiency in Microsoft Excel formulas (even if not typically regarded as 'programming') are often set by employers as a mandatory requisite. Moreover, in fields such as communication and web design, recruiters often expect practitioners to have skills in web scripting—not just so that designers can code, but so that they can effectively communicate within a multidisciplinary team.

Blikstein's (2018) defense of universal education in computer programming as to foster equal participation is undoubtedly tied with those demands of employers and industries. There, there is still a long way toward the equal participation of women in technological development. In fact, recent decades have shown a growing inequality: Sack (2019) notes a drop in female Computer Science graduates from 37% in 1984 to 14% in 2013. Though education may celebrate the achievements of Ada Lovelace, Grace Hopper, or Katherine Johnson (the NASA mathematician portrayed in the film *Hidden Figures*), equity also requires an equal reclamation of the mundane, from entry-level web development and middling project management jobs to the upper management offices of unremarkable enterprises.

Besides the economic rationales for learning to program, authors such as Nick Montfort (2016) also emphasize the cultural and social benefits. As Douglas Rushkoff (2010) notes, *"for the person who understands code, the whole world reveals itself as a series of decisions made by planners and designers for how the rest of us should live"* (p. 134). Programming not only allows understanding but creation and communication *"across the cultural divide between the artists and programmers"* (Mateas, 2005).

Programming can also be a satisfying and pleasurable activity, *"the pleasure of making and of discovery through making,"* as Montfort (2016) notes. Marina Bers (2017) criticizes a perspective of computational thinking limited *"to a problem-solving process that complements mathematical and engineering thinking, obscuring the relevance of expression and communication in the act of programming"* (p. 74), and advocates for an emphasis on the hedonic aspects of learning to code.

⁂

Teaching programming to first-time learners is particularly challenging for educators. Unlike in other disciplines, students are often absolute beginners that lack basic logical thinking skills (Gomes, Mendes, & Marcelino, 2014). In the following subsection, we shall summarize the past and the present of research about computing education. In §3.2, we shall review the common learning objectives expected of first-time learners of computer programming, alongside the most commonly employed pedagogical strategies.

## 3.1 *Computing education research*

Though we are living in the eighth decade since the first electronic computers, we find computing education research in higher education is still a recently crystalized field, perhaps owing to computing's lack of historicity. As recent as 2019, in their introduction to the newly released *Cambridge Handbook of Computing Education Research,* editors Sally Fincher and Anthony Robins admit:

> We still don't know enough about how students learn computing subjects, what effects different teaching approaches have, or how to equally engage people of all races and genders in the field. (Fincher & Robins, 2019)

The evolution and saliency of computing education research appear to be interrelated with the affordances of technological innovations. Mark Guzdial and Benedict du Boulay (2019) situate the origins of computing education research with the first observations of programming teaching during the mid- to late-1960s, whereas Paulo Blikstein and Sepi Moghadam (2019) point toward Papert, Feurzeig, and Solomon's initial proposal of the LOGO language in 1967. Guzdial and du Boulay (2019) identify two main research streams at the time. The first was about the psychology of computer programming, with research on novices' difficulties (most often applying observational methods) and their implications for the design of programming languages. The second research stream concerned the learning of computer programming in educational settings. The distinct reasons for learning imply two angles to this context of research: either about learning *through* code—of which Papert's work with LOGO is paradigmatic—or learning *to* code—e.g., studies of students' difficulties with languages' syntax or of the difficulties arising from preexisting beliefs in computers' 'intelligence' (cf. Pea, 1986).

According to Guzdial and du Boulay (2019), the 1970s saw a new research concern about computer programming teaching pedagogies, though most inquiries at the beginning looked into the choice of appropriate programming languages for first-time learners. This changed during the 1980s, as educational researchers and cognitive scientists arrived at the field, previously shaped by the work of technologists. Despite the paradigmatic shift into personal computing, much of the thinking about computing education at the time was dominated by LOGO and Smalltalk. Both Blikstein and Moghadam (2019) and Guzdial and du Boulay (2019) note a peak in the popularity of computing education research during the 1980s, which cannot be disentangled from the formation of the first international research groups and the influence of the first massive experiments in introducing microcomputers in the classroom—e.g., the aforementioned

BBC Computer Literacy Project in the United Kingdom (see §1.1). The ensuing decline in computing education research during the 1990s is possibly related to the consolidation of the GUI model, the emphasis on teaching IT competencies underpinned by particular application software tools, and the new research focus on the internet.

Blikstein and Moghadam (2009) note the resurgence of the computing education field in the mid-2000s. Factors for this resurgence may include the renewed educational focus on science, technology, engineering, and mathematics (STEM), the release of new programming languages intended for novices—e.g., Scratch, Alice, or Processing—as well as the founding of organizations dedicated to promoting computer programming and mitigating digital divides—e.g., Code.org and Girls Who Code in the United States, Kodea in Chile, Coding March in China, Code Tigers in India, Code for Change in South Africa, Adfaber in Romania, Edvira in Sweden, and Computing at School in the United Kingdom. Given the convergence of factors and the pace of innovations—organizational as technological—the small computing education research community has not been able to keep pace (Blikstein & Moghadam, 2019; Guzdial & du Boulay, 2019).

### 3.1.1 MAIN QUESTIONS IN THE FIELD

Guzdial and du Boulay (2019) agree that, over its existence, the field of computing education research has been shaped by three main groups of questions:

FIRST, questions related to learners' perceptions of how computers work and their mental models of computer programs, which refers to novices' need to overcome their belief in computers' intelligence, which Roy Pea described as the 'superbug' (1986). Graphical user interfaces (GUIs), further mediating the interactions with computers, may have reinforced those beliefs, as do the widespread contemporary dissemination of natural language processing—e.g., from search engines to the 'AI' assistants on our smartphones. This mismatch between the surface appearance, polished over seventy years of continual design and technological development, and the reality of computers' operation remains an ongoing concern to be addressed. Acquisition of procedural literacy, in the sense so ascribed by Mateas (2005), is herein especially relevant.

SECOND, the questions related to programming as a notation for formalizing thinking, which is tied to the research on the specific affordances of programming languages (cf. diSessa, 1985; Papert, 1985). Marian Petre (1995) suggested that text-based languages were superior to visual languages in most domains, though that may relate to most learners' greater comfort and expressivity with textual notations. Still, Guzdial and du Boulay (2019) suggest novices may find themselves more comfortable with visual languages. In this context, block-based languages such as MIT's Scratch, in which sections of code can be assembled like construction pieces, are a recent development that merges the textual and the visual in a way that eases the transferability of students' knowledge toward the more powerful textual languages (Weintrop & Wilensky, 2015). We will present a panorama of languages suitable for introductory programming in the fields of arts and design in the next chapter.

THIRD, questions related to whether animated representations of the algorithms being executed are beneficial. Guzdial and du Boulay (2019) note there is little evidence that animations lead to a better understanding of the algorithms. Still, we note the

influential suggestion of Learnable Programming (Victor, 2012), which calls for nothing less than a redesign of programming environments as to bring that visibility to the fore, and not just for the benefit of novices.

## 3.2 *Teaching objectives and pedagogical strategies*

> The ubiquity and invisibility of programs might make developing understanding of how programs work even harder. It is hard to learn about something one cannot see. (Guzdial & du Boulay, 2019)

Reviewing multiple studies from the early 1970s to the present day, Anthony Robins (2019) tells us introductory computer programming units have historically shown high failure and dropout rates. Anabela Gomes and colleagues (2014) note educational actors frequently fail to acknowledge how learning to program is a gradual process that takes time and instead pressure students to demonstrate planning and judgment abilities before they have been able to prove their knowledge and comprehension.

### 3.2.1 LEARNING OBJECTIVES FOR NOVICES

After a review of the literature about first-time programming learners, Robins, Rountree, and Rountree (2003) produced a model of common learning requirements associated with each stage of a program's development, which we present in the following table:

***Table 2.3.1.*** *Demands from computer programming learners, after Robins et al. (2003).*

| *Stages* | *Knowledge of…* | *Strategies for…* | *Models of…* |
|---|---|---|---|
| *Design* | Planning methods<br>Algorithm design<br>Formal methods | Planning<br>Problem-solving<br>Algorithm design | Notional machine<br>Problem domain |
| *Generation (writing)* | Languages<br>Libraries<br>Tools | Algorithm implementation<br>Coding<br>Accessing knowledge | Desired programs |
| *Evaluation* | Debugging tools<br>Debugging methods | Testing<br>Debugging, tracking/tracing<br>Repairing | Actual program |

A particular early point of the difficulty for learners concerns their conceptualization of how computers work. First-time learners often ascribe intelligence to computers, a misunderstanding Roy Pea (1986) characterized as the 'superbug.' To counter students' misconceptions, du Boulay (1986) proposed teaching a simplified model of a 'notional machine,' associated with the particular affordances of the programming language being learned.[*] Subsequent research found that introducing a notional machine model led students to better problem-solving outcomes (Robins, 2019).

In accordance, though first-time learners also face problems with the particulars of programming languages (e.g., their syntax), the literature also agrees that they had

[*] E.g., given how JavaScript manages memory allocation automatically, students can be told to conceive of a computers' memory as a space available for placing labelled 'boxes' (i.e., variables); such a model would be dangerously inadequate if teaching but the most basic C.

the most difficulties in planning and designing algorithms and in tracking and repairing issues with their programs—i.e., 'bugs'—competencies that require a clear concept of the machine's workings (Robins, 2019).

Paul Curzon and colleagues (2019) summarize widely-agreed computational thinking skills as algorithmic thinking, logical thinking, abstraction, generalization, and decomposition. As presented in Table 2.3.1, these skills mainly relate to the strategies required at the design and evaluation stages, and not so much with those needed for the middle stage of actually writing code in front of a computer (cf. Robins et al., 2003).

### 3.2.2 PEDAGOGICAL STRATEGIES

Falkner and Sheard (2019) reviewed the common pedagogical strategies adopted by computing educators. Chiefly amongst those, *active learning* pedagogies that demand deeper individual engagement are found to be effective across a number of disciplines (cf. D. Kember & McNaught, 2007). As Bryan Alexander jokes in his *Devil's Dictionary of Educational Technology* (2016):

> *Active learning, n.* [...] The strange idea that learning and learners should not be as passive as the dead. Like the dead, active learning is a source of wonder and dread to some of the living. (Alexander, 2016)

Common strategies to engage programming students in active learning include the use of live coding demonstrations (Rubin, 2013) of prepared or 'worked' examples (cf. Paas, Renkl, & Sweller, 2003), in addition to in-class exercises such as code completion (van Merriënboer, 1990) or 'Parson's problems,' which require learners to place pre-written blocks of code in the correct order for successful execution (Parsons & Haden, 2006).

*Collaborative learning* pedagogies such as studio-based learning, though variable in scope, require students to work together across the multiple stages of the design and implementation of the solution to a problem or in a larger project. These have often proven effective (cf. Carter & Hundhausen, 2011). Studio-based pedagogies also present a bridge between computing and arts education (Knochel & Patton, 2015; Peppler, 2010). Collaborative practices relate to *contributing student* pedagogies (Hamer, 2006), such as asking individual students or groups to produce artifacts for sharing with other students; these also meet common practices within the art studio.

Computing educators also leverage the possibilities of the internet to implement *blended learning* practices (cf. Graham, 2006; Lakhal & Meyer, 2019) such as a 'flipped classroom,' which Falkner and Sheard (2019) note are more complex than a mere inversion of classroom and homework activities, as most often it requires intensive technological support (e.g., video lectures). Though it may free class time for engaging activities, research has shown that blended learning, by itself, does not have an impact on students' engagement (Dziuban & Moskal, 2011) and should not be regarded as a means to reduce face-to-face contact (Hutchings & Quinney, 2015; Livingstone, 2014). As Alexander (2016) jokes:

> *Blended learning, n.* The practice of combining digital and analog teaching. Also referred to as 'teaching,' 'learning,' and 'the real world.' (Alexander, 2016)

※

In the next chapter, we shall focus on how computing is integrated into contemporary arts and design practice. We shall also address how programming is integrated into the higher education arts and design curricula, how computing pedagogies bridge the interdisciplinary gap, and which languages and tools offer salient affordances.

# 3

## *The arts, programming, and higher education*

★ SYNOPSIS We review present discussions and practices about the integration of computing within higher arts and design education. In section §1, we present a historical outline of the exploration of computer technologies within the arts. In section §2, we look at the specificities of arts education and the evolution of its goals and pedagogies in the face of digital media, with particular attention to the inclusion of computer programming in the curricula. Finally, in section §3, we turn our focus to the recent developments in programming languages specifically aimed at creative code education.

### 1 COMPUTERS AND THE ARTS

> Software influences all aspects of contemporary design and visual culture. Many established artists, such as Gilbert and George, Jeff Koons, and Takashi Murakami, have integrated software into their processes. Numerous prominent architects and designers use software extensively and commission custom programs to realize their ideas. [...] While these exciting developments are taking place at the highest levels of the creative professions, integrating them into design education is a challenge. Even the most motivated student will find the technical boundaries difficult to overcome. (Reas, McWilliams, & Barendse, 2010, p. 9)

Kylie Peppler regards the digital arts* as unstable and emergent, a field that *"can be situated as being distinct from traditional disciplines (such as visual art, music, dance, and theatre) but includes some overlap that arts educators may recognize, such as visual arts, animation, film, and, at times, electronic music"* (Peppler, 2010b). Moreover, as Brown and Nelson (2013) suggested, "*the interoperability of digital data between modes of output [...] has provided additional impetus to interdisciplinarity across the arts and with computer science."* Peppler (2010b) describes digital art *"as a 'meta-medium' that spans many different types of artistic practice,"* employing the term chosen by Alan Kay and Adele Goldberg (1977/2003) to refer to the realization of computers' potential.

⁂

* A term the author sees as synonymous to 'new media' or 'media arts,' her preferred term.

In the following subsection, we shall summarize the historical foundations of digital art, its relationship with the world of art and other artistic movements, and how the early research agendas still set contemporary concerns. In subsection §1.2, we shall address how the view of computers as a tool led to an infatuation with specific software products within arts and design disciplines. A counterpoint will be made in subsection §1.3, where we shall see how programming remains central to the exploration of computing as an expressive medium.

## 1.1 *Foundations of digital art*

The integration of computer technology in artistic and applied design processes began almost as soon as electronic computers became a reality, though, as we shall see, wide acceptance of computers as artistic tools or as a creative medium in the world of art came much later. Still, today's computational aesthetics research programs are, for the most part, a continuation of agendas set during the first two decades. Thomas Dreher (2014) names three such agendas: *information aesthetics*, from whence came the initial works of computer literature and computer graphics; *moving image*, from whence came the work in video and computer animation, from simulation to film; *cybernetics*, from whence came the pioneering interactive sculptures and the subsequent work on human-computer interaction that continues to this day.

### 1.1.1 EARLY COMPUTER ART

Some of the earliest artistic explorations of digital technology concerned the use of computers as poetry and music compositional aids. In Manchester, Christopher Strachey produced random *Love Letters* (1952) using the Ferranti Mark I, the first commercially available computer. Possibly influenced by the contemporary cut-up poetry experiments of William S. Burroughs or the earlier experiments of Dada artists in the 1920s, Theo Lutz in Stuttgart explored the production of *Stochastic Texts* (1958) using computer programs to rearrange words and sentences.

Electronic musical instruments were built since the turn of the 20th century, and a few electroacoustic studios were already established by the time computer technology became accessible at large research institutions. From the 1950s, composers such as Iannis Xenakis in Paris or Pietro Grossi in Florence employed computers as a compositional aid; in 1957 at the University of Illinois, fellow composers Lejaren Hiller and Leonard Isaacson fully programmed the *ILLIAC Suite* in the eponymous computer (Dietrich, 1986). As early as 1961, Max Mathews at the Bell Labs in New Jersey merged music and text, employing speech synthesis to create a computer performance of the song *Daisy Bell*[*] (Dietrich, 1986; Dreher, 2014).

⁂

[*] The famous *2001: A Space Odyssey* (1968) scene in which the HAL computer reverts to singing this song as the astronaut removes its memory modules was directly influenced by Mathews' demonstration.

Classically trained visual artists were more reluctant than musicians to engage with computers. The first exhibitions of computer graphic art took place in 1965 and presented works made by mathematicians and computer scientists (Dietrich, 1986). At the Howard Wise Gallery in New York, researchers such as A. Michael Noll presented works in which they had employed computers to draw mathematical patterns inspired by 'op art' paintings. Across the Atlantic, in Stuttgart, Georg Nees presented works such as *23 Ecke* (1964), shown in Figure 3.1.1a (next page), in which random generated 23-line shapes had been plotted in a cartesian grid; Nees had written graphics extensions for the ALGOL-60 general-purpose programming language, possibly devising the first libraries intended for artistic exploration. A fellow German mathematician and computer scientist, Frieder Nake, instead chose to develop a language wholly intended at graphics production, COMPART, in which Nake programmed hundreds of drawings (Dietrich, 1986; Lieser, 2009).

Some visual artists became interested in computing around the time of the first gallery exhibitions, thus heralding a stage of collaboration between artists and computer scientists (Dietrich, 1986). Artists such as Manfred Mohr learned computer science, while a fellow researcher of Noll at the Bell Labs in New Jersey, Ken Knowlton, developed programming languages for visual artists. Using the BEFLIX language to produce images that were recorded to film, Knowlton and artist Stan Vanderbeek created animations such as *Poem Field* (1966). Knowlton recognized how the specifics of the programming language distinctly shaped the visual outcomes and thus sought to accommodate artists' wishes. He developed EXPLOR for artist Lillian Schwartz, who from 1970 employed it in the production of animated films such as *Pixilation*, shown in Figure 3.1.2a (Dietrich, 1986; Lieser, 2009).

Frank Dietrich (1986) identified four fundamental areas of inquiry in early visual computer art: '*combinatorics*,' exemplified by Mohr's *P 159 (*1973), an exploration of possible projections of a tri-dimensional cube in a two-dimensional plane; investigations of the full range between *"order, chaos, and randomness,"* such the aforementioned work by Georg Nees; experiments with visualizing mathematical functions, such as A. Michael Noll's op art-inspired images or Frieder Nake's 1967 *Matrix Multiplications* series (see Figure 3.1.1b); and the processing of representational images, such as Ken Knowlton's early experiments with image scanning—e.g., *Studies in Perception I* (1966), which Dietrich suggests was the first 'digital nude'—or Charles Csuri's experiments with deconstructing vector drawings, such as *Sine Wave Man* (1967, see Figure 3.1.2b) or his *Hummingbird* (1968) short animation film. For all the technological progress since, the index of a relatively recent book about computational aesthetics—*Form+Code in Design, Art, and Architecture* (Reas et al., 2010)—reports the same fundamental areas of visual exploration under different names: repetition, transformation, parametrization, visualization, and simulation.

⁂

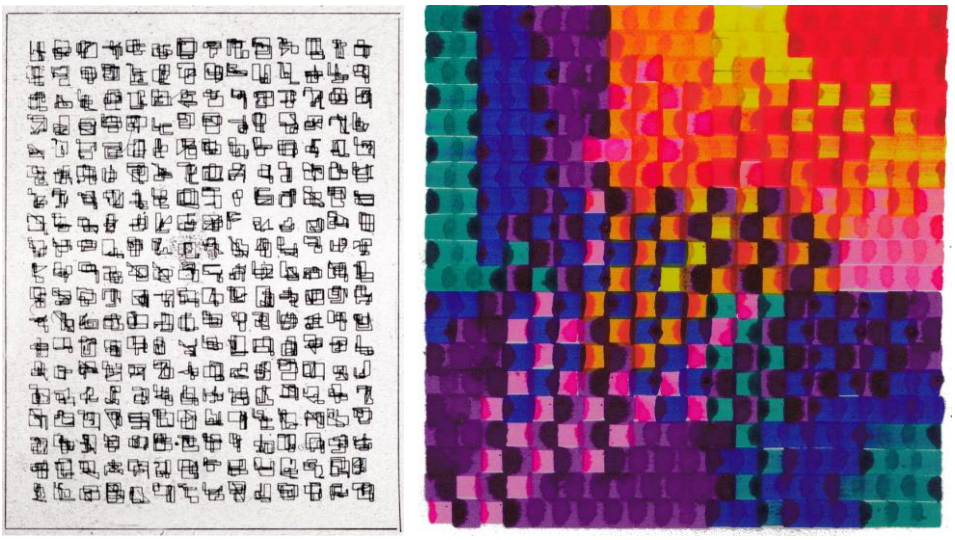


***Figure 3.1.1a*** *(left). George Nees,* 23 Ecke *(1964).*
***Figure 3.1.1b*** *(right). Frieder Nake,* Matrix Multiplication *(1967).*

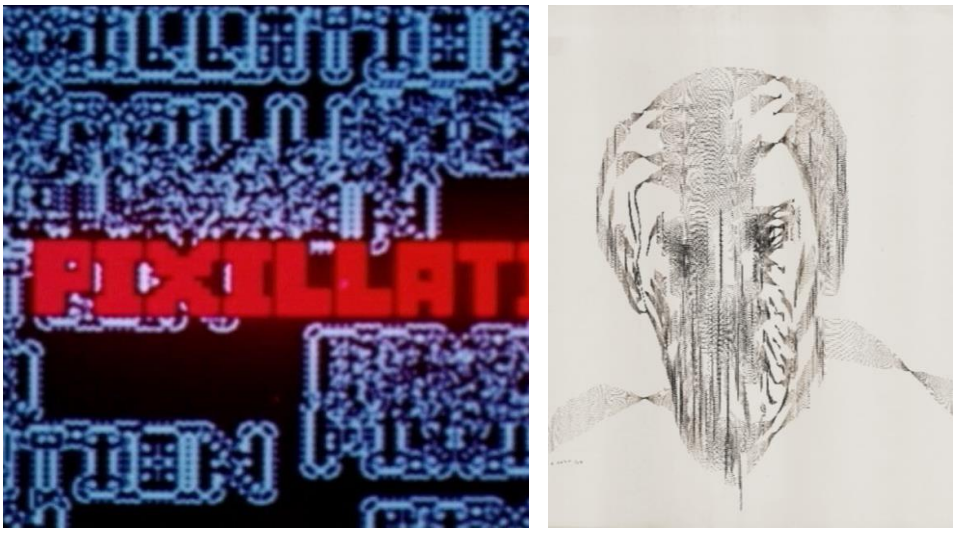


***Figure 3.1.2a*** *(left). Lillian Schwartz & Ken Knowlton,* Pixilation *(1970), still detail.*
***Figure 3.1.2b*** *(right). Charles Csuri,* Sine Wave Man *(1967).*

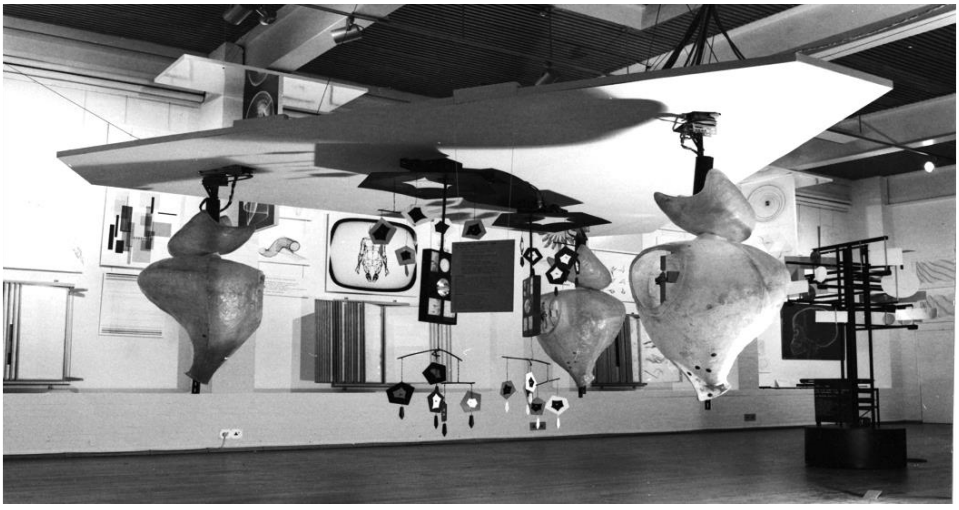


***Figure 3.1.3****. Gordon Pask,* A Colloquy of Mobiles*, at Cybernetic Serendipity (1968).*

According to Dreher (2014), cybernetics, borne out of the World War II research of Norbert Wiener onto control and feedback processes between human operators and machines (cf. Wiener, 1954/2003), configured a distinct research agenda in early computer art. Cybernetics caught the fascination of pioneering artists. Nam June Paik, one of the founders of video art, himself defined cybernetics as "*the science of pure relations, or relationship itself*" (Paik, 1964/2003), whereas fellow artist and educator Roy Ascott saw cybernetics as *"a coordinator of science, as Art is a coordinator of experience,"* and called for artists to turn to *"science as a tool and reference,"* which implied a *"turn to Cybernetics"* (Ascott, 1964). At the time, Ascott had initiated a 'groundcourse' at the Ealing School of Art in London, which, informed also by behavioral sciences, asked of its students the exploration of procedural activities rather than visual observation, as was common studio practice at the time.

The technologies employed by early cybernetic pioneers presented lower barriers to entry than computing; the artistic explorations of the interaction between humans and machines made during the 1960s employed far simpler and more accessible electronics. Gordon Pask's *Colloquy of Mobiles*, shown in Figure 3.1.3 (previous page), made for the 1968 *Cybernetic Serendipity* exhibition at the Institute for Contemporary Art in London, employed electronic sensors (e.g., sound and light-sensitive resistors) and electric motors to animate sculptural figures without the need for a computer.[*]

⁂

In hindsight, the earlier artistic explorations of computers possibly configured early examples of digital remediation (cf. Bolter & Grusin, 2000)—of kinetic sculpture, of visual poetry and Dada literature, of painting, of film. Some artists actively resisted referencing traditional forms: according to Dietrich (1986), Vera Molnár in Paris saw the computer as a means to create *"unimaginable images*" beyond conventional aesthetics.

### 1.1.2 COMPUTERS AND THE ART WORLD

The greater collaboration between computing researchers and artists during the latter half of the 1960s culminated in major exhibitions of computer and cybernetic art—besides *Cybernetic Serendipity* (1968) in London, *Some More Beginnings* (1968) at the Brooklyn Museum in New York, *Computers and Visual Research* (1969) at the Gallery of Modern Art in Zagreb, *Software* (1970) at the Jewish Museum, again in New York (Dietrich, 1986; Lieser, 2009). Though Marshall McLuhan observed, *"the serious artist is the only person able to encounter technology with impunity"* (1964/2008, p. 19), many in the world of art encountered computers with suspicion—if not hostility.

According to Frieder Nake (1971), the first exhibitions raised a serious discussion about whether the outputs of computer code could be art.[†] Some avoided engaging with the question; Nake himself stated he was not interested in using computers *"as a source*

[*] Indeed, cybernetic sculptures often shared the same technical underpinnings as penny arcade and theme park automata.

[†] As Golan Levin (2009) noted, the continued refusal of *Artforum* magazine to engage with digital art decades later demonstrates this polarization persisted, at least in some quarters.

*of pictures for the galleries,"* but rather as tools for *"the investigation of aesthetic information as part of the investigation of communication"* (Nake, 1971). Nonetheless, the question remained. Were computer programs to be seen as musical compositions—instructions to be performed—what philosopher Nelson Goodman termed 'allographic' art; or were programs to be regarded as sketches for unique outputs, thus 'autographic' art (Giovannelli, 2017)? Artists in the conceptual art movement would bring decisive contributions to the discussion. Allan Kaprow's *Fluids* (1967) 'happening' was announced by posters that presented the exact instructions:

> During three days, about twenty rectangular enclosures of ice blocks (measuring about 30 feet long, 10 wide and 8 high) are built throughout the city. Their walls are unbroken. They are left to melt.
> —Allan Kaprow, *Fluids* (1967)

Visual artist Sol LeWitt explored combinatorics in *Fifteen Etchings* (1973), subtitled as *"straight lines in four directions and all their possible combinations,"* or in *Five Cubes on Twenty-Five Squares* (1978), a book of photographs of all the possible placements. His *Wall Drawings* series often made the artworks' algorithms explicit. One of the creators of the Processing programming language, Casey Reas, reinterpreted *Wall Drawings* as computer graphics in his *Software Structures* (2004) exhibition at the Whitney Museum in New York. Take LeWitt's *Wall Drawing #392* (1983), presented in Figure 3.1.4. The artwork's 'algorithm' includes surfaces for exploring parametrization and randomness, yet its implementation in a programming language is trivial.*

***Figure 3.1.4**. Sol LeWitt,* Wall Drawing #392 *(1983): "a 12-inch grid covering the wall. Within each square, a vertical, horizontal, diagonal left or diagonal right line bisecting the square. All squares are filled. The direction of the line in each square is determined by the drafter." [Photo credit: Cathy Carver / Paula Cooper Gallery]*

* Indeed, the researcher often gave it as an early exercise while teaching introductory Processing programming.

Concept, not implementation, is vital. This understanding led to an *"anthropocentric view of computer programming in the arts"* (McLean & Wiggins, 2012): the outputs (e.g., sound, image, spatial or tangible experiences) are subordinate to the writing of code. Frieder Nake (2012), holding that creativity is *"determined by overcoming or breaking the material's resistance"* according to the artists' intent, also acknowledges that in the field of digital art, *"this resistant material is clearly the algorithm."* Nake (2012) maintained the creative value of the computer, noting how "*it may trigger super-surprise."*

⁂

After that initial wave of digital art borne out of digital research laboratories, a new wave was led by classically trained artists who extended their video art practices, such as American video art pioneer Gary Hill who used computer-generated images in works such as *Air Raid* (1974) or *Electronic Linguistic* (1977). As earlier 'electronic brain' hyperboles were recognized as rhetorical and more artists had actual contact with digital media, the perspective that computers were valuable tools gained acceptance in the 1980s, with the arrival of mass-produced personal computers.

## 1.2 *The infatuation with software tools*

The intense competition in the nascent personal computing business led manufacturers to put their products' ease of use and graphics and sound capabilities at the center of promotional efforts. Graphical User Interfaces (GUI) and Direct Manipulation systems (cf. Shneiderman, 1983) met the needs for an easy learning curve of the millions of new users in the home and in the workplace in the 1980s. In addition, the software tools that manufacturers commissioned to demonstrate their computer systems were often aimed at artists and designers. The 1985 launch gala for the Commodore Amiga synthesizes the time: the artist Andy Warhol was invited to demonstrate the computer's video capture capabilities and the ProPaint software to create a colorful portrait of Debbie Harry of the band Blondie, as shown in Figure 3.1.5 below. Computers became pop.

***Figure 3.1.5****. Andy Warhol, Debbie Harry, and the Commodore Amiga computer. [Retrieved from the Computer History Museum—computerhistory.org]*

The late 1980s and early 1990s saw a plethora of offerings aimed at different arts and design disciplines. Musicians used Cakewalk (released in 1987) and Steinberg Cubase (1989) to compose and control digital instruments connected through a MIDI interface; they also used Digidesign Pro Tools (1989) to record and edit multiple audio tracks. Graphic designers and illustrators used Adobe Illustrator (released in 1987), Macromedia Freehand (1988), or Corel Draw (1989) to create vector illustrations and compositions, prepared publications using Aldus PageMaker (1985) or QuarkXPress (1987); typographic designers created new fonts using Altsys Fontographer (1986). Visual artists edited and manipulated images in software such as Electronic Arts' Deluxe Paint (1985) or Adobe Photoshop (1988). Videographers used Lightworks (1988), Avid Media Composer (1989), or Adobe Premiere (1991). Animators and visual effects artists used Autodesk Animator (1989) or the Company of Science and Art's After Effects (1993); designers and 3D graphics artists used Softimage|3D (1988), Autodesk 3D Studio (1990), or Maxon Cinema 4D (1990) for modeling and animation. At that time, architects and product designers were well-versed in using computer-aided design (CAD) tools such as Autodesk AutoCAD (first released in 1982), ArchiCAD (1985), or Microstation (1985).

The list is long and would go on. Though specialized computing equipment held on to specific industries for a few years—e.g., the Quantel Paintbox (1981) for image editing or the Discreet Flint/Flame video workstations (1993)—the increasing power and affordability of personal computers led such products to industry niches. Moreover, as schools began to acquire multimedia computing equipment, the interactions of training and practice yielded industry standards. The 1980s were characterized by a zoo of computer systems and user interface conventions, yet by the mid-1990s, the market had consolidated around Windows PC and Apple Macintosh computers; after the acquisition of the original makers, most creative software had been consolidated under the direction of few companies (e.g., Adobe and Autodesk) which further tightened the conventions and affordances of their software.

Artists and designers understood they worked under the limitations of their software tools, yet some sought to expand the possibilities of computers through a more diverse collection of tools, such as software plugins and extensions—e.g., Kai Power Tools for Adobe Photoshop or Boris, Trapcode, or Magic Bullet for After Effects; curating a good selection of software instruments and effects plugins became almost integral to the use of digital audio workstation software. Yet, some understood a wider collection of tools was often antithetical to creative expression. The enthusiasm for tools had led many to disregard the medium. At the turn of the century, designer and technologist John Maeda reacted against what he perceived as a form of conformity:

> [In] the field of digital art, an entire generation of creators shop at the equivalent of home improvement megastores, eagerly acquiring all kinds of prefabricated components and add-ons. Blissfully unaware of—or even worse, uninterested in—the basic nature of the technologies they are using as tools, the creative élite oversee the assembly of substandard digital objects and experiences. (Maeda, 2000, p. 401)

The early suspicions about computers in creative processes had not disappeared. As Brown and Nelson (2013) suggest, that suspicion had passed on to artists themselves as they embraced computing and came to understand the weight of software: should the

authorship of a digital artwork be shared with Kai Krause (creator of Kai Power Tools)? Still, by the time of Maeda's criticism, a movement was already in place towards granting artists and designers greater freedom of exploration. Multimedia content production—first for CD-ROM distribution and later for the web—grew dramatically throughout the 1990s and brought in a closer working relationship between artists, designers, and technologists.

Specialized GUI tools began to include scripting functionalities. Macromedia Director, first released as an animation tool in 1986, included a scripting language to aid multimedia production—Lingo—from 1988. Macromedia Flash, intended for producing web animations, had scripting functions from its initial release in 1996. Easy to use web design software such as Vermeer Frontpage (released in 1995 and immediately bought by Microsoft) or Macromedia Dreamweaver (1997) exposed the underlying HTML markup and JavaScript code to developers and 'power users' alike.

Specialized scripting functionalities built within tools offered artists and designers a more extensive canvas while allowing software manufacturers to command a continued loyalty from their customers, given their significant investment in training. Adobe InDesign (1999), which replaced the earlier PageMaker desktop publishing software, offered scripting functionalities from the beginning, and the Adobe After Effects animation and video compositing software included such functionalities from 2003. Adobe Photoshop 4.0 (1996) included 'actions,' which brought users a greater quality of life.* Within the field of architecture, Robert McNeel & Associates' Rhinoceros 3D CAD and modeling tool (first released in 1993) carved an important niche after the 2008 addition of a visual scripting language—Grasshopper—that enabled practitioners to generate complex parametric forms more easily (see Figure 3.1.6 below).

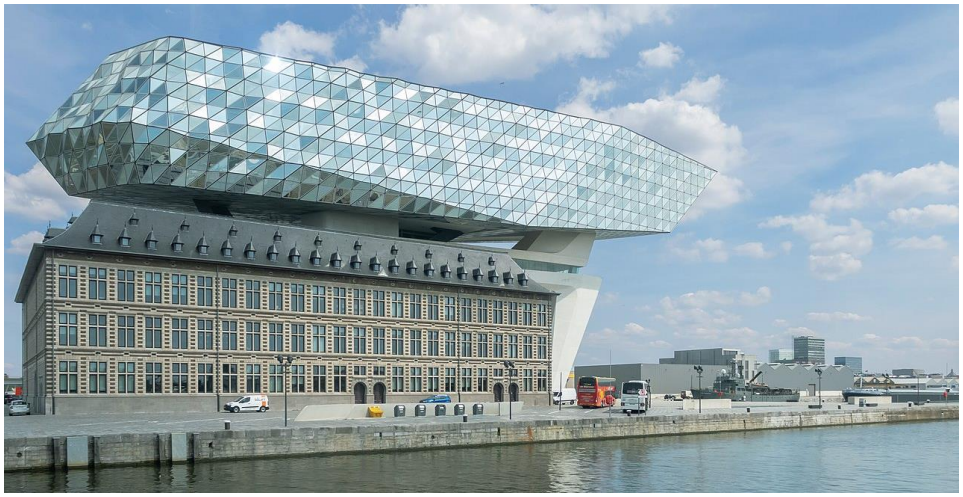


***Figure 3.1.6**. The extension of the Antwerp Port Authority Building by architect Zaha Hadid (2016) is an example of parametric design. [Photo credit: Michel Verbeek / Wikimedia]*

⁂

* While working at an unnamed company in the early 2000s, the researcher was asked to resize several hundred photos for inclusion in a multimedia product and given several days exclusively for the task. Though it took the workstation the better part of an afternoon to execute it, the researcher was done in just a few minutes after devising the right 'action.'

Programming languages, for all their greater expressivity, are still tools. As mentioned in the previous section, during the early years of computer art, Ken Knowlton developed distinct languages to meet the aesthetic interests of different artists. As described in the previous chapter, one of the extant questions in the field of computer education research concerns the distinct affordances of programming languages (see **2**.3.1.1). Grasshopper's emphasis on parametric design allows architects to shape the built environment in distinctive ways to the LISP programming language integrated into AutoCAD. The exploration of computers as an expressive medium, rather than as a tool, demands from artists a deeper engagement with programming and a critical perspective of the superficial affordances of languages and other tools (Meeken, 2012). As we will see in the next section, this became one of the organizing principles of engagement with computer programming within arts and design education.

## 1.3 *Programming and exploration of the medium*

In *Universities and Innovation Economies,* with the suggestive subtitle of *The Creative Wasteland of Post-Industrial Society*, Peter Murphy (2015) has a devastating assessment of the arts in the age of media convergence:

> In the visual arts, the period between 1890 and 1970, i.e., between Cezanne and Rothko, was outstanding. What followed was dismal. Too often the visual art of the post-modern era was forgettable. Exceptions—like Gerhard Richter or the late-period David Hockney—unfortunately only proved the rule, not least because they first produced artefacts in the 1950s. Contemporary post-2000 digital media art works were occasionally interesting but lacked the kind of stickiness (the durable claim on social memory) that great works in general secure. The media arts fused technology and the arts, moving them close to the STEM disciplines and away from the humanities. This intensified the wider tendency for social aesthetics to replace capital-A Art. This took aesthetics out of the galleries and into everyday life and industry. Yet as their ambivalent name suggests, the media arts lacked the kind of strong conceptual identity that marked the modernist era in art. Since the end of the modernist period, art has been dominated either by kitsch concepts or weak concepts. (Murphy, 2015, p. 26)

One might read Murphy's criticism as a conservative defense of modernist art against what came after. Yet, the author does make an important point about the absence of digital art from social memory. There are no widely known digital art 'masterpieces,' the most famous artworks are found in the history books, but that history is still short. After all, it took over seventy years and the genius of artists like Alfred Stieglitz or Eugene Atget for photography to be recognized as an art form. Moreover, aesthetics leaving the galleries may merely signal a return to an earlier paradigm of engagement with art. It is possible that some of the masterpieces in the future digital art canon already exist but are indeed found as computer games or 'demo scene'* productions.

* A computer subculture, particularly active during the 1990s, about competitive programming of real-time audiovisual 'demos' – short for 'demonstrations.'

Murphy's assessment implies a separation between technology and the arts before digital media *"fused"* them. Yet, writing in the early 1970s, Les Levine suggested that *"art was always technology anyway,"* adding *"we are coming back technologically to cave painting"* (Levine, 1971/2011, p. 55). He added:

> The art of the cave painter was a technological art. It was an art of how to do it, how to image the society, how to present society with a working model of itself that would in fact be a 'conceptual tool', a valuable and useful device for the shaping of that society. (Levine, 1971/2011, p. 55)

Not only does the artistic exploration of new technology herald a new beginning for art, but after Latour's (1991) dictum, it can mediate *how* is "*society made durable.*" As Peter Gouzouasis (2006) stated, *"the role of art is to liberate humans from the mechanical status that they impose on themselves by merely using and adapting to technologies."* Indeed, we find that technologists are often likely to regard their work as a kind of aesthetical exploration. According to Warren Sack (2019), computer scientists *"have argued that computing is not a science, software is a literature, and computer programming is a kind of essay writing,"* and *"consequently, contemporary education, research, industry, and technology development all need to change to better recognize how the arts sit at the center of computing"* (Sack, 2019).

One may wonder whether such a position may constitute an overreach, a third-act reversal worthy of a thriller—the arts were pulling strings on technology all along, an amusing but unpragmatic idea. A crucial distinction, therefore, is to note how the arts *can* sit at the center of computing. If we are to recognize that programming is a kind of writing, often it is legal writing, akin to legislation and contracts, as noted by Douglas Rushkoff (2010). Hence, placing the arts within computing, with code as poetry, is not so much a matter of fact as a matter of praxis.

According to Jon McCormack and Mark d'Inverno (2012), *"computing is not seen as a creative subject by the general public or even at schools and universities in many countries around the world,"* even though, paradoxically, *"there is a general consensus that computers can transform and inspire human creativity in significantly different ways than any other artificial or human-made device."*

Patrick Blessinger and Linda Watts (2017) report a gap in the scholarship about learning to think creatively in higher education. As the same authors note, the word 'creativity,' though in use in the late 19th century, is a relatively recent notion that did not enter English lexicography until after World War II. Defined as a concern "*with how to apply imagination in real life, with all its limitations, constraints, boundaries, and other practical considerations"* (Blessinger & Watts, 2017), creativity was primarily associated with artistic endeavors until nearly the end of the 20th century. Though research about creativity dates back to the 1950s, it focused on the individual until the 1980s—i.e., personalities and cognition processes—given an understanding of creativity as an innate characteristic. As the authors state, only from the 1980s did studies of creativity start to focus on social and cultural contexts of practice and began to carry interdisciplinary approaches and foci (Blessinger & Watts, 2017). Creativity was understood as a skill:

> If all students possess the potential for creativity, then teachers do well to consider how their instructional objectives, curricular designs, learning assessments, and institutional structures reflect that fact. (Watts & Blessinger, 2017)

Artists, designers, programmers, but also athletes and gamers instinctively recognize Mihaly Csikszentmihalyi's (1990) notion of *flow*, a feeling of addictive immersion and deep focus in a task which yields creative results 'almost without trying.' Though artists throughout history spoke of divine inspiration and athletes talk of 'being in the zone,' Csikszentmihalyi's key insight is that *flow* is contingent on extrinsic factors: *flow* is found when a challenge matches individual skills so that boredom and anxiety remain at equilibrium and cancel mutually (Csikszentmihalyi, 1990, pp. 110–111). Therefore, a key question for educators in creative fields is how to lend students that feeling of *flow*.

For Blessinger and Watts (2017), the dearth of research about creativity within higher education suggests that the creation of *"authentic creative learning environments is more challenging than it may appear."* Though, as the authors state, creativity skills can be taught intentionally, it is possible those skills remain mostly taught invisibly or implicitly as a tacit component of meeting practical learning objectives. In the fields of the arts and design, 'creativity' can be perceived as tacit, a hollow term; or as a mere signifier of interdisciplinary saliency—as in 'creative code.' Blessinger and Watts (2017) value the *"learning to make connections across domains,"* which might be a central tenet of learning creativity. Still, we do not mean to suggest that transdisciplinarity is a prerequisite; as the authors note, the extant literature on individual creativity indicates that most people's inventiveness is specific to their practical domain and, moreover, related to practitioners' experience and proficiency.

⁂

In the following subsection, we shall summarize the common characteristics of arts and design education—objectives and signature pedagogies—and address its contemporary challenges. In §2.2, we shall focus on the perspectives on digital media within arts and design education. In §2.3, we shall address the creative and critical engagement with computer programming within those disciplines.

## 2.1 *Characteristics of arts and design education*

According to the editors of the Stanford Encyclopedia of Philosophy, the pragmatist philosopher John Dewey posited in *Art as Experience* (1934) that *"expression and art require material used as media,"* and *"an intrinsic connection exists between medium and the act of expression"* (Leddy, 2019). A distinct position, of analytic philosopher Nelson Goodman, held aesthetics as a branch of epistemology (Giovannelli, 2017), but still underpinned by the materiality of the artwork. This emphasis on craft is historically integral to arts education practices (Hausman, 2000; Wilks, Cutcher, & Wilks, 2012), which were borne out of a tradition of apprenticeship alongside experiential learning; life drawing sessions dated back to the Accademia dei Carracci in 16th century Bologna, if not before.

> The challenge for art teachers involves helping others to rediscover the act of perceiving and form making, encircling and informing consciousness and helping students to engage their world with an informed mind and spirit. (Hausman, 2000)

Across all disciplines, 'learning through doing' is essential for promoting understanding (Kember & McNaught, 2007, p. 41). Alison Shreeve and colleagues interviewed 35 higher education educators in the fields of arts and design and identified distinctive pedagogical aspects, all underpinned by *"an expectation that students will undertake their own creative development of the subject,"* experimenting and exploring as to produce *"diverse responses to projects"* that replicate *"the experience of being a practitioner"* (Shreeve, Sims, & Trowler, 2010). Educators avoid taking a didactic approach, creating a *"culture of ambiguity in the learning environment"* that *"students have to learn to negotiate."* Replicating the practitioners' experience implies situating learning in a studio, which the authors regard as *"a space of shared, prolonged, communal activity in which the process of making is visible and a focus for comment and debate".* In addition, Shreeve and colleagues (2010) add that many of their interviewees noted that the conditions of the studio environment have a *"dialogic relation to the student and the making of creative artefacts,"* and thus shape the work taking place there.

Learning arts and design *"involves living with uncertainty and unknown outcomes:"* educators *"have to be prepared to go along with students' ideas and to trust them"* (Shreeve et al., 2010), a notion that is antipodal to the pervasive distrust of students' contributions in many other disciplines. Unlike in fields such as, e.g., law or medicine, providing students with a shared body of canonical knowledge is not the main thrust of artistic education; it is, instead, to *"develop independent creative practitioners"* (Shreeve et al., 2010). Learning is a social process that frequently emerges from the informal

interactions between students and between students and educators,[*] which are often discussions and formative feedback about the 'work in progress.' Obviously, this relates to the key aspects of materiality and physicality: *"the engagement with material itself and with development of ideas through sketchbooks, drawing, performance, recording of process"* which all *"involve the emotions and the senses"* (Shreeve et al., 2010). The resulting artifacts not only invite discussion but are also the central component of learning assessment, either after class presentations or public exhibitions. Moreover, in alignment with the objective of developing independent practitioners, visibility leads to a preoccupation with the audience—be it the class colleagues in earlier exercises, or the anonymous audience of an external exhibition or festival.

These signature pedagogies, in which students perform as quasi-practitioners, point toward the continued centrality of experiential learning (Hausman, 2000; Wilks et al., 2012). Giving the training of performing artists as an example, Shreeve and colleagues (2010) tell us, *"the emphasis on doing [...] is about understanding what it means to be a skilled performer, with all the socially situated understanding that comes with that."* Therefore, learning arts and design is ontological rather than epistemological; moreover, *"determined by the nature of disciplinary knowledge, which is provisional, unstable and has constantly changing ideas about what is important, new or worthy of investigation"* (Shreeve et al., 2010).

### 2.1.1 CONTEMPORARY CHALLENGES TO THE SIGNATURE PEDAGOGIES

In a separate paper, Alison Shreeve (2011) reviews the recent challenges to the signature pedagogies of arts education. A greater focus on the transparency of higher education, though theoretically benign, has eroded the preexisting *"culture of trust"* within arts education institutions; indeed, implementation of the Bologna Process, which we will discuss in 4.1, brought emphases on standardization and accountability, which may threaten the *"ontological approaches to learning, likely to be replaced by those that are more directly measurable"* (Shreeve, 2011).

The growth in higher education participation during the 2000s was often accompanied by managerial preoccupations instead of an investment in the educational experience: class sizes increased as contact hours were often reduced dramatically.[†] This led to additional pressures on the studio. Spaces became more crowded, and students had lesser access to specialized equipment (e.g., from drafting tables to ceramics ovens), which, moreover, was often replaced by general-purpose computers. As Shreeve (2011) notes, *"expensive material processes [...] are likely to disappear in favour of cleaner,*

[*] Shreeve et al. (2010) note the key role of the interactions between students at different levels of experience. The researcher notes the unfortunate frequent separation between courses and students of different years due to the conditions at institutions he had taught, which was recognized in docent meetings as a problem to be addressed.

[†] The researcher's own experience, though anecdotal, illustrates this: his Film and Video licentiate, completed during the early 2000s, had been a 4-year course in which each year had 36 weeks of classes, and each week had 26 or more hours of sessions. Ten years later, while teaching at the same institution, the licentiate was a 3-year course divided in 15-week semesters, and each week only had 20 hours of face-to-face sessions; by the time those students graduated they had spent about half the time attending lessons or practicing in the studio. Hence, the growth in master's degrees.

*smaller, less resource intensive digital media-based subjects,"* moreover as institutions, though not explicitly so, often pass on the onus of investing in equipment to the student.

The managerial impetus has possibly flattened in the years since Shreeve's assessment, as the last decade was characterized by consolidation rather than an acceleration of the previous decade's trends.* Though digital media contributed to the erosion of arts education's signature pedagogies, the greater interdisciplinarity brought forward by media convergence and some economic preoccupations with creativity and efficiency (i.e., 'synergies') also released, as we shall see in the following pages, some of the signature pedagogies of arts education into other domains.

⁂

Given the brisk pace of technological progress and the unstable process of media convergence over the last few decades, Sandra Palhares (2018) places arts education at a crossroads. *"Though the Visual Arts tend to be accomplices to paradigmatic changes,"* the author tells us, "*it is also certain they are a fertile field for aesthetic experimentation that enables a necessary broader understanding of the world and foment the needed participative, systemic, and disruptive education"* (Palhares, 2018).

## 2.2 *Perspectives on the encounter with digital media*

Despite the expectation of artists' leadership on the engagement with new technologies and media (cf. McLuhan, 1964/2008), the relationship has been fraught, as seen in the previous section. In that context, arts education has been, for the most part, conservative (Bailey, 2011), if not outright resistant, to engaging with the possibilities brought by computing (Wilks et al., 2012). Radclyffe-Thomas (2008) narrated some of the causes:

> Reluctance to engage fully with ICT may result from teachers' pedagogical belief that the computer does too much for the student, or that the computer itself is a barrier to students' artistic expression with students using 'found material' [...], or getting carried away with what the technology can do, losing touch with the source which inspired them originally and losing sense of the outcome aesthetic. (Radclyffe-Thomas, 2008)

Though these fears are readily identifiable among arts and design educators still today, disengagement from digital media yields a higher price, as artists and educators have admonished for decades. Take video artist Bill Viola:

> Despite the anti-technology attitudes which still persist [...], the present generation of artists, filmmakers, and video-makers currently in school, and their instructors, who continue to ignore computer and video technology, will in the near future find that they have bypassed *the* primary medium, not only of their own fields, but of the entire culture as well. It is imperative that creative artists have a hand in the developments currently underway. (Viola, 1982/2003)

* However, at the time of writing, the future of arts education practices after the COVID-19 crisis remains unpredictable.

As Ilana Snyder and Scott Bulfin (2007) noted, digital media are not antithetical to traditional forms of art, *"but their natural ally."* Institutions may turn to digital media in order to conserve resources, but as Kylie Peppler (2010) notes, by themselves, new technologies do not imply the replacement of traditional arts education: traditional ideas and educational goals can be extended into the new media.

One may approach the relation between arts and technology from the opposite direction. Peter Gouzouasis (2006) regards technology education as inextricable from artistic exploration. From that perspective, the author sees commonplace emphases on teaching employable competencies as ironic, as they often view the arts as an inferior 'investment' while maintaining "*technology education is central to new and emerging curricula."* Brown and Nelson (2013) note the digital arts sector benefits universities' innovation and research agendas, and Palhares (2018) adds that the embeddedness of the arts in the broader societal and economic contemporaneity justifies the continued development of arts education.

Snyder and Bulfin (2007) also placed the relationship between arts education and the new literacies as the key question for future research in the field. Key aspects of arts education, such as material practice, are integral to gaining broader computational or procedural literacy (cf. Bogost, 2005; Vee, 2013). Such competencies are especially salient within the applied arts, which demand from practitioners a persistent engagement with technology. Brian Johnson (2017) calls for teaching computational thinking skills to designers and architects. Though we summarized different views of computational literacy in the previous chapter (see **2**.3), it is worth reproducing the author's description of how designers' often practice algorithmic thinking, which calls to Bogost's (2005) notion of procedural literacy:

> Weaving patterns on looms, cooking, cutting or bending patterns in metal and wood, bricklaying, parquetry—all of these utilize repeated patterns of action. Designers often acknowledge thinking systematically, working from some set of principles or rules to a complete design. (Johnson, 2017)

The author adds that design and architecture curricula *"often view digital technology as simply another medium in which to complete traditional work within the academic program and to secure employment"* (Johnson, 2017). Moreover, higher education ought to commit to 'lifetime learning,' with Johnson adding a crucial point about the fragility of teaching competencies in specific tools: *"a university-level design computing course needs to offer knowledge with a longer shelf-life than most software study guides"* (Johnson, 2017). One might say skills in using specific tools may only last until the next big software update,[*] but literacy is for life. Johnson's point is not at all antipodal to the expectation of lifelong learning from engaged practitioners; it instead reinforces it. Indeed, teaching whichever knowledge underpins the operation of specific tools is essential for filling that void of knowledge that Cabeleira (2018) saw as problematic in most recommendations of *'teaching to learn'* (see **2**.2.2).

[*] Disruptive updates or deprecations—e.g., the 2011 update of the Final Cut Pro X video editing software or the end of lives for Apple HyperTalk, Macromedia Director, or Adobe Flash—happen frequently enough that they ought to warn higher education curriculum designers from placing an emphasis on teaching *specific* software tools.

⁂

Johnson (2017) also asks educators to *"jettison the comforting notion that CAD is 'just another pencil' once and for all,"* calling for education to consider the 'side-effects' of specific digital tools; in other words, to teach an awareness of affordances (cf. Norman, 1988). When engaging with technology, arts, and design, educators ought to have an answer to whether *"does the student control the medium, or does the medium control the student"* (Ettinger, 1988) and *"when is the computer considered to have had 'too much' involvement in the process of making art"* (McCormack & D'Inverno, 2012).

## 2.3 *Computer programming within arts and design education*

> Along with teaching people to use computers comes the responsibility to direct a focus on critical study of the values underlying information-based technology, and simultaneously of the values underlying our field. (Ettinger, 1988)

As we have seen in §1.2, the growing popularity of GUI software tools for arts and design was accompanied by little awareness of their limitations. Applications such as Adobe Photoshop and Macromedia Freehand reduced computers to augmented (and indeed useful) brushes and pencils. Though Linda Ettinger questioned, as early as 1988, whether the use of such tools narrowed students' understanding of the digital medium, the question remains just as pertinent nowadays, if not more. McCormack and D'Inverno (2012) note the seldom examined dominance of the keyboard and the mouse (to which we may add the capacitive touch surface) as interface paradigms.

John Maeda, one of the early critics of the uncritical use of software tools, was also a proponent of writing custom computer programs for visual design work. At the MIT Media Lab's Aesthetics and Computation Group, Maeda led one early effort—Design by Numbers—to create a compact programming tool for designers and artists to generate computer graphics* (Maeda, 1999). The success of this early experiment inspired two of Maeda's students—Casey Reas and Ben Fry—to begin development on Processing, arguably the most successful of *creative code* tools and which we shall describe in greater detail in the next section. The ethos of creative code is summarized in Daniel Shiffman's introduction to his Processing textbook:

> What can you make, what can you design if, instead of using someone else's tools, you write your own? (Shiffman, 2008, p. ix)

The appearance of a new generation of programming languages aimed at artists and designers lets educators deemphasize the weight of software tools in curricula while allowing a greater dedication to computational literacy skills with a 'longer shelf-life' (cf. Johnson, 2017). Aaron Knochel and Ryan Patton (2015) contend that *"computational thinking through critical digital making should be embedded in the art curriculum to equip students with coding skills important to digital literacy, to expose the computational world surrounding all of us,"* while Nick Montfort (2016) maintains that artists should learn

* Though Design by Numbers resembles a simpler implementation of LOGO, its use of Cartesian coordinates positions it to an altogether different audience.

*"exploratory programming, which involves using computation as a way of inquiring about and constructively thinking about important issues."* Acknowledging the *"burgeoning online communities of tween/teen game designers, animators, and digital artists,"* Kylie Peppler (2010) also explicitly positions programming as an essential skill to be acquired through arts education. Distinctions between computational thinking, computational literacy, or procedural literacy, which we reviewed in 2.2.3, though substantive to how institutions may include computer programming in arts and design curricula, do take a backseat in the central message of these authors: artists should learn to code.

Indeed, they have. Writing in 2013, Lev Manovich notes the previous decade's progress in the acceptance of computer programming within the arts:

> In the 2000s, the number of students in media art, design, architecture, and humanities who use programming or scripting in their work has grown substantially—at least in comparison with 1999 when I first mentioned 'software studies' in *The Language of New Media*. Outside of culture and academic industries, many more people today are also writing software. To a significant extent, this is the result of new programming and scripting languages such as ActionScript, PHP, Perl, Python, and Processing. (Manovich, 2013, p. 16).

Though Knochel and Patton (2015) argue for the centrality of schools in shaping the critical exploration and understanding of digital materials, they recognize the crucial influence of informal spaces for learning programming and expressing creative practices—e.g., hackerspaces or makerspaces (cf. Halverson & Sheridan, 2014b; Tan, 2019), workshops, online lessons. Montfort (2016) also mentions the key role of social influence, noting, *"those who decide to become new programmers often find the motivation to do so in their encounters with computers and in interacting with others who are using programming to think about interesting problems."*

According to Peppler (2010), research *"indicates that youth intuitively learn the 'big ideas' of computer programming through media art making,"* provided that they have enough time to explore. Gouzouasis (2006) emphasizes the need for the exploration of digital materials to start in earlier schooling, noting that "*the training in the use of new media applications for graphics, animation, music, sound, and film production begins at postsecondary institutions, possibly too late in an individual's life to reconsider the cohesiveness of arts and technology."*

### 2.3.1 CREATIVE CODE PEDAGOGY

The inclusion of computer programming within arts and design curricula involves more than merely employing common computer science pedagogical strategies (see 2.3.2) to the introduction of languages such as Processing. As Michael Mateas stated:

> Teaching programming for artists and humanists shouldn't merely be simplified computer science with lots of visually engaging examples, but rather an alternative CS curriculum. [..] For new media students we need to emphasize that, while programming does have its abstract aspects, it also has the properties of a concrete craft practice. (Mateas, 2005)

Moreover, authors such as Olivia Gude (2004) rightly point that art education ought to be suspicious of emphases on formalism. What is interesting about a digital rendering of Sol Lewitt's *Wall Drawing #392* is not how it fills the screen with random lines and is conducive to additional formal exploration (e.g., looking for harmonies with different colors or lines of different angles); instead, it can illustrate and lead students to think about stochastic processes and grids, and to question their meanings and contexts. The temptations of formalism ought to be addressed while teaching creative code, lest it becomes the opposite of its purported role of enabling critical technological practices.

Kylie Peppler (2010b) tells us that *"the practice of computer programming in the context of the arts begins with an intention (be it visual, audio, gestural, or interactional)"* that guides the exploration of a programming language's affordances. This contrasts with conventional computer science teaching, *"where new programming commands are presented in the contexts of math or science problems that are introduced by the instructor"* (Peppler, 2010b). Indeed, pedagogical practices which are intrinsic to arts education are nowadays the focus of active experimentation across other fields (cf. Donnelly & Grushka, 2015; Halverson & Sheridan, 2014a; Palhares, 2018). Aesthetic forms of inquiry—e.g., video production—found their way into science classrooms (Halverson & Sheridan, 2014a), while studio-based learning and other collaborative pedagogies have been applied in computer science and engineering education (cf. A. S. Carter & Hundhausen, 2011; Hundhausen, Narayanan, & Crosby, 2008).

Testifying to the interdisciplinary relevance of pedagogical strategies born out of the art studio, Knochel and Patton (2015) note the influence the observation of art classrooms had on Seymour Papert's development of LOGO and later advancement of constructionist pedagogy, which places students as designers and producers of artifacts (cf. Papert, 1985; Wilensky & Papert, 2010). As Peppler tells us:

> Constructionism thus serves the goals and aims of arts education because it focuses, on one level, on the design of artifacts rather than on the use of artifacts and tools, as is commonly found in other sociocultural theories of learning, and yet, on another level, it focuses on the bidirectional relationship between an individual and a community of learners. (Peppler, 2010a)

Despite being borne out of arts education, Peppler adds that constructionism has had a reduced influence on arts education. On the one hand, this might be one case of the adage, attributed to McLuhan, of *"the fish knowing nothing about water."* On the other, it maybe relates to how the aforementioned economic factors and emphases on standardization have placed constraints on the art studio while constructionism found its way to other disciplines: as McCormack and D'Inverno (2012) note, *"companies are now beginning to recognise that they want technologists who can think like artists,"* and traditional teaching methods based *"on engineering-based problem-solving will not be sufficient for the new challenges of software development."* Recent research, for instance, into the use of Lego bricks in both learning fashion design (cf. James, 2013) and software engineering (cf. Kurkovsky, 2015) testifies to this development.

### 2.3.2 THE RESEARCH GAP

Knochel and Patton (2015) recognize that the field of arts education has remained a distant venue for research about the teaching of computer programming. We used the software application Publish or Perish 6.32[*] to search academic literature search engines, and indeed confirmed the authors' assessment.

Though variants of the search phrase[†] "*'art education' AND 'computer science'*" yielded over 15,000 literature entries in Google Scholar (excluding citations) up to 2018, we are unsure about the reliability of the total as the relevancy of the search results appeared quite poor. Indeed, the same query only yielded seven papers in the Web of Science Core Collection and 82 papers in Scopus, which attests to the limited scholarship that bridges both fields. Combining the search for literature about arts education with terms such as *'computing education,' 'teaching computer programming,'* or *'learning to code'* yielded even fewer results, most of which already produced by the initial query.

Variants of the now common expression *'learning to code,'* without additional terms, yielded 3,100 results in Google Scholar,[‡] yet only 22 papers in the Web of Science (all published after 2012) and 40 in Scopus (36 of which published after 2012). The same pattern repeated itself when combining terms related to computing with other terms related to arts and design education. Searching for variants of the phrase *'design education'* within the *'computer science'* search results yielded a total of 20 papers in the Web of Science and 530 papers in Scopus (many of which were related to the design of computer systems and therefore unsuitable to our purposes); doing so with variants of the phrase *'media studies'* yielded 12 papers in the Web of Science and 193 in Scopus; *'music education'* yielded only four papers in the former and 93 in the latter database.

[*] harzing.com/resources/publish-or-perish/

[†] We adapted our queries to the characteristics of each search engine—e.g., *'(art OR arts) education'* for Google Scholar, *'art* education'* for Web of Science.

[‡] Again, most results had little relevancy, as the word 'code' often yielded a significant number of articles relating to qualitative research practice.

> Environments for programming (typically, BASIC) became easily available to the everyday home computer user in the late 1970s and 1980s, but as powerful, complex IDEs and compiled languages have been developed, programming has in some cases become more difficult to access. People do not encounter it casually in the ways they used to. (Montfort, 2016)

In his 1980 book *Mindstorms*, Seymour Papert likened the languages in common use at the time, such as BASIC or Pascal, to the QWERTY keyboard layout: dated products of the constraints under which they were designed, entrenched due to inertia (Papert, 1980, pp. 32–35). The plethora of languages derived from C, which dates to the early 1970s—e.g., C++, C#, Objective-C, D, Java, PHP—testifies to the path dependence that had often shaped innovation in computing; it is also logical that the needs of computer engineers would shape most languages' development.

Despite early exceptions—e.g., BEFLIX—purposeful development of languages for the needs of practitioners outside of what Ted Nelson termed the 'priesthood' (cf. Nelson, 1974/1987) had been rare until computers found their way in practitioners' studios. LOGO, devised for graphics production by school-age children without the knowledge of Cartesian coordinates, clearly stood out. Two decades thence, Casey Reas and Ben Fry developed Processing after their work with John Maeda in Design by Numbers, and out of a recognition that owes as much to Papert:

> It's an unfortunate fact that many languages used within the arts were not originally designed for those areas. The software desires of designers, architects, and artists are often different from those of scientists, mathematicians, and engineers. (Reas et al., 2010, p. 21)

A tight feedback loop is one such key software desire. Christopher McLean and Geraint Wiggins, after the writings of the painter Paul Klee, stress the importance of feedback in the creative process: *"making a mark on canvas, perceiving the effect, and reacting with a further mark [...] a tight creative feedback loop places the artist close to their work, guiding an idea to un-foreseeable conclusion through a flow of creative perception and action"* (McLean & Wiggins, 2012). Though creative computing practices are mediated by code, and the feedback may not be as immediate as that of a mark on a canvas, the authors nevertheless echo Frieder Nake (2012) in noting how this gap carries the possibility of surprises, also an essential contribution to creativity (McLean & Wiggins, 2012).

The gap between writing code and output is dependent on the specific affordances of the language; McLean & Wiggins (2012) mention *interactive programming,* which automatically modifies the resulting output as the code is written or devised (e.g., using visual languages); in *live coding*[*] performances, the writing of the program that generates the sounds or the images is made into a visible part of the show.

⁂

[*] We note the term *'live coding'* may also refer to the pedagogical method of using a projector to demonstrate the writing of code examples to the class.

If one reckons that programming languages are still tools, albeit flexible and expressive, one sees serious artists looking to engage with the boundaries of what is possible in the medium would need to experiment with different languages. Still, the initial contact with computer programming is crucial. In the following subsection, we shall briefly address the issue of language selection for introductory teaching. Then, in §3.2, we shall review programming languages commonly used for arts and design education nowadays.

## 3.1 *Criteria for educational programming languages*

Parker, Beard, and Davey (2019) identify a series of criteria for selecting programming languages that are adequate for teaching in higher education: acceptance in academia (including whether quality textbooks are available), industry penetration, cost, and technical characteristics (e.g., system requirements) are important, but so are the pedagogical features, its design and intent, whether it had student-friendly features (e.g., an integrated development environment or debugging functionalities), whether its paradigm supports the teaching approach, and whether support is available and instructors are familiar with the language. Though these criteria envision teaching computer science and engineering courses, we find them perfectly transferrable to the fields of arts and design.

Parker and colleagues (2019) also identify two distinct approaches to applying their criteria: pragmatic approaches favor industry penetration and employability, whereas pedagogical approaches are more concerned with developing students' computational thinking skills (cf. Wing, 2006) and thus favor the selection of languages explicitly aimed at novices, even if they have little use in the industry.

Educators favoring a pragmatic approach should still mind Brian Johnson's (2017) warning about the shelf life of knowledge about specific software tools. Institutions ought to also mind the same preoccupations before committing to purchasing expensive software licenses that may anchor the teaching of deprecated and unsupported software. A recent high-profile example was the 2017 announcement and the 2020 discontinuation of Adobe Flash[*] and its associated ActionScript language. As we will see in the next chapter, ActionScript was nonetheless still being taught at several arts and design undergraduate courses at Portuguese institutions in 2018/2019. Programming languages with proven and widely used open-source or free software[†] implementations may have a longer shelf-life, independent from the whims of individual software vendors—indeed, see the continued successes of C, Python, JavaScript,[‡] or PHP.

[*] We refer to the Flash platform. Adobe continues to update the Animate authoring tool, which now supports JavaScript.

[†] *"Free as in freedom,"* which means the possibility of inspecting and changing the source code (i.e., 'open-source') and the possibility to redistribute modified versions.

[‡] Implemented in all major web browsers, most of which are open-source projects.

## 3.2 *Common languages for novices and practitioners*

The development of programming languages aimed at novices or intended for users outside of computer science disciplines has been intense during the past two decades. In a 2003 review, Mark Guzdial had identified three families of languages for learning: LOGO (cf. Papert, 1985) and similar programming environments; rule-based languages such as Smalltalk (cf. Kay & Goldberg, 1977/2003) or Prolog; or the easier to learn amongst the traditional imperative languages, such as BASIC or Pascal. In the IEEE's 2016 ranking of language relevance, only Visual Basic was listed in the top twenty (at number 18); Delphi (a variant of Pascal) was listed in 27$^{th}$ place, and Prolog in 37$^{th}$; LOGO and Smalltalk were unlisted (IEEE, 2016). Moreover, as we shall see in chapter 4, none of the languages Guzdial had identified in 2003 now enter contemporary introductory programming syllabi in the arts and design undergraduate courses offered by Portuguese higher education institutions.

As we have seen in 2.3.1, the resurgence of concerns with computing education in the mid-2000s owed much to a renewed focus on STEM disciplines and new initiatives aimed at mitigating digital divides through the promotion of computer programming (Blikstein & Moghadam, 2019). Yet, the release of new programming languages aimed at novices and disciplinary contexts was a major factor as well. Moreover, the nascent web design and development field also brought code to the forefront of the contact with computers—anyone could peek 'under the hood' of a webpage by choosing 'view source' in the right-click menu. [*] Over the following pages, we shall review the major programming languages adopted in arts and design education.

### 3.2.1 GENERAL-PURPOSE PROGRAMMING LANGUAGES

C, Java, Python, and C++ respectively occupy the four places at the top of the IEEE's 2016 programming language ranking. Moreover, Beard (2019) considers Java, Python, C++, and C# as the best languages for teaching computer science and information science novices. C, designed by Dennis Ritchie at Bell Labs in 1972, is still a ubiquitous language used in contexts as diverse as embedded systems or programming high-performance applications, thanks to standardization and the wide availability of major free software implementations—e.g., GCC or Clang—which also support C++, a variant designed by Bjarne Stroustrup in 1985 which extends C with object-oriented functionality.

Python, designed by Guido van Rossum in 1991, emphasizes code readability and is a higher-level programming language; it is thus considered the easiest for computer science novices to learn (Beard, 2019). Reference Python implementations are also free software and distributed for many different systems.

Whereas C and C++ emphasize performance and applications need to be compiled for the particular type of computer systems where they will run, Java, designed by James Gosling at Sun Microsystems in 1995, was aimed at the online distribution of applications that could run on any computer system, as long as they had the Java *virtual machine*

[*] Nowadays, most websites' plethora of JavaScript dependencies and use of code minimization and obfuscation techniques complicates the exercise.

installed. Nowadays, Java is one of the main languages for Android app development alongside Kotlin, a recent variant.

This description of general-purpose languages would be gratuitous if not for their connection to the programming tools aimed at disciplinary practices. Processing programs run in the Java virtual machine; indeed, Processing can be described as a set of Java libraries for working with graphics and sound that are accompanied by a tool that converts the simplified Processing syntax to Java source code at the time of execution. Performance is thus one of Processing's weaknesses. Artistic practitioners familiar with C++ often use the OpenFrameworks or Cinder libraries* instead. Python users may also employ the Pygame† library, which though intended for game development, includes ample multimedia functionality.

### 3.2.2 WEB PROGRAMMING

The JavaScript language specification is one of the core web standards alongside the Hypertext Markup Language (HTML) and the Cascading Style Sheets (CSS). HTML documents, first specified by Tim Berners-Lee in 1990, are text files that include tags for structuring their contents and for placing inline multimedia content or links to other files. CSS, standardized by the World Wide Web Consortium in 1996, is a language for laying out how the specific sections of the associated HTML documents should be presented (e.g., placement, color, font). Though design students and practitioners have frequent contact with HTML and CSS, neither are programming languages.

However, JavaScript is. Web browser software in common use includes JavaScript‡ interpreters since Netscape first introduced the language designed by Brendan Eich in 1995. Web creators may embed JavaScript code within the HTML documents presented to clients, which is thus executed in users' web browsers. JavaScript code may manipulate the entirety of the webpage being presented, from simple interactions (e.g., making a menu visible when the mouse interacts with a link) to complex single-page applications (e.g., Google Maps). There is also a rich ecosystem of JavaScript extensions, including P5.js, which aims to allow the execution of Processing source code (after few modifications) within a webpage.

JavaScript has also found its uses outside the web browser. Several specific tools, such as Adobe Acrobat, adopted JavaScript as their embedded scripting language. Perhaps more significantly, Node.js, first released in 2009, implements JavaScript as a general-purpose programming language, adequate for web server use, among other things. This places JavaScript in an attractive position for learners looking to acquaint themselves with one single language.

Still, one of the most popular programming languages for web development remains PHP (which stands for PHP Hypertext Preprocessor), designed by Rasmus Lerdorf in 1994. Though it can be used as a general-purpose language, PHP was created with the needs of web page creators in mind. Programmers can easily write PHP code to

* openframeworks.cc, libcinder.org

† pygame.org

‡ Despite the naming coincidence, JavaScript has no relation whatsoever with the Java programming language.

verify user authentication or to interact with a database. In addition to its accessibility, PHP is free software, unlike alternatives such as Microsoft Active Server Pages (ASP). Many higher education institutions' IT departments allocate students a web space quota that supports PHP scripts; in addition, commercial web hosting providers often only support that language in their more affordable plans. Hence, PHP is often taught alongside JavaScript to web and communication design students.

### 3.2.3 PROCESSING AND P5.JS

> A good example of a contemporary programming environment, very popular among artists and designers and which, in my view, is close to [Alan] Kay's vision, is Processing. Built on top of the Java programming language, Processing features a simplified programming style and an extensive library of graphical and media functions. It can be used to develop complex programs and also to quickly test ideas. Appropriately, the official name for Processing projects is *sketches*.
> (Manovich, 2013, p. 105)

First released in 2001, Processing was borne out of the work of Casey Reas and Ben Fry at the MIT Media Lab under the guidance of John Maeda, as we mentioned earlier. The designers retrospectively referred to *"a modern Prometheus"* (Reas & Fry, 2018), an hyperbole that may also be a tongue-in-cheek reference to the subtitle of Mary Shelley's *Frankenstein*, but that nonetheless testifies to the aim for the Processing project. The language and associated tools are free and open-source software and are available for multiple platforms at processing.org. The tools include an integrated development environment (IDE) application, shown in Figure 3.3.1, that provides a text editor with syntax highlighting, debugging features, and library management. Though adequate for learners, the use of the IDE is optional and alternative editors such as Atom or Sublime Text also support Processing.

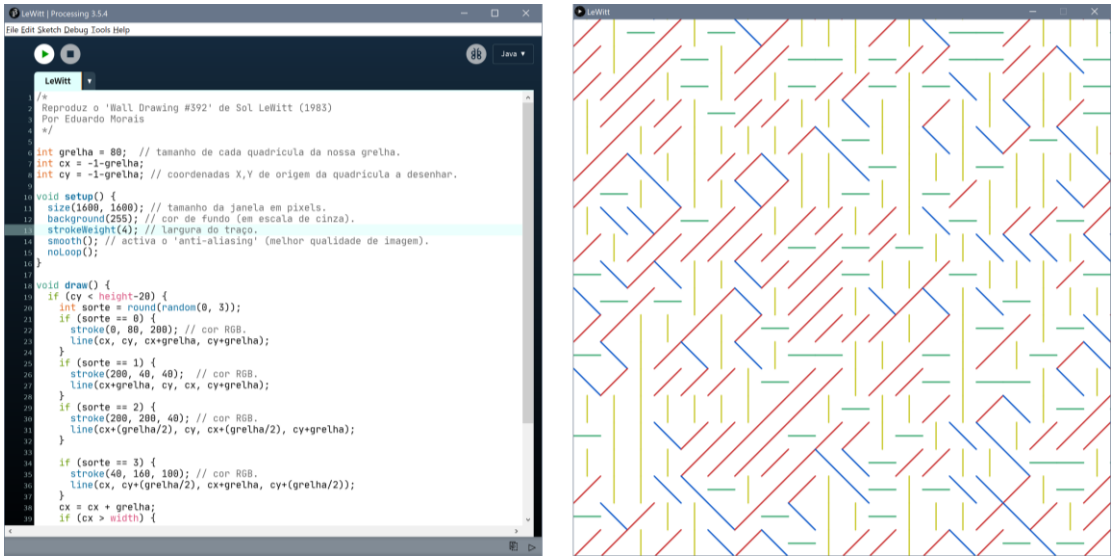



***Figure 3.3.1**. The Processing 3.5 IDE and an output window, displaying a digital rendition of Sol LeWitt's Wall Drawing #392.*

At the time of writing, the latest stable Processing version is 3.5.4, released in early 2020, and version 4.0 is in active development. Processing has a large community dedicated to supporting Processing with programming libraries and educational materials. The IEEE ranked Processing as the 23rd most popular programming language in 2016, which is remarkable considering it is rarely used for the production of software meant for distribution, given its performance weaknesses: as of 2020, Processing only ranked as the 39th most-used language in Github code repositories—66th if only considering those repositories with more than 100 'followers' (Sroczkowski, 2020).

A 2014 study of Design and Multimedia undergraduate students at the University of Coimbra found that those learning to program for the first time with Processing had significantly better educational outcomes than those students who, in previous years, had learned to code with Python (Gomes, Mendes, & Marcelino, 2014). The authors state *"the reason was not language simplicity (on the contrary), but its ability to easily support the development of applications that display drawings, animations and art works"* (Gomes et al., 2014), thus lending credence to the notion that a language's affordances are crucial. Collins (2018) has also shown the benefits of teaching computer graphics with Processing as to motivate struggling computer science students.

Given the aforementioned growth of JavaScript outside of the confines of the web browser, P5.js may well supersede Processing as an educational language in the future. Created by Lauren McCarthy in 2013, P5.js is a JavaScript library also supported by the Processing Foundation. Porting code from Processing to P5.js often requires no more than minute modifications. Yet, P5.js sketches can also take advantage of the JavaScript environment to interact with webpage elements. Like the project that inspired it, there is also a vast ecosystem of extensions—e.g., libraries for supporting and processing audio and video, for user interface design, for physics calculation and simulation, or for communication with external hardware such as Arduino microcontroller boards. In addition, the project also offers a free online development environment at editor.p5js.org that allows users to write, store, and share their 'sketches,' shown in Figure 3.3.2:

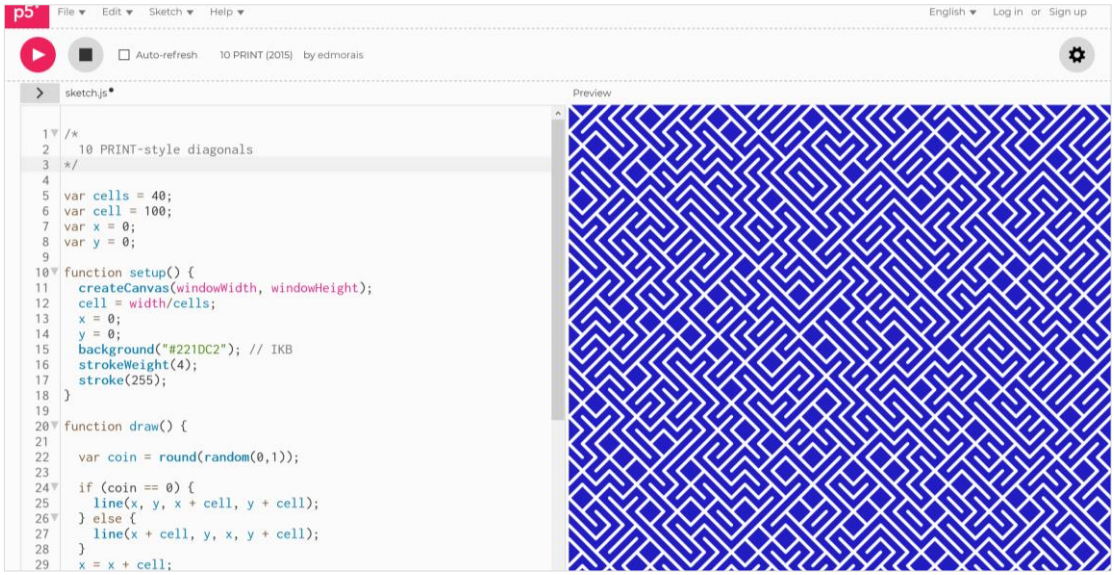



***Figure 3.3.2**. The P5.js online IDE.*

### 3.2.4 VISUAL PROGRAMMING LANGUAGES

Visual 'dataflow' programming* has proven useful across multiple disciplines. Engineers use the visual language within LabVIEW (first released in 1986) to define and represent automation processes more efficiently. Video effect artists using The Foundry's Nuke software (1993) can devise new effects by placing and connecting nodes representing operations between video inputs and outputs; 3D artists using Blender may use a similar system to create generative textures. And as we have mentioned in §1.2, architects and designers may use Grasshopper to devise parametric forms. In these languages, all operations and functions are represented as objects on the screen; rather than writing, programmers draw lines to connect the appropriate inputs (i.e., function arguments) and outputs (i.e., return values) in a process that resembles working with electronics.

One particularly significant visual programming language aimed at artists is Max, shown in Figure 3.3.3. Initially designed in the mid-1980s by Miller Puckette at IRCAM (Institut de Recherche et Coordination Acoustique/Musique), Max acquired a large user community over the last three decades, notably after the release of the MSP extensions for audio signal processing in the 1990s (which is why Max is sometimes referred to as Max/MSP). Though Max is commonly employed to develop music software and to underpin live performances, the release of the Jitter extension in 2003, which added video capabilities, has also popularized Max outside the domain of computer music. We note that Max is commercial, closed-source software. Still, an evaluation version for multiple systems is available at cycling74.com/products/max.

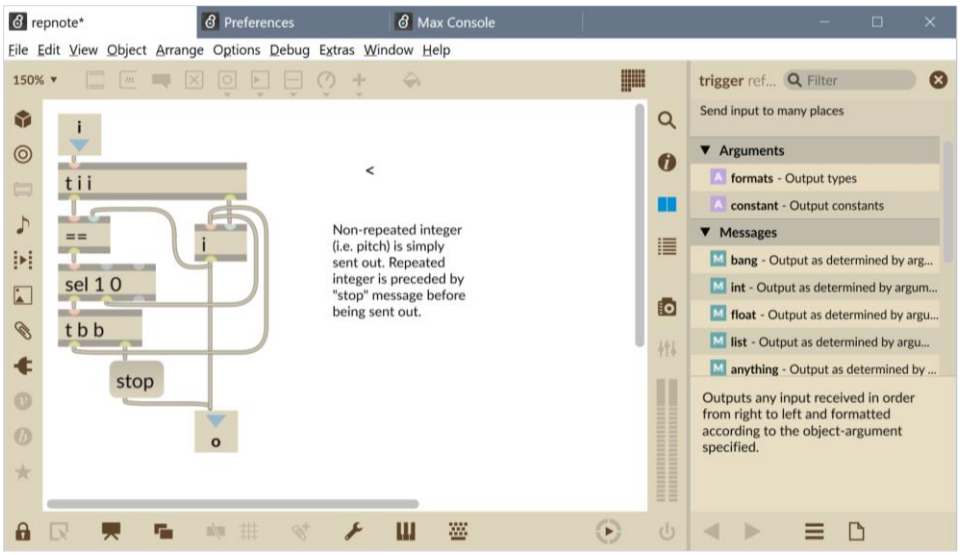



***Figure 3.3.3**. The Max 8 user interface and a small patch.*

There are several other visual languages used by musicians and visual artists which are worth mentioning. Pure Data,† also designed by Miller Puckette, was first released in 1996 and has since remained free software; yet a confusing ecosystem with multiple distinct distributions and uneven platform support complicates its educational use.

* We will address blocks-based languages, another visual paradigm, in §3.2.7.

† www.puredata.info

TouchDesigner by Canadian company Derivative, first released in 2002, is a commercial alternative to Max with a greater emphasis on graphics generation and processing. Apple includes Quartz Composer, a visual language for graphics processing, with its XCode development environment; though popular, it is only available for Mac platforms. VVVV, released by the eponymous German group in 1998, and EyesWeb, produced by the InfoMus Lab in Italy from 1997, are open-source alternatives for Windows which also lack cross-platform support.*

Though dataflow language development continues apace—recent examples include Tooll and Nodes†—their use also raises some objections. Reas and colleagues (2010) tell us, *"visual programming languages make it easy to generate and filter images and sounds, but they are often too cumbersome for writing long, complicated programs"* (p. 19). Marian Petre (1995) also noted that the skills needed to read graphical notations complicate the notion that these are easier to learn. Hence, though visual languages may prove adequate for representing signal processing flows, their surface for exploration is relatively more limited than that of text-based languages.

### 3.2.5 GAME CREATION SYSTEMS

Game creation systems date back to the 1980s. Committed microcomputer users could use tools such as Broderbund's Pinball Construction Set (released in 1982) or Adventure Construction Set (1984) to create customized games for the Sinclair ZX Spectrum, the Commodore C64, or similar machines. To this day, applications such as RPG Maker (first released in 1992) permit enthusiasts to create role-playing games of appreciable depth by using a simple built-in scripting language. Software such as Scirra's Construct (2007) or Roblox (2006) respectively include JavaScript and Lua language scripting as to permit the development of a wider variety of games.

Some game creation systems were developed by video game manufacturers for internal use, and as to finance their continued development, some companies licensed their 'game engines' to third parties. One example is Unreal Engine, initially developed by Tim Sweeney at Epic Games in 1998. Users can design the three-dimensional game environments and then employ either C++ or the built-in Blueprints visual language to program the behaviors for all the objects within those environments. Though nowadays Unreal Engine is freely available for multiple operating systems at unrealengine.com, the software remains proprietary and commercial use requires the payment of royalties.

One of Unreal Engine's main competitors is Unity, first released in 2005. Like the former, the software is available for personal use at unity.com, though commercial use requires licensing. In addition to 3D environments, Unity also includes features for the design of two-dimensional games, and its scripting system only employs the C# programming language developed by Microsoft. Godot, available at godotengine.org, is a free software alternative first released in 2014; games developed with Godot can be sold without the need to pay royalties. Like Unity, Godot includes functionalities for 2D and 3D game development. Its scripting language, GDScript, is a variant of Python.

---

* www.vvvv.org, www.infomus.org/eyesweb_ita.php

† www.tooll.io, www.nodes.io

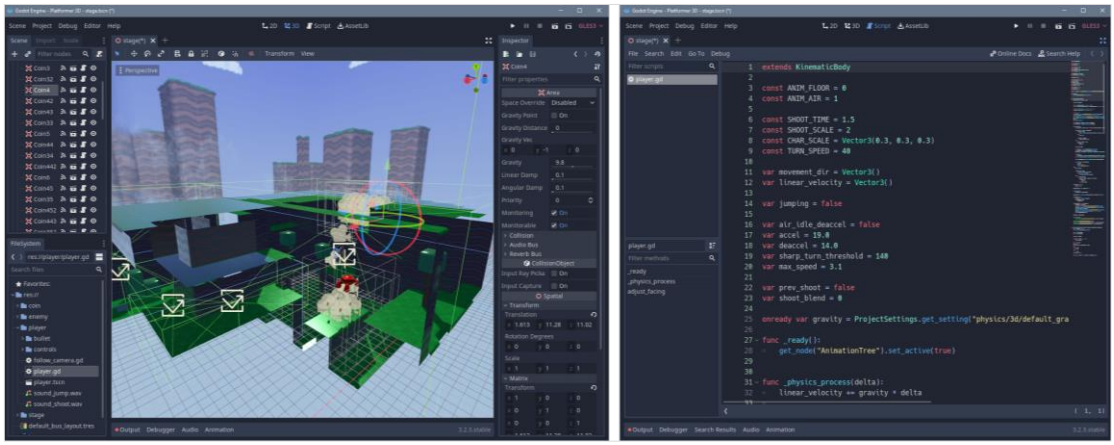


***Figure 3.3.4**. The Godot user interface: 3D mode (left), scripting mode (right).*

Game creation systems have been used in education outside of specialized game design courses. Game engines' wide range of features can be explored in audiovisual projects; indeed, Unity and Unreal Engine have also been used in film and television production. In addition, the multidisciplinary nature of game development, which invokes skills as distinct as storytelling, visual design, audio production, and computer programming, is seductive for setting multimedia group projects.

It might be tempting to introduce computer programming to students already familiar with 3D modeling or design tools from within a similar environment. Yet, the still-nascent research on the pedagogical use of game creation systems suggests that it is not as straightforward. Indeed, though a recent study of Unity use in secondary education found that game design was attractive for most learners due to the broad set of skills they were asked to develop and demonstrate, it also found Unity's steep learning curve was not appropriate for the initial contact with programming (Comber, Mayer, Motschnig, & Haselberger, 2019). One should note the C# scripting language employed by Unity is a lower-level language and not as easy for novices to learn as, for instance, Godot's GDScript. Hence, it is likely that those findings are not generalizable to the use of other game creation systems in programming education.

One rather interesting recent development is the growing popularity of 'fantasy computers' such as the TIC-80, the PICO-8, or the Prism-384[*] among enthusiasts. These are game creation systems integrated with distinct virtual machines that simulate simple computers inspired by the early home computers of the 1980s. Early research has found working with such models to be beneficial in teaching computer engineering (Cruz, Ferrández, Redondo, Álvarez, & Ortigosa, 2019). Given how 'fantasy computers' include tools for graphic and sound design much like the 'serious' game creation systems, they may be helpful for teaching introductory programming aimed at game and multimedia production, provided the curricula then provides time for transferring that knowledge onto the tools of contemporary practice.

[*] tic80.com, lexaloffle.com/pico-8.php, grapefruitopia.itch.io/prism-384

### 3.2.6 HARDWARE TOOLS

A few tools and programming languages allow artists and designers to explore modes of interaction beyond the typical input and output devices and thus examine the familiar mouse and keyboard interface paradigm (McCormack & D'Inverno, 2012). Programmers using Processing or similar languages can use computer vision and signal processing libraries to devise interactions using cameras or microphones. Tangible computing tools branch out even further and propose a continuation of the cybernetic agenda of artistic exploration of technology (see §1.1).

The accessibility of tangible computing tools is not a new development. In the 1980s, several companies sold 'turtle robots' whose movements could be programmed in LOGO (cf. Papert, 1985); significantly, Lego, with their Dacta series of products, was one of those manufacturers. Nonetheless, the contemporary interest in exploring hardware creation is closely tied with the Arduino project, which began in 2005 at the Interaction Design Institute of Ivrea, in Italy.

As Joshua Noble (2009) explains, the word Arduino now refers to a toolkit comprised of three different tools: a range of mutually compatible microcontroller boards, a language based on C, and development environment software that not only aids programming but also interface with the Arduino hardware. The boards can thus be used as independent controllers in electronics projects or as programmable interfaces between a computer—which may be running software made in Processing or any other language that has the requisite libraries—and electronic components such as sensors or actuators. Brucker-Cohen (2013) praises Arduino's accessibility, noting its *"ease of use and rapidly expanding developer community has aided an entire generation of programmers, designers, practitioners, and artists who never thought they would be designing hardware systems."* Nowadays, there are several types and configurations of Arduino and Arduino-compatible boards: bigger one such as the Uno or the Due feature a more powerful processor and, crucially, a wider array of inputs and outputs; smaller boards can be used for specialized applications, such as the LilyPad for the embedding in textiles or the Esplora for devising custom videogame control pads.

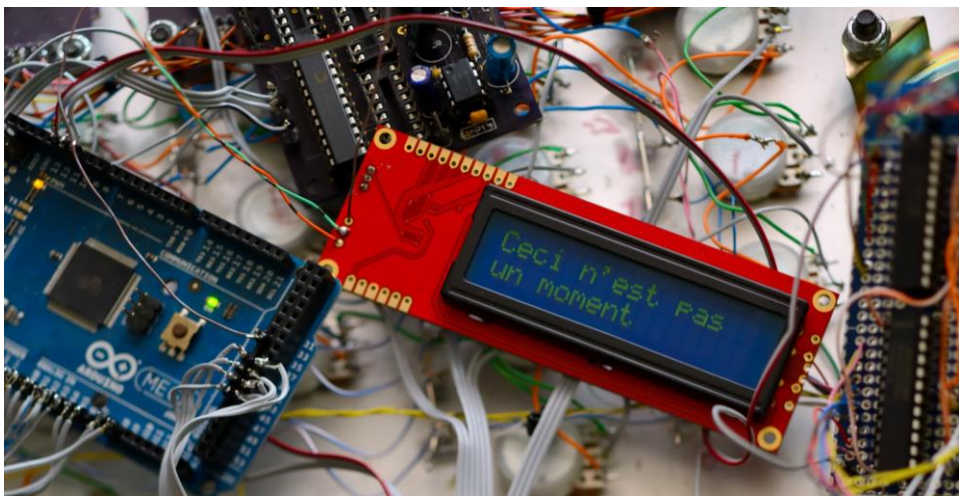



***Figure 3.3.5**. An Arduino board (left) connected to an LCD screen.*
*[Photo credit: Greg Francke / Wikimedia Commons]*

### 3.2.7 BLOCKS-BASED LANGUAGES

Whereas Processing may have taken its inspiration from LOGO but was aimed at adult artists and designers, Scratch—another project also developed at the MIT Media Lab during the early 2000s—was explicitly aimed at children. Designed by a team led by Mitchel Resnick and first released in 2007, Scratch organizes traditional text-based programming in visual objects or blocks, like training wheels for programming. Novices using Scratch do not have to contend with syntactical details; like LOGO, users are not expected to have knowledge of Cartesian coordinates and instead may program the relative movements of the figures they place on the programming canvas (Resnick et al., 2009). The development environment, available online at http://scratch.mit.edu, offers an easy interface for customizing visual and audio elements and for browsing the language's operations and functions.

An earlier blocks-based language also intended for young learners, Alice, was developed at Carnegie Mellon University in 1998. Whereas Scratch retains its original simplicity and accessibility for younger children, Alice—available at alice.org—evolved onto an older audience, as newer versions are intended for 3D game design and merge a blocks-based programming language with characteristics of creation systems such as Unity or Godot.

Snap!, designed at Berkley University and first released in 2011, is also aimed at older novices, as it features a more extensive set of functionalities than Scratch. Like Scratch, Snap! has an online development environment at http://snap.berkeley.edu, where users may create and share their programs, shown in the following figure:

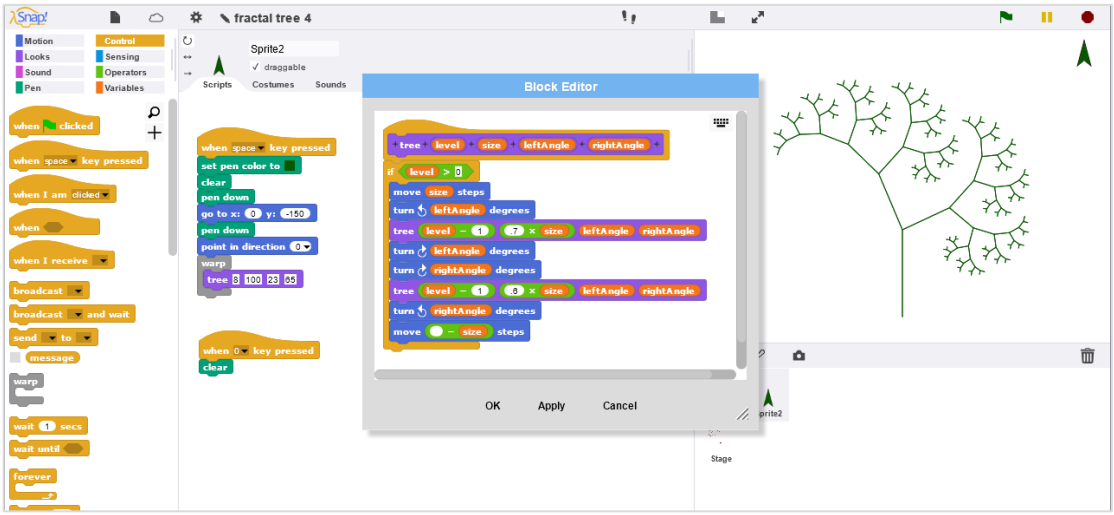



***Figure 3.3.6**. The Snap! online blocks editor.*

A review of studies about the use of Scratch and Alice in introductory computer programming found students were more interested and effective (Lau & Yuen, 2014). Yet, at some point, the training wheels do need to come off. A study of secondary education students, who began learning to code with Snap! before moving to Java, found those learners considered blocks-based programming less powerful, slower, and, in addition, inauthentic (Weintrop & Wilensky, 2015), hence ascertaining the transitional utility of these tools.

※

In the next chapter, we shall address the Portuguese higher education system with a particular emphasis on public undergraduate arts and design courses. We shall then review the inclusion of computer programming on curricula.

# 4

# *Code within Portuguese public arts and design undergraduate education—the present panorama*

★ SYNOPSIS We describe the present state of computer programming education in the arts and design undergraduate courses offered by public higher education institutions in Portugal. In section §1, we set the relevant terminology, present a wide-angle overview of the evolution of the Portuguese public higher education network in the recent decades, and zoom in on the present characteristics of education in arts and design fields according to the qualification schema therein described. In section §2, we zoom in further and identify the specific arts and design undergraduate courses that include computer programming in their curricula, after which we present a systematic review of the curricular unit plans concerning the subject.

## 1 THE PORTUGUESE PUBLIC HIGHER EDUCATION SYSTEM

According to the Portuguese General Directorate of Education and Science Statistics (*Direção-Geral de Estatísticas da Educação e Ciência*), there were, as of the start of 2020, 34 public civilian[*] higher education institutions in the country: 14 public universities, 15 public polytechnic institutes, and 5 schools (*Escolas Superiores*) unaffiliated with any university or polytechnic (DGEEC, 2020c).[†] In total, the Portuguese public and civilian higher education system was, at that date, comprised of 173 organic units (faculties and schools). Upon a closer examination using the PORDATA web interface at pordata.pt, we gathered these organic units were located in just 50 municipalities out of 308, as shown in Figure 4.1.1 (PORDATA, 2020d):

[*] In addition, there are dedicated higher education institutions for military and police forces: the Military University Institute (*Instituto Universitário Militar*) and the Police Sciences and Internal Security Institute (*Instituto Superior de Ciências Policiais e Segurança Interna*).

[†] These include three Nursing Schools (*Escolas Superiores de Enfermagem*) in Porto, Lisbon, and Coimbra; the Tourism and Hotel Management School (*Escola Superior de Hotelaria e Turismo do Estoril*); and the Nautical School (*Escola Náutica Infante D. Henrique*) in Lisbon.

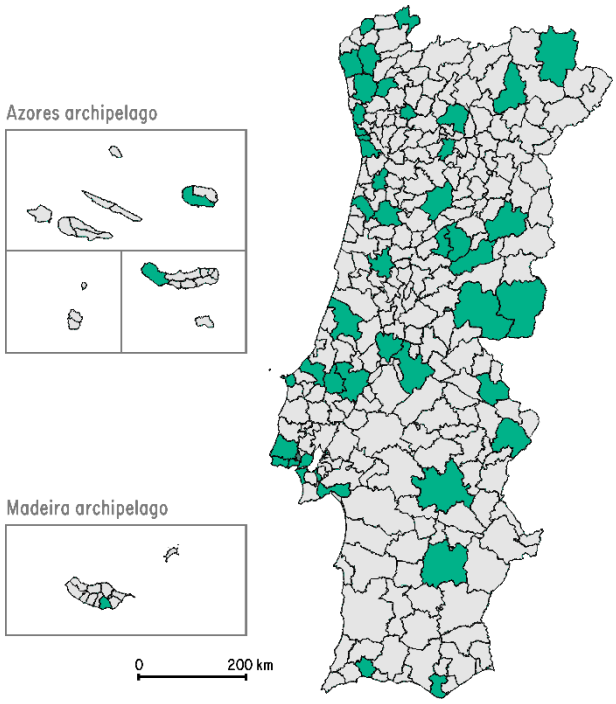



***Figure 4.1.1.*** *Municipal coverage of public higher education (PORDATA, 2020d).*

In subsection §1.2, we will present a short chronology of the development of this public higher education system and report on the evolution of students and academic staff at universities and polytechnics in recent decades. In §1.3, we will focus on the educational offers dedicated to the Arts, Design, and Architecture, as so defined by the institutions themselves within a standard national schema. But first, we shall present the vocabulary we shall employ throughout this dissertation to refer to specific educational institutions, educational personnel, and educational programs.

## 1.1 *Terminology*

We will subsequently employ the following terminology throughout this and the coming chapters in this dissertation. With the differences between the Portuguese system and the higher education systems in English-speaking countries in mind, our choices do not replicate the vocabulary employed in any one such country in particular. We will rather strive for readability and etymological closeness to Portuguese terminology. Therefore:

- FACULTIES are organic units (i.e., formal divisions) within a University, equivalent to how 'colleges' or 'schools' are referred to in some countries. In this dissertation, we will not employ the term 'faculty' to refer to *academic staff*.
- SCHOOLS—from *'escolas superiores'*—are polytechnic organic units. *Schools* are often formal divisions of a Polytechnic Institute (like a *Faculty* within a University), though some independent *Schools* exist within Portuguese higher education.
- ACADEMIC STAFF refers to docents and researchers of various ranks at an academic institution, not including other employees (e.g., administrative, clerical, or custodial staff). Therefore, our usage of *academic staff* is equivalent to 'faculty' in American English.

- COURSES are programs of study that award an academic diploma upon successful completion of all required *curricular units*. The term c*ourse* is, therefore, equivalent in meaning to a 'course of study' in British higher education. Given this dissertation's focus on computer programming, we will avoid referring to 'programs of study' to help readability.
- CURRICULAR UNITS, to be shortened as UNITS, are the individual components of a *course*'s curriculum, typically taught over several scheduled sessions over one or two academic semesters (i.e., terms). Therefore, our usage of *unit* is equivalent to that of the term 'course' within the context of American higher education.

We will also employ the following terms when referring to present-day academic degrees (i.e., after the 2006 implementation of the Bologna Process reforms):

- LICENTIATES are first-cycle degrees with a course plan that is three or four years long. Most *licentiates* have no precedence requirements other than a secondary school diploma (in the Portuguese educational system, attained after successful completion of twelve years of basic and secondary formal education). A present-day Portuguese *licentiate* can be regarded as equivalent to what several other countries call a 'bachelor's degree.'
- INTEGRATED MASTER'S are joint first and second-cycle degrees with a course plan that is five or six years long. As with *licentiates*, applicants to integrated master's programs are most often only required to have completed secondary school. Some institutions allow outside students to apply for the second cycle of an *integrated master's* as long as they completed a preceding *licentiate*.
- INDEPENDENT MASTER'S are second-cycle courses lasting three to four academic semesters. Applicants are required to have completed a *licentiate*.
- UNDERGRADUATE concerns either first-cycle courses that do not require applicants to hold a prior academic diploma or their respective students. We thus may employ the term *undergraduate* as to refer to *licentiates*, *integrated master's*, and the now-extinct *bachelor's degrees,* while excluding other courses offered by higher education institutions (e.g., independent master's, specializations, doctorates, or technical vocational courses—CET, CTESP—that do not grant an academic degree).

Before the Bologna Process implementation, Portuguese higher education recognized *bachelor's degrees* that may be considered equivalent to present-day *licentiates*, as well as 'old' *licentiates* that may be regarded as similar to present-day *integrated master's* degrees. We shall detail these modifications in §1.2.1.

## 1.2 *A short chronology in facts and figures*

For the purpose of establishing a short chronology of the foundation of public Higher Education institutions in Portugal, we found the tables available at Wikipedia perfectly adequate, as the editors profusely cite the relevant legislative acts and statutes ("*Lista de instituições de ensino superior de Portugal*," 2020).

The University of Coimbra, founded in the 13th century, sat for centuries as the sole higher education institution in Portugal. From the 18th Century onwards, several schools were founded in Lisbon (e.g., the Polytechnic School or the Medical School) and Porto (e.g., the Naval School or the Polytechnic Academy) that were merged, from 1911 onwards, into the University of Lisbon and the University of Porto.

Except for the foundings of the Technical University of Lisbon in 1930 and of several independent schools, mostly dedicated to nursing and health services, in the mid-20th century, systematic development of a public higher education network did not start in earnest until the 1970s. The Lisbon University Institute (ISCTE *Instituto Universitário de Lisboa*) was founded in 1972; the New University (*Universidade Nova*), also in Lisbon, the University of Aveiro, the University of Évora, and the University of Minho, primarily located in Braga, were founded in 1973. After the Carnation Revolution of 1974, both the University of the Azores and the University of the Algarve—primarily located in Faro—were founded in 1976. The University of Beira Interior was founded in Covilhã in 1986, the same year the University of Trás-os-Montes *(Universidade de Trás-os-Montes e Alto Douro,* UTAD) was founded in Vila Real. The University of Madeira was founded in 1988, the same year as the Open University (*Universidade Aberta*)—still the country's sole public university fully dedicated to distance learning.

In the period between 1976 and 1986, Portuguese authorities founded polytechnic institutes in all districts lacking a public university: the Beja, Santarém, Setúbal, and Viseu polytechnic institutes were founded in 1979, and the Guarda, Portalegre, and Viana do Castelo polytechnics were founded the following year; Castelo Branco's was founded in 1982, and Bragança's in 1983. The founding of the Leiria Polytechnic Institute in 1987 completed the authorities' initiative. In the meantime, polytechnics were also founded in Portugal's major university cities: in Coimbra in 1979, in Lisbon and Porto in 1985. A new institute for the heavily populated Minho region—the Cávado and Ave Polytechnic (*Instituto Politécnico do Cávado e do Ave,* IPCA)—was founded in Barcelos in 1994. In 1996, the schools located at the Tomar campus separated from the Santarém Polytechnic and constituted an independent institute.

Except for the merging of the Technical University into the University of Lisbon in 2013, the Portuguese public education landscape in 2020 resembles that of 1996 at the institutional level. In addition, though the number of individual organic units—schools and faculties—grew significantly from 125 in 1990 to 173 in 2001, it has remained relatively stable during the last two decades: as of 2020, there were 184* public higher education schools and faculties (PORDATA, 2020d).

⁂

Despite the apparent stability of the public higher education network since the turn of the millennium, the Bologna Process's implementation, intended to harmonize Higher Education across Europe, left a profound mark on the Portuguese system in the late aughts. A brief discussion of those changes is hence warranted.

* Including eleven organic units exclusive to police and military higher education.

### 1.2.1 THE BOLOGNA PROCESS

Before the 2008 deadline for implementing and launching the European Higher Education Area, Portuguese institutions offered two tiers of undergraduate courses: bachelor's degrees were typically three-year courses awarded at polytechnic institutes, while licentiates (*licenciaturas'*) were four or five-year courses awarded at universities. From the mid-1990s some institutions offered second-stage one or two-year licentiate courses (*licenciaturas bietápicas*) to students who had completed a preceding bachelor's degree. A stark division existed between undergraduate and graduate studies, as granting master's and doctorates was an exclusive of universities.

After the implementation of the Bologna Process, codified by the new legal regime of degrees and diplomas created by Decree (*Decreto-Lei*) 74/2006, bachelor's degrees ceased to exist (Portugal, 2006). '*New' licentiate* courses, either three or four years long, now correspond to the first cycle of undergraduate education; hence 'Post-Bologna' licentiate degrees are equivalent to 'bachelor's degrees' awarded by other European countries' higher education institutions. Besides, whereas a 'Pre-Bologna' licentiate may suffice, the professional guilds (*ordens profissionais*) that regulate certain activities under a legal charter (e.g., Architecture or Medicine) now require a master's diploma from applicants. Thus, higher education institutions offer *integrated master's* courses—in practice, not dissimilar to the two-stage licentiates that existed before. Nonetheless, institutions continue to offer *independent master's* courses.

Though universities remain oriented toward research and polytechnic institutes' main focus remains professional education, there are far fewer differences between their educational offers nowadays. Polytechnic institutes have long awarded master's degrees, and recent legislation—Decree 65/2018—will allow them to award doctorates. In addition, Decree 43/2014 reenabled polytechnics to offer technical-vocational courses (*curso técnico superior professional,* CTESP), which last up to two years and do not grant an academic degree; still, students with such a technical diploma can benefit from fast-tracked access or completed credits if they wish to apply to licentiate or integrated master's programs.

⁂

Coincident with the Bologna Process implementation, there were deep cultural changes within Portuguese institutions, especially within public universities. According to António Nóvoa, rector of the University of Lisbon at the time, a *"discourse of urgent modernization took over universities after 2006/07"*, which emphasized *"short-term concepts and practices"* such as employability and entrepreneurship (Nóvoa, 2018). As we described in chapter **2**, that discourse might be rooted in the processes of media convergence and digitization coinciding with liberal policies. And as we will see in §2.1.2, that period indeed coincides with a technical turn within arts and design education. We agree with Nóvoa's admonition that the discourse of modernization may trap institutions *"in a rushed present and away from responsibility toward the future,"* a notion we will keep throughout our research and which we will revisit in chapter **8**.

### 1.2.2 ACADEMIC STAFF IN RECENT DECADES

According to data published by General Directorate of Education and Science Statistics (DGEEC), academic staff employment has remained consistent since the end of 2001,* when nearly 24,300 were teaching at public universities and polytechnic institutes. A small uptick began in 2014, from nearly 24,500 academic staff at the end of that year to almost 27,300 by the end of 2018. According to DGEEC's data, over 16,600 academic staff were then employed at public universities, and over 10,600 were employed at public polytechnic institutes (DGEEC, 2020a).

In contradiction to women's greater academic qualification, already evident in the 2001 Census, as we will describe in §1.2.3, DGEEC's data shows that men have kept the majority of teaching positions until the present and that the road towards women's true representativity in academic employment has been slow—from 40.6% of positions in 2001 to 44.7% in 2018, as shown in Figure 4.1.2. The charts also show that academic staff aged between 50 and 59 went from 17.6% in 2001 to 31.2% in 2018, and positions held by academics over sixty went from 5.3% to 15.0%, which constitutes undeniable evidence of the aging of teaching staff (DGEEC, 2020a).

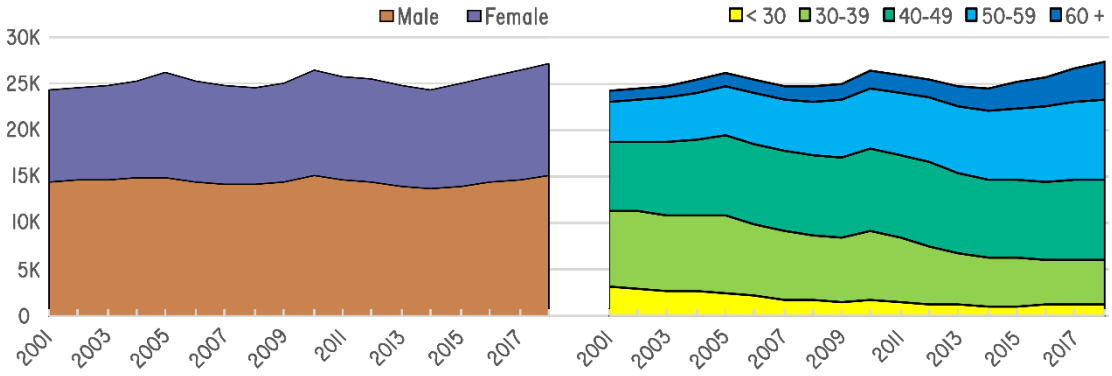



***Figure 4.1.2.*** *Academic staff at public higher education, by gender (left) and age group (right), from 2001 to 2018 (DGEEC, 2020a).*

A partial justification for the aging of academic staff may lie in the increasing barriers to entry. The 2009 revision of the Teaching Career Statutes† set the doctorate as the minimal qualification to future entrants. Academic staff under thirty were reduced from 12.5% in 2001 to 4.2% in 2018. In the same period, over nine thousand docents obtained a doctorate: only 31.1% of academic staff had a doctoral degree in 2001, 45.9% did so by 2008, and 61.9% had their doctorate by 2018. Back in 2001, 29.8% of academic staff only had a master's diploma, and 39.1% had an even lower academic qualification (licentiate or bachelor's degree); as of 2018, only 19.9% have no qualification beyond a master's degree, though 18.2% are still undergraduates (DGEEC, 2020a). A stark difference exists between public universities and polytechnic institutes: 73.4% of those teaching at the former in 2018 held a doctorate, yet only 44.2% of those teaching at the latter did.

* The DGEEC has published data prior to 2001, yet only concerning a subset of institutions, hence we withheld presenting figures prior to that break in the series.

† The University Teaching Career Statute (*Estatuto da Carreira Docente Universitária*), Decree 205/2009, and the Polytechnic Teaching Staff Career Statute (*Estatuto da Carreira do Pessoal Docente do Ensino Superior Politécnico*), Decree 207/2009.

### 1.2.3 STUDENTS AND DIPLOMAS IN RECENT DECADES

According to the Portuguese Censuses (INE, 2011), only 0.9% of the resident population had a higher education diploma in 1970, and men were more likely to have graduated (1.4% versus 0.5% of women). By 1981, 2.4% of residents had graduated (3.3% of men versus 1.6% of women), and by 1991, 4.0% did (4.5% of men versus 3.6% of women). The portion of women with a diploma (9.3%) overtook men's (7.4%) in the 2001 Census. As more of the resident population had a higher education diploma—8.4% in 2001, 14.8% in 2011—women's lead widened (16.9% versus 12.4% of men in 2011). As we have seen, women's higher qualifications are not yet reflected in their employment.

A look into a younger segment of the population—30 to 34-year-old residents—brings this picture into starker focus. In 1999, 11.1% had a higher education diploma (13.4% of women versus 7.9% of men), and a decade later, in 2009, that number had risen to 21.3% (24.9% of women and 17.6% of men). By 2019, 36.2% of residents in that age range (42.5% of women versus only 29.7% of men) had obtained a diploma from a higher education institution (PORDATA, 2020e).

Hence, most present-day Portuguese young adults still lack an educational qualification beyond high school, which suggests there is still some space for additional growth, despite the dramatic increase in residents' qualifications since the 1990s. Figure 4.1.3 clearly displays the consistent growth in the number of students enrolled at public universities and polytechnics between the early nineties and the early aughts:

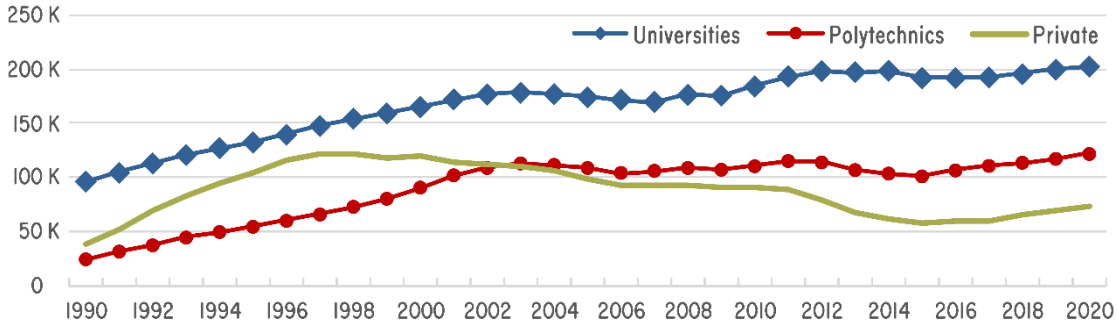



***Figure 4.1.3.*** *Students attending public universities, public polytechnic institutes, and private higher education institutions, from 1989/90 to 2019/20 (PORDATA, 2020a).*

A significant drop in the birth rate during the 1980s* may have dampened growth after the turn of the millennium, and economic crises brought further oscillations. Still, as of the 2019/20 academic year, the number of students enrolled in all of Portuguese higher education has been rising for five straight years, up to 397,000 students from 350,000 in 2014/15 (PORDATA, 2020a).

The decline of private institutions is worth noting, however. From their heyday in the late 1990s (with over 120,000 students), private institutions, which have significantly higher tuition fees, were affected by multiple factors: the territorial consolidation and growing diversity of the public higher education network, the unfavorable demographic evolution, the economic crises (e.g., the 2011 sovereign debt crisis), in addition to a crisis

* From 16.2 births per thousand residents in 1980 to 11.7 in 1990—the same exact figure as in 2000 (INE, 2020).

of confidence in the quality of private institutions—since the government watchdog was founded in 2009, the Higher Education Assessment and Accreditation Agency (*Agência de Avaliação e Acreditação do Ensino Superior*, A3ES) has ordered 21 private institutions closed (A3ES, 2020). Still, tighter regulation and a favorable economic outlook granted the private higher education system continuous growth for the past five years, from over 57,000 students in 2014/15 to a little over 73,000 in 2019/20 (PORDATA, 2020a).*

⁂

Allowing for the above economic and demographic factors, Figure 4.1.4 shows that undergraduate courses' attendance has been consistent since the late 1990s. Though far from its 2002/03 peak, when almost 368,000 students were attending either bachelor's or licentiate courses (as they existed before the Bologna Process), there were over 289,000 attending licentiates or integrated master's in 2019/20. Moreover, their number has been steadily rising for the past half-decade (PORDATA, 2020b).

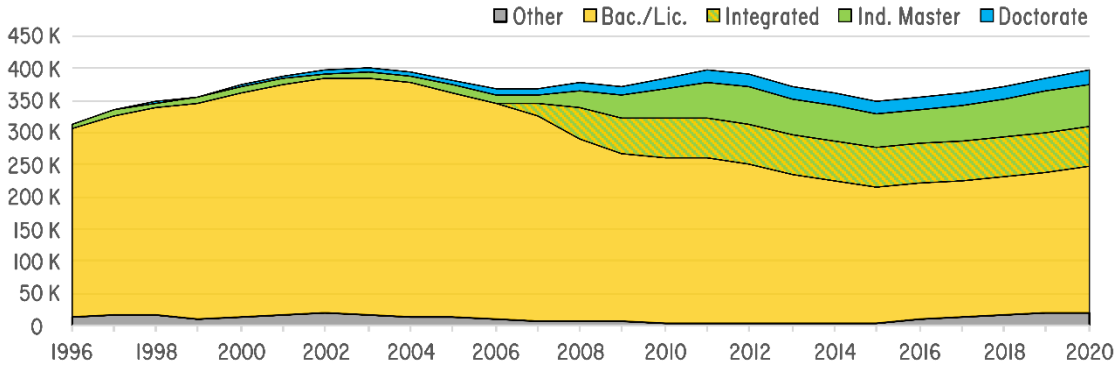



***Figure 4.1.4.*** *Students attending higher education (in both public and private systems), by degree level, from 1995/96 to 2019/20 (PORDATA, 2020b).*

There was an immense growth in the demand for secondary qualifications after the implementation of the Bologna Process. Though little over 11,000 students attended an independent master's course in 2006/07, by 2010/11 over 55,000 were attending such a program. Growth stabilized afterward, and as of 2019/20, almost 65,000 students were attending an independent master's course. Demand for tertiary qualification anticipated the Bologna Process: there were slightly less than 3,000 doctoral students in 1999/2000, almost 7,000 in 2004/05, and over 16,000 in 2009/10. As of 2019/20, there were nearly 22,000 doctoral students, suggesting moderate growth during the previous decade.

Included in Figures 4.1.4 and 4.1.5 as part of 'other' qualifications alongside other specializations and complements, we note the recent emergence of technical-vocational courses (CTESP). These occupy the qualification space formerly taken by two or three-semester technical specialization courses (*Cursos de Especialização Tecnológica*, CET) created in 1999 and which could be offered by higher education institutions up till 2016 (DGES, 2020).

* Yet, whether most private institutions will have a sustainable business after the COVID-19 pandemic remains unclear at the time of writing.

The number of students obtaining their diplomas in each year maps to the number of students enrolled in the courses, as shown in Figure 4.1.5. We were able to obtain such figures since 1991, which again evinces the sustained growth of Portuguese higher education during the 1990s (PORDATA, 2020c):

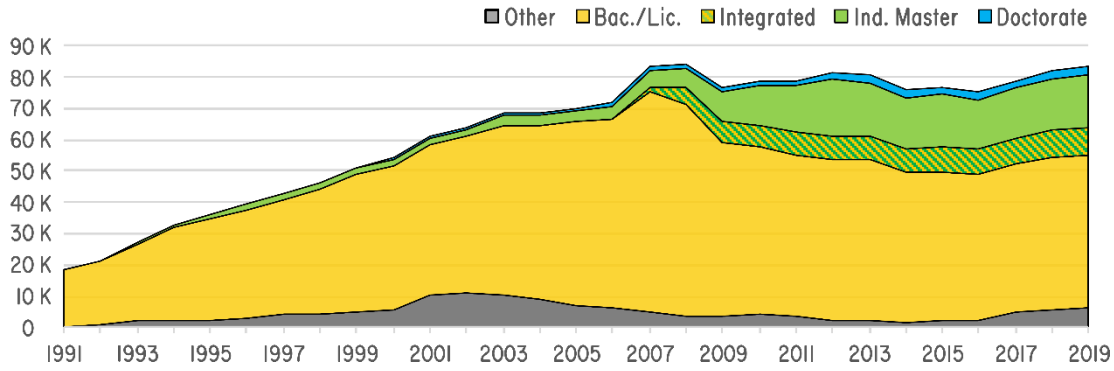



***Figure 4.1.5.*** *Students obtaining a diploma at a higher education institution (in both public and private systems), by degree level, from 1990/91 to 2018/19 (PORDATA, 2020c).*

We note typical integrated master's courses last four or five years, which explains the lower rate of graduations compared to that of independent master's courses.

## 1.3 *Higher arts, design, and architecture education*

As to begin sketching our panorama, which we will describe in detail in §2, we employed the public database offered by the Higher Education General Directorate (*Direcção Geral do Ensino Superior,* DGES) at www.dges.gov.pt/en/pesquisa_cursos_instituicoes to look for the undergraduate courses identified as granting an education in Arts, Design, or Architecture within the schema of the National Classification of Education and Training Fields (*Classificação Nacional das Áreas de Educação e Formação*—CNAEF).

The CNAEF taxonomy is directly inspired by the ISCED—International Standard Classification of Education (UNESCO, 1997, 2013) and was made into the Portuguese legal framework by Ordinance (*Portaria*) 256/2005. CNAEF is a three-level decimal classification of education and training fields, with eight groups at the topmost level (Portugal, 2005). In the coming pages, we shall examine some noteworthy aspects of the schema. The exclusion of the field of Architecture from the Arts and Design group required consideration of whether to include it in our review; we argue for its inclusion out of a shared history and institutional practices in §1.3.2.

### 1.3.1 ARTS AND DESIGN WITHIN THE CNAEF CATEGORIZATION

The CNAEF places arts and design training within the Arts (CNAEF 21) field of study, which is, in turn, part of Group 2—Arts and Humanities. Though attributing different codes, Portuguese legislation reproduces the ISCED almost to the letter in specifying CNAEF 21 as comprising seven fields of training related to five distinct areas:

- CNAEF 211, FINE ARTS, concerns *"creative visual expression, as well as theory, history, techniques, execution, and production of artworks, pictures and sculptures in particular"* (Portugal, 2005). Equivalent to ISCED 0213.
- CNAEF 212, PERFORMING ARTS, concerns *"principles and techniques related to the execution and interpretation in music, speech, movement, mime, role-playing, improvisation, and scenic art"* (Portugal, 2005). Equivalent to ISCED 0215.
- CNAEF 213, AUDIOVISUAL AND MEDIA PRODUCTION, concerns *"the necessary techniques for producing books, newspapers, radio and television programs, movies, videos, musical recordings, and graphic reproduction."* The legislator further adds that this area concerns *"color reproduction, photography, computer graphics, and the association of images, text, and illustrations"* (Portugal, 2005). As we will see, several Communication Design courses will be classified thusly. Equivalent to ISCED 0211.
- CNAEF 214, DESIGN, concerns *"the use of lines, forms, textiles or other materials, to the production of clothing, industrial products, and items and other elements for interior décor"* (Portugal, 2005). The legislator lists specific studies included by this classification: Interior Architecture, Scenography, Interior Design, Fashion Design, Industrial Design, Window Dressing. Equivalent to ISCED 0212, which explicitly clarifies that *"publishing Design and graphic Design are excluded"* (UNESCO, 2013).
- CNAEF 215, HANDICRAFTS, "*is to be understood in opposition to industrial production."* The legislator explains the classification concerns *"certain manual crafts, such as jewelry, ceramics, weaving, wooden sculpture, etc."* (Portugal, 2005). Equivalent to ISCED 0214.

Training or qualifications simultaneously related to three or more categories within CNAEF 21 and *"in which no field of training is clearly dominant"* are classified as CNAEF 210, ARTS (TRANSVERSAL). Specific arts studies not classifiable in any other area are assigned the code CNAEF 219 (Portugal, 2005).

⁂

We note no central authority oversees the CNAEF categorization of Portuguese higher education educational offers. Institutions themselves report courses' fields of training when filing for accreditation with the A3ES, a process to be repeated every few years. Thus, individual courses' CNAEF categories depend on institutions' interpretation of the norm when filing for the courses' accreditation. Using the search function at the A3ES website—www.a3es.pt/en/accreditation-and-audit—one may find a few disparities. For instance, of four Communication Design licentiates, three were classified as Audiovisual and Media Production courses (CNAEF 213), but the University of Porto's was filed* as a Design course (CNAEF 214). Several courses' A3ES filings also list a second category to express interdisciplinarity; for instance, the Lisbon Polytechnic Institute's Cinema licentiate† is categorized both as a Fine Arts and as an Audiovisual and Media course.

* According to the A3ES accreditation filing ACEF/1415/0412722.

† According to the A3ES accreditation filing ACEF/1415/0403517.

### 1.3.2 THE SPECIFIC CASE OF ARCHITECTURE

Architecture training and qualifications are notably left out from the Arts field of study in the CNAEF schema. Instead, Architecture and Urbanism (CNAEF 581) and its parent field of study, Architecture and Construction (CNAEF 58), are part of the Engineering, Industries, and Construction group (Portugal, 2005). The national classification follows the lead of the International Standard, in which Architecture and Town Planning (ISCED 0731) are part of ISCED qualification group 07, also dedicated to Engineering, Manufacturing, and Construction. However, other common training and qualification schemas include Architecture among the Arts: the OECD's (2007) Field of Science classification* places Architectural Design in the Arts (6.4) area; likewise, UNESCO's (1988) Nomenclature for Fields of Science and Technology includes Architecture (code 6201) in the Sciences of Arts and Letters area.

Exclusion of Architecture courses from our review and subsequent research could be justified by adherence to the CNAEF (and ISCED) classification of Arts training and qualifications. However, we cannot ignore the long standing of Architecture within a historical understanding of The Arts (Britannica, 2018). We also believe it is relevant that the teaching of Architecture in Portugal was historically linked to Fine Arts, from the foundation of both the Lisbon and Porto Academies in 1836 to the creation, in 1979, of Faculties of Architecture separate from the Fine Arts Schools† that had succeeded the Academies (Calado & Ferrão, 2013; Santos, 2011).

Further complicating matters, the Faculty of Architecture of the University of Lisbon (FAUL) has been offering undergraduate Design and Fashion Design courses (both categorized with CNAEF 214) since the mid-1990s.‡ At the University of Coimbra, the Department of Architecture is indeed a part of the Science and Technology Faculty (FCTUC), yet that Department co-administers a Design & Multimedia course (doubly categorized§ as CNAEF 213 and 481, computer science) alongside the Department of Computer Engineering, a fact that in itself suggests the contribution of the Department of Architecture is to bring aesthetic and artistic qualifications to the table.

Contemporary institutional practices place the teaching of Architecture in a gray area concerning its historical relationship to Fine Arts and a working relationship with Design areas. To speak nothing of the products of Design disciplines, seen by the taxonomists as part of the Arts, we witness how often works of contemporary architecture qualify for Arthur Danto's *"artworld"* (1964) and meet Nelson Goodman's *"symptoms of the aesthetic"* (as cited in Giovannelli, 2017), hence placing architectural productions within conventionalist definitions of art (Adajian, 2018). We valued the art-historical and the institutional relations over the technocratic schemas. We thus elected to include Architecture undergraduate courses in our review.

---

* Employed by the Portuguese Science and Technology Foundation (FCT).

† In the early 1990s the Fine Arts Schools—ESBAL in Lisbon and ESBAP in Porto—were also integrated into the cities' Universities as Faculties of Fine Arts.

‡ Created, respectively, by Deliberations (*Deliberação*) 19/UTL/92 and 14/UTL/92. Categories according to A3ES filings ACEF/1415/0410377 and ACEF/1415/0410382, respectively.

§ According to the A3ES accreditation filing ACEF/1415/0409137.

### 1.3.3 FACTS AND FIGURES

We had difficulties obtaining segmented data to accurately portray the recent evolution of public Arts, Design, and Architecture education. We reviewed public statistics and reports that, nonetheless, will allow us to sketch a good enough picture of contemporary public higher education in those fields.

According to a recent report by the General Directorate of Education and Science Statistics (DGEEC), the portion of students attending public Arts and Humanities (CNAEF group 2) courses of all levels has remained consistent at around ten percent since 1995/96: from little over 20,000 students (10.1% of all students in public higher education) in that year, to over 29,600 (9.6%) in 2010/11, to over 35,200 (10.9%) in 2019/20 (DGEEC, 2020b).

As the General Directorate's data does not discriminate further between Arts and Humanities, neither between different degrees, we looked into the analyses published by the Higher Education Assessment and Accreditation Agency (A3ES) which, albeit not as up to date, offer a sharper image of the state of higher education within specific fields of study. According to the field analysis for the 2010/11 academic year, there were 229 public Arts (CNAEF 21) courses of all levels attended by about 16,400 students (5.3% of all attending public higher education); Arts students were a slight majority (55.4%) in the Arts and Humanities group (Fonseca & Encarnação, 2013). Of Arts students, nearly 13,500 (82.3%) were enrolled in the 120 licentiates on offer (45 at universities and 75 at polytechnic institutes). According to the authors, those licentiates made nearly 3,900 vacancies available for first-time candidates in 2011. We may also share the authors' figures about private higher education: 68 private Arts licentiates were offered in the same year, attended by little under 5,100 students.

In addition to Fonseca & Encarnação's (2013) systematic analysis of all fields of study, the A3ES also published several detailed analyses concerning specific CNAEF fields of training, which may include more up-to-date figures:

- PERFORMING ARTS (CNAEF 212), 2015/16: Little over 3,200 students were enrolled in 33 public licentiates, which made almost 1,000 vacancies available for first-time applicants in 2016 (Friães, 2017a).
- AUDIOVISUAL AND MEDIA PRODUCTION (CNAEF 213), 2010/11: Over 4,750 students were enrolled in 35 public licentiates, which opened over 1,300 vacancies for first-time applicants in 2011 (Fonseca & Encarnação, 2013). As we have seen, this training field comprises many Communication and Visual Design courses.
- DESIGN (CNAEF 214), 2015/16: Almost 2,800 students attended the 23 public licentiates, which opened 770 vacancies for first-time applicants in 2016 (Friães, 2017b).
- ARCHITECTURE (CNAEF 581), 2013/14: There were just slightly over 5,000 students attending eight public integrated master's courses, for which there were almost 800 vacancies for first-time applicants in 2014 (Amaral, 2016).

To date, no detailed analysis of Fine Arts (CNAEF 211) courses was published. According to the A3ES' 2011 systematic analysis, over 3,750 students were enrolled in 70 Fine Arts courses of all levels, though no separate figures were given regarding undergraduate courses (Fonseca & Encarnação, 2013). In the same year, over 400 students attended eleven Handicrafts (CNAEF 215) courses, and just 12 students attended two transversal (CNAEF 210) Arts courses.

Though relying on figures dating back to 2014 or 2011, we may estimate that around 18,500 students attended close to 128 Arts, Design, or Architecture undergraduate courses (Amaral, 2016; Fonseca & Encarnação, 2013).

⁂

The Accreditation Agency recently published a characterization of Portuguese academic staff, albeit based on a census conducted in 2011 (Pinheiro, Machado, & Sarrico, 2018). According to the authors, there were 1,096 docents employed in the public higher education system whose highest qualification was classified within Arts (CNAEF 21) and 906 docents whose highest degree was classified within Architecture and Construction (CNAEF 58). While 392 (35.8%) of the former were women, only 247 (27.3%) of the latter were so. In comparison to the whole of the public system, in which 43.0% of teaching positions were occupied by women as of 2011 (DGEEC, 2020a), we find a greater gender imbalance among the academic staff qualified in Arts, Design, or Architecture.

Among the docents with an Arts degree (CNAEF 21) as their highest qualification, 179 (41.2%) of the 434 teaching at a public university had completed a doctorate as of 2011; however, only 80 (12.1%) of the 662 teaching at a public polytechnic institute had done so (Pinheiro et al., 2018). According to the same analysis, 127 docents (29.3%) teaching at universities and 257 docents (38.9%) at polytechnics held a master's degree; the remaining docents held no higher qualification than an Arts licentiate—29.5% of docents at universities and 49.0% of docents at polytechnic institutes.

Among the 906 docents teaching at public higher education institutions with an Architecture and Construction degree (CNAEF 58) as their highest qualification, 424 professors (46.8%) had completed their doctorate as of 2011, 237 (26.2%) had a master's degree, and 245 (27.0%) had no higher qualification than a licentiate (Pinheiro et al., 2018). Only 182 docents qualified in Architecture and Construction were teaching at polytechnic institutes, whereas 724 were employed by public universities. Of the former, only 31 (17.0%) had completed a doctorate.

These figures regarding the qualification of academic staff dating from 2011 ought to be construed as a contribution to a sketch of the present-day picture. As we have seen in §1.2.2, the portion of docents holding a doctorate in the entire public higher education system has risen from 45.9% in 2008 to 61.9% in 2018, driven by the new requirements of the Teaching Career Statutes approved in 2009.

We began our systematic review of Arts, Design, and Architecture public undergraduate courses (which we henceforth abbreviate as *arts and design*, unless noted) that include an education in computer programming as part of their curricula by conducting searches in the database available at the Higher Education General Directorate's (DGES) website—www.dges.gov.pt/en/pesquisa_cursos_instituicoes. We looked for courses identified as granting an education in Arts (CNAEF 21) fields—i.e., Fine Arts, Performing Arts, Audiovisual and Media Production, Design, and Handicrafts—or in Architecture (CNAEF 581), according to their host institutions' categorization under the schema of the National Classification of Education and Training Fields, as described in §1.3.

The DGES' website search interface does not allow selection of specific training fields beyond the general group categorizations (i.e., Arts and Humanities—CNAEF 2), and though it gives the specific three-digit code in its search results, it only presents one such code in the case of interdisciplinary courses in the same group (e.g., Art History degrees that may be categorized as CNAEF 225—History and Archeology and 211—Fine Arts). As to decide as to their potential exclusion, we conducted cursory examinations of the curricula of Humanities courses by following the links to their host institutions' webpages provided in the DGES' search results.

We completed a list of 101 licentiates and 8 integrated master's* offered at 23 public universities and polytechnic institutes in 2017/2018 that met our criteria of granting practical qualifications in arts and design, after excluding the interdisciplinary courses determined to lack a practical curriculum (e.g., Artistic Studies, Art History, or Cultural Heritage). We then examined the curricula of the courses in our provisional list, noting the curricular units that could potentially relate to computer programming. Apart from the explicit and obvious (units named after 'Programming' or a specific programming language), we scoured the curricula twice for terms potentially related to computational creativity and programming practice† and made a list of all matching curricular units, opting for their inclusion when in doubt.

In the next step, we conducted a cursory review of the curricular unit plans (CUP) available at the host institutions' websites to confirm computer programming was mentioned in the syllabus or the learning objectives sections. We also reviewed the bibliographies: in several cases, in which the rest of the document was incomplete or ambiguous, the recommendation of a programming manual was considered compelling evidence. We thus confirmed that 74 curricular units were related to teaching computer programming, though we were unable to obtain any public documentation for 69 others. Therefore, we included 143 curricular units in 50 public arts and design courses in our provisional list, though nearly half lacked thorough confirmation.

Nonetheless, the preliminary list was the basis upon which we planned our preliminary surveys of students and professors, presented in chapter **6**. In establishing

* All eight integrated master's courses were in Architecture, though none of the licentiates.

† This was an emergent judgment: e.g., 'Application' or 'Games', 'Computing', 'Digital', 'Interaction', 'Multimedia', 'Web' in combination with 'Development', 'Laboratory', 'Project', 'Studio', 'Technologies', or terms related to particular techniques such as 'Generative' or 'Procedural'. We also attended to course-specific terminology, as 'Electroacoustic Music'.

the contacts with the institutions to conduct the surveys, we also requested course coordinators and academic administrators access to the curricular unit plans we had been unable to locate. We then revised our provisional list of relevant curricular units at the beginning of the 2018/19 academic year, at which point we obtained several of the CUPs previously missing. In the cases we obtained no response to our request from the host institutions, we also investigated their public filings with the Higher Education Assessment and Accreditation Agency (A3ES), which yielded some missing unit plans, albeit outdated. Hence, an examination of the CUPs still outstanding led us to exclude several units from further review after concluding they lacked any mention of computer programming; this was common among the curricular units related to web design, in which the syllabus did not progress beyond presentation and formatting.* We remained unable to analyze the documentation for nine curricular units, which we nevertheless kept in our review given their unambiguous naming (e.g., 'Programming Languages').

Our revised list comprises 130 curricular units relating to computer programming, which were offered within 42 public arts and design undergraduate courses in 2018/19. Of the 121 documents we did analyze, 19 were undated, yet newly obtained from the respective institutions' websites in late 2019; another 9 were obtained from A3ES filings and were dated between 2015 and 2017. We found a great variation in the detail and completeness of each CUP: 16 plans did not identify the professor responsible for the unit or include an author, 5 plans did not include the section about their unit's learning objectives, 23 did not describe the teaching methodology, and 17 did not define the evaluation methodology; 10 documents omitted a bibliography.

⁂

In subsection §2.1, we will present and describe the arts and design courses we identified as including computer programming education. In §2.2, we will present a systematic cross-examination of the curricular units we identified, following CUPs standardized sections and categories.

## 2.1 *The courses*

Thus, our panorama comprises 130 curricular units, 20 of which elective, that were a part of the curricula of 42 courses offered at 20 separate universities and polytechnic institutes. Six of the licentiates did not include programming as part of their mandatory curricula but offered at least one elective unit about the subject. In Table 4.2.1 (next page) we summarize the forty licentiate and two integrated master's courses which we determined to include at least one curricular unit about computer programming and for which we were able to obtain the respective plan.

* Some plans mislabeled HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets) as programming languages, though these are text and page formatting and layout standards that offer no way to express an algorithm.

***Table 4.2.1.*** *Summary of Portuguese public arts and design undergraduate courses with curricular units about computer programming, updated for 2018/19.*

| *Institution* | *Course* [a] | CNAEF | *Units* [a] *(+ electives)* | *ECTS* [a] *(+elective)* | *Total students* | *First-year students* [b] |
|---|---|---|---|---|---|---|
| *Algarve University* | Communication Design | 213 | 2 | 8.0 | 130 | 40 |
| | *Visual Arts* | *211 213* | *(+1)* | *(+5.0)* | *71* | *25* |
| *Aveiro University* | Design | 214 | 1 | 4.0 | 179 | 64 |
| | New Technologies of Communication | 213 | 4 | 42.0 | 254 | 77 |
| | Product Design & Technology | 214 520 | 1 | 6.0 | 119 | 37 |
| *Beira Interior University* | Multimedia Design | 214 | 1 | 8.0 | 167 | 54 |
| *Coimbra University* | Design & Multimedia | 213 481 | 5 | 42.0 | 227 | 75 |
| *Évora University* | *Fine Arts & Multimedia* | *211 213* | *(+5)* | *(+24.0)* | *124* | *34* |
| | *Design* | *214 213* | *(+3)* | *(+10.0)* | *127* | *40* |
| | *Music* | *212* | *(+1)* | *(+3.0)* | *174* | *48* |
| *Lisbon University* | Architecture *(MA—5 years)* Architecture Faculty—FAUL | 581 | 1 | 3.5 | 991 | 190 |
| | Architecture *(MA—5 years)* Technical Institute—IST | 581 | 1 | 6.0 | 372 | 56 |
| | Multimedia Arts | 213 | 2 | 12.0 | 209 | 70 |
| | *Communication Design* | *213* | *(+1)* | *(+6.0)* | *216* | *72* |
| *Madeira University* | Design | 214 | 1 | 8.0 | 82 | 29 |
| *Minho University* | Music | 212 | 1 | 5.0 | 165 | 52 |
| | Product Design | 214 | 1 *(+1)* | 5.0 *(+6.0)* | 119 | 40 |
| *Porto University* | Communication Design *(4 years)* | 214 | 1 *(+1)* | 6.0 *(+4.5)* | 220 | 55 |
| | Fine Arts *(4 years)* | 211 | 1 | 4.5 | 389 | 102 |
| *Trás-os-Montes Univ.* | Multimedia & Communication | 213 | 7 | - | 189 | 60 |
| *Bragança Polytechnic* | Communications Technologies [c] | 213 | 3 | 18.0 | 32 [c] | *N.A.* [c] |
| | Digital Games Design | 213 | 5 | 30.0 | 139 | 49 |
| | Multimedia | 213 | 6 | 36.0 | 133 | 42 |
| *Castelo Branco Poly.* | Musical Electronics & Production | 212 | 7 [d] | 26.5 [d] | 33 | 12 |

| *Institution* | *Course* [a] | CNAEF | *Units* [a] *(+ electives)* | *ECTS* [a] *(+elective)* | *Total students* | *First-year students* [b] |
|---|---|---|---|---|---|---|
| | Communication & Audiovisual Design | 213 | 1 | 6.0 | 110 | 57 |
| *Coimbra Polytechnic* | Multimedia Design & Communication | 213<br>481 | 6 | - | 136 | 45 |
| *Guarda Polytechnic* | Multimedia Communication | 213 | 6 [d] | 33.0 [d] | 161 | 53 |
| *Leiria Polytechnic* | Graphic & Multimedia Design | 213 | 2 | 12.0 | 333 | 96 |
| | Digital Games & Multimedia | 213<br>481 | 4<br>*(+4)* | 24.0<br>*(+24.0)* | 184 | 57 |
| *Lisbon Polytechnic* | Audiovisual & Multimedia | 213 | 1<br>*(+2)* | - | 319 | 104 |
| | Music Composition | 212 | 2 | 6.0 | 64 | 16 |
| | Music Technologies | 212 | 1 | 3.0 | 63 | 15 |
| *Portalegre Polytechnic* | Communication Design | 211<br>214 | 2 | 12.0 | 58 | 17 |
| | *Multimedia Design & Animation* | *214* | *(+1)* | *(+6.0)* | *58* | *14* |
| *Porto Polytechnic* | Multimedia | 213 | 5 | 35.0 | 81 | 32 |
| | Music Composition | 212 | 4 | 14.5 | *21* [e] | *6* [f] |
| | Music Production & Technologies | 212 | 2 | 9.0 | *52* [e] | *12* [f] |
| | Theater—Light & Sound | 212 | 1 | 5.0 | - | *6* [f] |
| | Web Technologies & Information Systems | 214<br>480 | 9 | 56.0 | 104 | 35 |
| *Tomar Polytechnic* | Graphic Arts Design & Technology | 213 | 1 | 5.0 | 87 | 28 |
| *Viseu Polytechnic* | Multimedia Design & Technologies | 213 | 6 | 33.5 | 151 | 53 |
| | Fine Arts & Multimedia | 210<br>481 | 6 | 30.0 | 114 | 36 |

*Notes:*
[a] Courses that only offer programming in elective units shown in italics. Units and ECTS credits shown in regular type are mandatory.
[b] Average first-year enrollment from 2017/18 to 2018/19 (DGEEC, 2019).
[c] Course had been discontinued and was not accepting new applications.
[d] Some curricular unit plans were inaccessible, thus we estimated from their names whether the units concerned programming.
[e, f] Student data was absent from the DGEEC database (2019) and was obtained from the Porto Polytechnic Institute's website at portal.ipp.pt.
[f] Estimated from the number of places available for first-year enrollment.

We present these courses' territorial distribution in Figure 4.2.1, overlaid on the map of the Portuguese public higher education network presented in Figure 4.1.1. We did not identify courses in the Azores archipelago and in the Viana do Castelo (far north), Setúbal, and Beja districts (in the south).

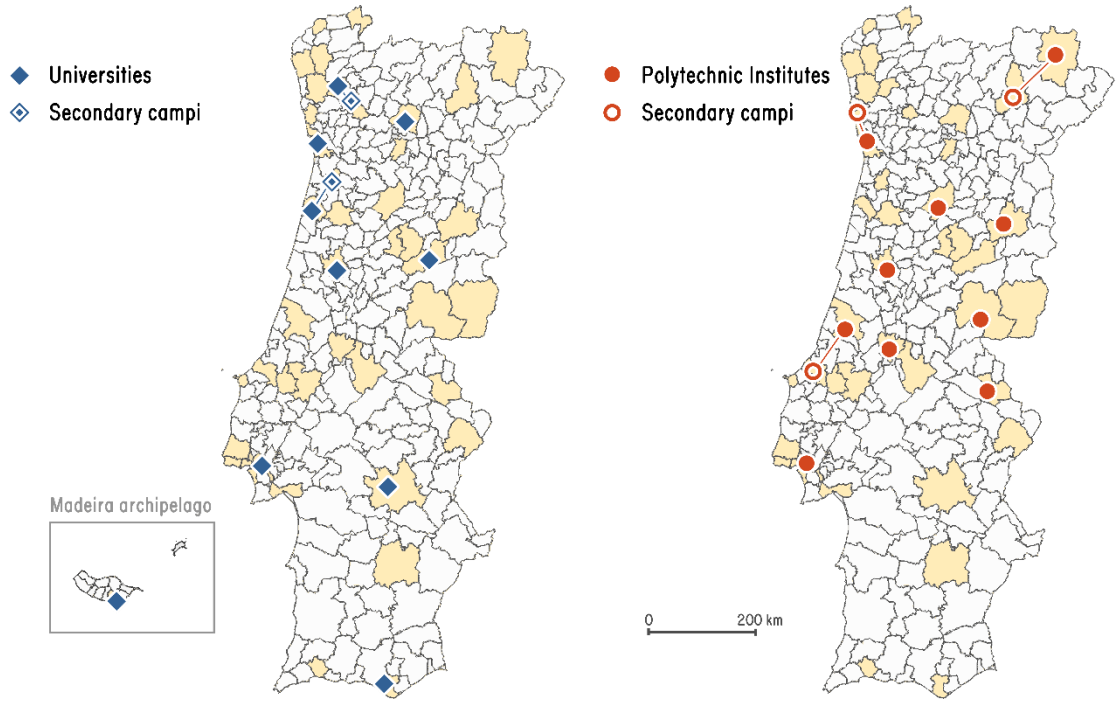



***Figure 4.2.1.*** *Territorial distribution of the public arts and design undergraduate courses that include an education in computer programming, 2018/19. Some courses were offered in faculties and schools away from the host institutions' main campi.*

We looked at the courses' CNAEF categorization, according to their A3ES filings. Apart from the two integrated master's in Architecture, four licentiates were categorized as Fine Arts (CNAEF 211), eight as Performing Arts (CNAEF 212), 21 as Audiovisual and Media Production (CNAEF 213), and ten as Design courses (CNAEF 214); one licentiate was categorized as a transversal Arts course (CNAEF 210).

Ten licentiates were interdisciplinary and categorized under two CNAEF fields of training. Four courses shared two distinct classifications within Arts (CNAEF 21), and six were categorized within Arts and a separate field of study: one such course was also categorized under transversal Engineering (CNAEF 520), another one was classified under transversal Informatics (CNAEF 480), and four were categorized under Computer Science (CNAEF 481). Though these interdisciplinary courses bridging Informatics and the Arts assigned considerable ECTS credits to units about computer programming, we note that five licentiates solely categorized under Audiovisual and Media Production also assigned over 30 ECTS credits to units about computer programming.

### 2.1.1 THE STUDENTS

According to the data published by the General Directorate of Education and Science Statistics relating to the 2018/2019 academic year, there were almost 6,900 students, 54.5% of which female, enrolled in the 39 courses for which we found figures (DGEEC, 2019). Though the data lacked information about three courses at the Porto Polytechnic Institute, we estimated their students from the yearly report available at the institution's website. We estimate a total of over 6,950 students were enrolled in these 42 courses; over 6,200 enrolled in the 36 courses requiring the mandatory completion of at least one unit related to computer programming.

An average of a little over 2,000 students began their first year at either of the 42 courses during the 2017/18 to 2018/19 period; of these, over 1,750 students enrolled in the 36 courses with mandatory curricular units related to programming. The DGEEC's data (2019) included figures relating to students' candidacies when applying for public higher education for the first time in 2019. On average, 55.8% of these students enrolled in their first-chosen course,[*] though there was significant variation: less than 17% had picked the Multimedia licentiate at the Bragança Polytechnic as their first choice, and only 29% had first applied for Graphic Arts Technology at the Tomar Polytechnic or Web Technologies & Information Systems at the Porto Polytechnic. While the two former examples may be explained by students being less likely to apply for institutions located in smaller inland cities, the latter may be explained by the established competition from other public Design or Multimedia courses in the same region (even in the same institution). Indeed, 79% of students who enrolled in Communication Design at the University of Porto and Design at the insular University of Madeira, did so at their first-chosen course.

### 2.1.2 A BRIEF CHRONOLOGY

We conducted a brief investigation of the 42 courses' past history. We faced frequent difficulties in ascertaining the courses' creation date or obtaining past curricular plans from their institutions' websites, which we find noteworthy. Therefore, we conducted searches on the Portuguese government gazette (*Diário da República*)—dre.pt—for the legal acts creating or modifying the courses.

The median initial year of activity of the courses was 2001/02. Five courses existed before their recognition as granting higher education degrees in 1983 and 1984: now at the University of Lisbon, Architecture[†] and Design; now at the University of Porto, Design and Fine Arts; and now at the University of Madeira, Design.[‡] Three other bachelor's degrees or licentiates were created in the 1980s: the two Music Composition courses now offered at the Lisbon and Porto Polytechnics, and the Graphic Arts course now offered at the Tomar Polytechnic Institute.[§] Eleven courses were created between 1990 to 1999, following that decade's dramatic expansion in enrollment (see §1.2), and nineteen others were created from 2000 to 2009, in the run-up to and the immediate aftermath of the implementation of the Bologna Process. Only four of the licentiates in our review were created after 2010: Digital Games at the Bragança (2010) and the Leiria (2014) Polytechnic Institutes, Product Design at the University of Minho (2012), and Web Technologies & Information Systems at the Porto Polytechnic Institute (2015).[**]

---

[*] The general application contest for Portuguese public undergraduate courses employs a ranked choice method. Performing Arts courses are not a part of the general contest: students are required to apply directly at each institution and to complete prerequisites (e.g., interviews or auditions).

[†] Meaning the course now offered at the Faculty of Architecture (FAUL), given the duplicate offering of Architecture courses at the University of Lisbon.

[‡] Their recognition was granted, respectively, by Ordinance (*Portaria*) 448/84, Government Decree (*Decreto Governamental*) 38/83, Law Decree (*Decreto-Lei*) 80/83 (both University of Porto courses), and Ordinance 707/84.

[§] Respectively created by Ordinances (*Portaria*) 765/86, 167/87, and 593/87; the latter course was offered by the Santarém Polytechnic before the Tomar Polytechnic was spun off.

[**] Resp. created by Dispatches (*Despacho*) 2887/2010, 7394/2014, 10269/2012, and 9221/2014.

As far as we could determine from our analysis of the discontinued curricular plans published in the government gazette, Communication Design at the University of the Algarve was the first of the 42 courses in our panorama to offer students an education in computer programming. According to Ordinance (*Portaria*) 897/92 (DR 214/92 I-B, Sept. 16th), that course's initial curriculum for the 1992/93 academic year included two Computer Programming and two Hypermedia Systems units. Though we were unable to review the 28-year-old CUPs, it appears the present-day curriculum assigns lesser weight to computer programming (two units weighing 8 ECTS credits). Nonetheless, though nineteen of the courses we identified had begun their activities by the turn of the millennium, only three courses may have included computer programming in their curricula at that point in time: the aforementioned Communication Design course, Graphic Arts & Technology at the Tomar Polytechnic (the revised 1998 curriculum added seven units named after Information Technology, Informatics, or Multimedia*), and Communication & Audiovisual Design at the Castelo Branco Polytechnic Institute (the initial 1999 curriculum included a Multimedia Programming Languages unit†).

Of the ten courses under review that began to offer an education in computer programming between 2000 and 2004, only three had been previously created: Design and New Communication Technologies, both at the University of Aveiro, and Musical Composition at the University of Évora.‡ It is telling the median initial offering of computer programming education for all 42 courses in our panorama was in the 2007/08 academic year, coinciding with the implementation of the Bologna Process. Of the eleven courses created between 2005 and 2009, ten included computer programming in their curricula from the beginning. During the same period, eight existing courses revised their curricula as to offer a programming education, all between 2006 and 2007: the Architecture integrated master's at the Technical Institute (*Instituto Superior Técnico*—IST) of the University of Lisbon,§ the Fine Arts and Communication Design licentiates at the University of Porto,** the Communication Design course at the Portalegre Polytechnic,†† the Music Composition courses at the Lisbon and the Porto Polytechnic Institutes, in addition to the Music Production and Theater—Light and Sound licentiates at the latter.‡‡

---

* Published in Ordinance 1021/2000. Though we were unable to review the CUPs, we believe it likely that some form of computer programming was taught, considering the practices of 1990s Multimedia development and the number of units involving computing.

† The initial curriculum for the course, under the prior Designation of Image Arts, was published in Ordinance 494/2000.

‡ Respectively created by Dispatches 29-R/96, 15-R/93, and 17688/98. Curricular units concerning programming were found in the revised curricula respectively published in Dispatch 22420/2001 (for both courses at the University of Aveiro) and Deliberation *(Deliberação)* 1174/2004.

§ Created by Deliberation 479/UTL/98, and then revised in Dispatch 1902/2007.

** Respectively revised in Dispatches 1051/2006 and 1041/2006; the former also created the Multimedia variant of the Fine Arts licentiate.

†† Created in 1998 from an existing Communications Engineering course which changed Designation, according to Ordinance 325/99. Yet, its curriculum eschewed computing topics during the first ten years' activity until the revision published in Ordinance 1198/2007.

‡‡ Respectively revised in Dispatches *(Despacho)* 8472/2008, 19158/2008, 19047/2008, and 19155/2008.

Seven of the 42 courses in our panorama revised their curricula to include units dedicated to computer programming from 2010 to the date of writing (in addition to the four aforementioned licentiates created in this late period): the integrated master's in Architecture at the University of Lisbon's Faculty of Architecture (FAUL) and the licentiates in Communication Design at the University of Lisbon's Faculty of Fine Arts and in Design at the University of Madeira,* as well as the licentiate in Visual Arts at the University of the Algarve, the two licentiates in Visual Arts & Multimedia and in Design at the University of Évora, and the licentiate in Graphic & Multimedia Design at the Leiria Polytechnic's Institute's School of Arts and Design.† Of this latter group, all but the Design course at the University of Évora (created in 2007) had been established before the implementation of the Bologna Process.

## 2.2 *The curricular units*

Table 4.2.2 (following pages) presents a list of the 130 curricular units we identified as related to computer programming within the 42 courses described in the previous subsection. We include the semester in which the unit is placed in the course plan or whether the unit is an elective option (marked with *Op.*); note there are free elective units not assigned to a particular semester. We also list how many hours are allocated to sessions during the term, and of what type: whether these are theoretical lectures (T), mixed theoretical-practical sessions (M), studio or laboratory practice (L), or individual orientation or tutoring (or).‡ We also list the main programming languages mentioned in each curricular unit's plan.

* Revised curricula respectively published in Dispatches 7513/2013, 6840/2019, and Notice (*Aviso*) 12444/2014.

† Respectively created by Deliberation 678/2004, Dispatches 3/SAC/96 and 9182/2008, and Ordinance 1310/95. Curricular units concerning programming were found in the revised curricula respectively published in Notices 14903/2017 and 8553/2018, and Dispatches 9459/2011 and 8608/2013.

‡ There is no overarching definition of each session type. Instead, each institution's internal regulations may set particular requirements—e.g., on the recommended number of students that may attend a theoretical session vis-à-vis a studio practice before requiring partition in multiple classes.

***Table 4.2.2.*** *Summary of units concerning computer programming in Portuguese public arts and design undergraduate education, updated for the academic year 2018/19.*

| *Course* | *Curricular unit* | *Sem.* [a] | *Hours* [b] | *Main tools* |
|---|---|---|---|---|
| ALGARVE UNIVERSITY | | | | |
| *Communication Design* | Interaction and Multimedia Design | 3rd | 15T 15M | Processing, Arduino |
| | Multimedia Techniques | 3rd | 15T 30M | Processing |
| *Visual Arts* | Experimental Multimedia Lab | *Op.* | 45 L | Arduino |
| AVEIRO UNIVERSITY | | | | |
| *Design* | Interaction Design | 4th | 15T 15M | JavaScript |
| *New Technologies of Communication* | Multimedia Lab II | 2nd | 30T 60L 15or | JavaScript |
| | Multimedia Lab III | 3rd | 30T 60L 15or | JavaScript |
| | Multimedia Lab IV | 4th | 30T 60L | PHP, SQL, Unity, C# |
| | Multimedia Lab V | 5th | 30T 60L 15or | Java, Android |
| *Product Design & Technology* | Information Systems | 1st | 15M 45L | Python |
| BEIRA INTERIOR UNIVERSITY | | | | |
| MULTIMEDIA DESIGN | Multimedia Production | 6th | 60M 15or | Processing |
| COIMBRA UNIVERSITY | | | | |
| *Design & Multimedia* | Introduction to Programming | 1st | 84 M | Processing |
| | Multimedia Programming | 2nd | 84 M | Processing |
| | Internet Technologies | 3rd | 28T 28L | JavaScript, PHP |
| | Computer Graphics | 6th | 28T 14M 28L | Processing, C |
| | Intro. to Artificial Intelligence | 6th | 28T 14M 28L | Processing, C |
| ÉVORA UNIVERSITY | | | | |
| *Fine Arts & Multimedia* | Tangible Multimedia I | *Op.* | 30M 15or | Python, PD, Arduino |
| | Tangible Multimedia II | *Op.* | 30M 15or | Python |
| | New Media Technologies II | *Op.* | 60 L | Processing |
| | New Media Technologies III | *Op.* | 60 L | Processing |
| | New Media Technologies IV | *Op.* | 60 L | *"Game engine"* |
| *Design* | Introduction to Interactive Systems | *Op.* | 30M 15or | Processing |
| | Techniques & Materials I | *Op.* | 30M 15or | JavaScript, PHP |
| | Techniques & Materials II | *Op.* | 30M 15or | ActionScript |
| *Music* | Electroacoustic Music IV | *Op.* | 30 M | Max |
| LISBON UNIVERSITY | | | | |
| *Architecture MA at FAUL* | Geometric & Generative Modelling | 7th | 45 M | Grasshopper |
| *Architecture MA at IST* | Programming & Computation | 5th | 15T 45L | Julia |
| *Multimedia Arts* | Multimedia Computing | 1st | 63 L | Processing, Unity, C# |
| | Multimedia Workshop | 5th | 63 L | Unity, C#, Arduino |
| *Communication Design* | Computer Graphics | *Op.* | 42 M | Processing |
| MADEIRA UNIVERSITY | | | | |
| *Design* | Interactive Media Design | 5th | 80 M | JavaScript |

| *Course* | *Curricular unit* | *Sem.* [a] | *Hours* [b] | *Main tools* |
|---|---|---|---|---|
| MINHO UNIVERSITY | | | | |
| *Music* | Musicology Informatics | 4th | 20T 15M 10L | Haskell |
| *Product Design* | Digital Form and Techniques | 4th | 20M 25L | Processing |
| | Data Analysis | *Op.* | 30 L | R, SPSS |
| PORTO UNIVERSITY | | | | |
| *Communication Design* | Sound & Image Lab | 4th | 48 M | Processing |
| | Web Design II | *Op.* | 48 M | JavaScript |
| *Fine Arts* | Multimedia Lab | 5th | 64 M | Processing, Max, Arduino |
| TRÁS-OS-MONTES UNIVERSITY | | | | |
| *Multimedia & Communication* | Programming & Algorithms Basics | 1st | 15T 30L | Portugol |
| | Introduction to Web Development | 1st | 30T 30L | JavaScript |
| | Multimedia Tools and Applications | 2nd | 30T 30L | Flash ActionScript |
| | Web Applications Development I | 3rd | 30T 30L | PHP |
| | Web Applications Development II | 4th | 30T 30L | PHP, JavaScript |
| | Intro. to Digital Game Development | 5th | 30T 30L | *"Game engine"* |
| | Multimedia Project | 5th | 60 M | PHP, JavaScript |
| BRAGANÇA POLYTECHNIC INSTITUTE | | | | |
| *Communications Technologies (discontinued)* | *Programming and Algorithms* | *2nd* | *15M 45L 20or* | *C* |
| | *Multimedia I* | *4th* | *15M 40L 20or* | *JavaScript* |
| | *Multimedia II* | *5th* | *15M 45L 20or* | *ActionScript* |
| *Digital Games Design* | Programming I | 1st | 15M 45L | C |
| | Programming II | 2nd | 15M 45L | Java |
| | Computer Graphics | 3rd | 15M 45L | Unity, C# |
| | Game Engines | 4th | 15M 45L | Unity, C# |
| | Artificial Intelligence | 5th | 15M 45L | - |
| *Multimedia* | Introduction to Programming | 1st | 60 M | Python |
| | Object-oriented Programming | 2nd | 60 M | C# |
| | Web Development I | 3rd | 60 M | JavaScript |
| | Web Development II | 4th | 60 M | JavaScript, PHP |
| | Interaction Technologies | 4th | 60 M | C++, Max, Arduino |
| | Applications Development | 5th | 60 M | ActionScript |
| CASTELO BRANCO POLYTECHNIC INSTITUTE | | | | |
| *Musical Electronics & Production* | Programming Languages I | 1st | 64M 16or | Processing |
| | Programming Languages II [c] | 2nd | 64M 16or | - |
| | Programming Languages III [c] | 3rd | 64M 16or | - |
| | Programming Languages IV [c] | 4th | 64M 16or | - |
| | Musical Programming I [c] | 4th | 32M 8or | - |
| | Musical Programming II [c] | 5th | 32M 8or | - |
| | Algorithmic Composition [c] | 6th | 32M 8or | - |
| *Communication & Audiovisual Design* | Multimedia Lab II | 5th | 60 M | JavaScript |

| *Course* | *Curricular unit* | *Sem.* [a] | *Hours* [b] | *Main tools* |
|---|---|---|---|---|
| COIMBRA POLYTECHNIC INSTITUTE | | | | |
| *Multimedia Design & Communication* | Programming and Algorithms I | 1st | 13T 39M | Scratch, Java |
| | Programming and Algorithms II | 2nd | 13T 39M | Java |
| | Web Programming I | 3rd | 13T 39M | JavaScript |
| | Web Programming II | 4th | 13T 39M | JavaScript |
| | Applications Development | 4th | 13T 39L | JavaScript |
| | Multimedia Lab II | 5th | 52 L | Processing, C++ |
| GUARDA POLYTECHNIC INSTITUTE | | | | |
| *Multimedia Communication* | Multimedia Programming | 2nd | 60M 7.5or | Processing |
| | Interaction Lab I [c] | 3rd | 60M 7.5or | - |
| | Interaction Lab II [c] | 4th | 60M 7.5or | - |
| | Web Development I | 4th | 60M 7.5or | JavaScript |
| | Web Development II | 5th | 60M 7.5or | JavaScript, PHP |
| | Development for Mobile Devices [c] | 5th | 60M 7.5or | - |
| LEIRA POLYTECHNIC INSTITUTE | | | | |
| *Graphic & Multimedia Design* | Project Lab I | 5th | 60 M | Processing, P5.js |
| | Project Lab II | 6th | 60 M | P5.js, Arduino |
| *Digital Games & Multimedia* | Introduction to Programming | 1st | 30M 45L | C# |
| | Game Programming | 2nd | 30M 60L | Unity, C# |
| | Web Development | 3rd | 45 L | JavaScript |
| | Game Engines I | 3rd | 60 L | Unreal Engine |
| | Game Engines II | *Op. 4th* | 60 L | C++, Unreal Engine |
| | Computer Graphics | *Op. 5th* | 60 L | C++, OpenGL |
| | Adv. Topics for Game Programming | *Op. 5th* | 60 L | C++ |
| | Artificial Intelligence for Games | *Op. 5th* | 60 L | Unreal Engine |
| LISBON POLYTECHNIC INSTITUTE | | | | |
| *Audiovisual & Multimedia* | Interactive Applications Lab | 5th | 15T 30L | JavaScript, PHP |
| | Augmented Reality | *Op. 6th* | 15T 30L | Unity, C# |
| | Real-Time Interaction | *Op. 6th* | 15T 30L | Supercollider, P5.js, C++, Arduino, PD, ... |
| *Music Composition* | Sound Synthesis | 3rd | 45 M | CSound |
| | Real-Time Electronics | 4th | 45 M | Max |
| *Music Technologies* | Computer Programming | 2nd | 22.5 T | Matlab/Octave |
| PORTALEGRE POLYTECHNIC INSTITUTE | | | | |
| *Communication Design* | Multimedia Production I | 5th | 30M 30L 20or | ActionScript |
| | Multimedia Production II | 6th | 30M 30L 20or | Processing |
| *Multimedia Design & Animation* | Computer Programming | *Op.* | 60M 20or | JavaScript |
| PORTO POLYTECHNIC | | | | |
| *Multimedia* | Programming I | 2nd | 60 M | Unity, C# |
| | Programming II | 3rd | 60 M | JavaScript, Node |

| *Course* | *Curricular unit* | *Sem.* [a] | *Hours* [b] | *Main tools* |
|---|---|---|---|---|
| | Programming III | 4th | 60 M | JavaScript |
| | Laboratory II | 4th | 60 M | Processing |
| | Laboratory III | 5th | 75 M | Processing, Max, Arduino |
| *Music Composition* | Musical Programming I | 2nd | 30 M | Max |
| | Sound Analysis and Synthesis | 3rd | 30 M | Max |
| | Musical Programming II | 4th | 30 M | Max, Processing |
| | Interactive Digital Systems | 6th | 30 M | Max, Processing |
| *Music Production & Technologies* | Applied Electronics III | 5th | 30 M | Arduino |
| | Multimedia Systems Programming | 6th | 50 M | Max |
| *Theater Light & Sound* | Introduction to Digital Lighting | 6th | 75 M | Processing |
| *Web Technologies & Information Systems* | Algorithms and Data Structures | 1st | 32M 48L | C# |
| | Web Applications and Technologies | 1st | 16M 48L | JavaScript |
| | Object-oriented Programming | 2nd | 32M 48L | JavaScript |
| | Web Programming I | 3rd | 32M 48L | JavaScript |
| | Web Programming II | 4th | 32M 48L | JavaScript |
| | Graphics Systems | 4th | 32M 32L | JavaScript, OpenGL |
| | Game Development | 5th | 32M 32L | Unity, C# |
| | Distributed Systems | 5th | 32M 48L | JavaScript, Arduino |
| | Programming for Mobile Devices | 6th | 32M 32L | Java, Swift |
| TOMAR POLYTECHNIC INSTITUTE | | | | |
| *Graphic Arts Design & Technology* | Multimedia Design | 4th | 60 M | JavaScript |
| VISEU POLYTECHNIC INSTITUTE | | | | |
| *Multimedia Design & Technologies* | Programming and Problem-solving | 1st | 20T 52M | C# |
| | Programming and Data Structures | 2nd | 13T 52L | C# |
| | Object-Oriented Programming | 3rd | 13T 39L | C# |
| | Internet Applications I | 3rd | 13T 52L | JavaScript |
| | Internet Applications II | 4th | 13T 52L | JavaScript, PHP |
| | Applications for Mobile Devices | 5th | 20T 52L | C#, Java |
| *Fine Arts & Multimedia* | Informatics | 1st | 30M 60L | SQL |
| | Multimedia II | 2nd | 30M 45L | Processing |
| | Web Studio | 3rd | 30M 45L | JavaScript, PHP |
| | Art & Multimedia Lab I | 4th | 45L | *"Scripting"* |
| | Art & Multimedia Lab II | 5th | 90L | Arduino |
| | Art & Multimedia Lab III | 6th | 45L | Processing |

*Notes:*

[a] Curricular semester. Elective units are labeled 'Op.'

[b] (T)heoretical lectures; (M)ixed theoretical-practical sessions; (L)aboratory practice sessions; (or)ientation or individual tutoring.

[c] We were unable to obtain and review the curricular unit plan. These units are presumed relevant on account of their names.

Of the 130 curricular units reviewed, 30 were first-year units (i.e., taking place in the 1st or 2nd semester), 46 were second-year units, and 40 were third-year units; all but one of the latter took place in the final year of the respective licentiates. In addition, there was a single fourth-year unit. We note 13 elective units were not specific to a particular year, as students had two or more opportunities to enroll throughout their course. According to our analysis of the curricular plans, 46 units can be regarded as introductory (i.e., the first programming unit in the respective course). Students enrolling in the remaining 84 units can be expected to have acquired a familiarity with computer programming. Of these, 27 units were preceded by one semester in which students were expected to have contact with programming; 21 units were preceded by two such semesters, 20 were preceded by three semesters, 13 were preceded by four semesters, and three units were preceded by five semesters.*

Some of the curricular unit plans (CUPs) we reviewed included a section about enrollment requirements or recommendations. Passing one or more preceding units was a strict requirement for enrollment in 18 units. Other CUPs advised, yet not required, prior competency in computer programming (10), in mathematics (6), in the English language (2), in specific media manipulation and editing software (4), and in general computer use (3).

✦ SESSION CHARACTERISTICS The curricular units we reviewed emphasized practice. Though 33 units included theoretical lectures in their schedules, only one such unit did not include additional sessions of other types. Mixed theoretical-practical sessions were part of 91 curricular units' schedules (exclusively so in 34 cases). Studio or laboratory practice sessions were part of 67 units' schedules (exclusively so in 17 cases). Always in complement to sessions of other types, 28 curricular units included tutorial orientation in their schedules. Though many CUPs omitted that information, Portuguese was listed as the main teaching and working language of 29 units; of these, 16 also mentioned that there would be support for English-speaking students. An additional 17 CUPs explicitly mentioned bilingual English and Portuguese lessons, and 7 other plans listed English as the main teaching language.

All but 16 CUPs identified the professor responsible for the curricular unit, as well as any assistants. We found that 80 units were taught by a single professor; in 21 units, the professor would be aided by one assistant, and in 6 units, the professor would be aided by two or more assistants. We identified 98 individual docents involved in the curricular units. In stark contrast with the courses' overall student make-up, only 18 of these docents were female; furthermore, female professors only had been attributed 23 out of the 142 total teaching positions related to computer programming.

✦ LEARNING OBJECTIVES We used the Atlas.ti 8 software to categorize the items enumerated in each CUP's learning objectives section according to the main Cognitive Process levels in Krathwohl's revision of Bloom's Taxonomy of Educational Objectives,

* The latter correspond to the three courses that featured computer programming in every semester of their curricula: Fine Arts & Multimedia at the Viseu Polytechnic, Musical Electronics & Production at the Castelo Branco Polytechnic, and Web Technologies & Information Systems at the Porto Polytechnic's Media Arts and Design School.

*"a cumulative hierarchical framework"* in common use (Krathwohl, 2002).[*] Though in Bloom's taxonomy the *"achievement of the next more complex skill or ability required achievement of the prior one,"* Krathwohl's revision has a less rigid hierarchy (2002). We present a summary of our category counts, with examples, in Table 4.2.3:

***Table 4.2.3.*** *Mentions related to Bloom and Krathwohl's cognitive levels found in the learning objectives sections of the curricular unit plans.*

| ***Level*** | ***Units*** | ***Examples*** |
|---|---|---|
| *1 Remember* | 84 | *"Identify the principles of object-oriented programming."*<br>– Programming II, Digital Game Design (Bragança Polytechnic)<br>*"Recognize common technologies for web multimedia application creation."*<br>– Application Development, Multimedia Design & Communication (Coimbra Poly.)<br>*"Describe the structure and the operation of a computer."*<br>– Informatics, Fine Arts & Multimedia (Viseu Polytechnic) |
| *2 Understand* | 66 | *"Understanding the potential of media integration and interaction."*<br>– Multimedia Techniques, Communication Design (Algarve University)<br>*"Distinguish and understand the specifics of interactive multimedia applications and the different technologies available for their development."*<br>– Multimedia Lab III, New Technologies of Communication (Aveiro University)<br>*"Understand the need for employing game engines."*<br>– Intro. to Game Development, Multimedia Communication (Trás-os-Montes Univ.) |
| *3 Apply* | 98 | *"Apply and implement Artificial Intelligence algorithms in a game engine."*<br>– Artificial Intelligence, Digital Games & Multimedia (Leiria Polytechnic)<br>*"Apply concepts of cognitive psychology, affordance, communication, gestalt psychology, and semantic theory to Designing virtual user interfaces."*<br>– Interactive Media Design, Design (Madeira University)<br>*"Know and apply digital image formats and their compression algorithms."*<br>– Introduction to Digital Lighting, Theater—Light & Sound (Porto Polytechnic) |
| *4 Analyze* | 27 | *"To analyze the dynamic webpage technologies available in the market."*<br>– Multimedia Lab IV, New Technologies of Communication (Aveiro University)<br>*"To analyze and extend a game programming library."*<br>– Advanced Topics for Game Programming, Digital Games & Multimedia (Leiria Poly.)<br>*"To analyze the various application areas, techniques, methodologies and audio development tools."*<br>– Multimedia II, Fine Arts & Multimedia (Viseu Polytechnic) |
| *5 Evaluate* | 22 | *"Develop critical and reflexive skills about the contribution of computational aesthetics to the creation of digital artifacts."*<br>– Project Lab I, Graphic & Multimedia Design (Leiria Polytechnic)<br>*"Reflect on the signification and the production of meaning present in the different processes of interaction."*<br>– Real-Time Interaction, Audiovisual & Multimedia (Lisbon Polytechnic) |
| *6 Create* | 91 | *"Create, plan, prototype multimedia systems and applications that integrate interfaces and interactive or dynamic audiovisual content."*<br>– Multimedia Production, Multimedia Design (Beira Interior University)<br>*"Pursue processual and transformative activities enabling the unexpected, the serendipitous, and the indeterminate, so that students experiment different work processes and methods in the development of their Designs."*<br>– Multimedia Lab, Fine Arts (Porto University) |

[*] We chose to apply Krathwohl's revision due to its simplicity and robust dissemination (according to Google Scholar, over 7,500 citations up to 2019).

Contrary to our intuitions, we found no noteworthy co-occurrences between the learning objectives' level and the curricular units' position in the curriculum or the sessions' characteristics.

Some of the CUPs included an additional section about their units' transversal learning objectives. Most mentioned were the acquisition of skills concerning critical thinking (20), autonomous learning (16), teamwork (20), or project management (12). We also found several mentions of competencies in creative thinking (9), logical thinking (4), interdisciplinary work (6), professionalism (4), entrepreneurialism (3), English language (5), visual culture (2), and awareness of the free and open-source software movements (2).

### 2.2.1 THE SYLLABI

Several CUP documents mentioned their units' subfield classification according to the *ad hoc* categorizations devised by courses' host institutions or departments.* Hence, e.g., one course's 'computer science' may be another's 'informatics'. Given their indicative nature, we grouped similar subfields and found units had diverse internal classifications. Still, those curricular units with an internal categorization related to computer science (35), engineering (7), and mathematics (1) were more likely to feature a syllabus that emphasized computer programming foundations and techniques. Other units, more preoccupied with the practical possibilities and outcomes of programming, were classified under design (11), visual arts (5), music (5), communication science (4), and theater (1).

We employed Atlas.ti 8 to explore and code the syllabi included in the curricular unit plans and discover emergent categories. Eight syllabi included an explicit reference to addressing the role of the particular unit within the wider course curriculum. We also found explicit mentions of theoretical content related to contemporary artistic practice (13), digital art (11), Art History (6), media studies (10), design theory (5), and aspects of digital culture (7). Given the overlaps in these contents, only 23 units evinced concern for framing their desired practical objectives within their students' study and practical domains. Most syllabi were solely focused on computing techniques and applications.

Therefore, we found 15 mentions of general computing in the syllabi we reviewed, including computer history (8), system use basics (4), and computer security (4). There were also 7 syllabi that mentioned electronics and 2 that mentioned mathematics. Within content related to computer programming and computer science, we found 27 mentions of algorithmics, including specific mentions of flowcharting (6). We found 68 syllabi that mentioned an *"introduction to programming"* or topics which we may, in any case, also categorize as foundational, such as variables (45), control flow (including loops and conditional structures, 44), and functions (41). We found programming topics we may categorize as intermediate mentioned in 55 syllabi, such as object-orientation basics (i.e., objects, properties, and methods, 34), interfacing with inputs and outputs (21), and

* Though courses' legal acts of creation or modification present the constituting subfields of study and the ECTS credits assigned to each, such fields rarely resembled a known schema. In certain cases, such *ad hoc* subfields comprise those related to a course's practice (e.g., Theater Technique) and those considered generalist (e.g., Art History) so that units thus classified can be shared with other courses at the same institution.

using external libraries and extensions (16). Finally, we found 14 syllabi that included what we categorized as advanced programming topics, relating to the object-oriented paradigm (i.e., inheritance and polymorphism, 8), code optimization (6), and memory management (e.g., pointers, 2). We also found topics we categorized as related to software product development methods in 56 syllabi; we note the subtopics relating to project management (22) and publishing workflows (13).

✦ PRACTICAL APPLICATIONS It is evident that teaching computer programming *per se* was not the main focus of most curricular units, which were rather more concerned with the practical possibilities. We found 46 syllabi including topics related to computer graphics, including two-dimensional (18) and three-dimensional graphics (19), bitmap image processing (16), video manipulation (11), animation (16), augmented or virtual reality (7), typography (3), and data visualization (1). Often, yet not necessarily in tandem with graphics topics, we found mentions of topics related to physics simulations (11) and generative processes (8). We believe it is worth noting 11 of the syllabi that mentioned computer graphics also included topics related to audio. Overall, we found 17 curricular unit plans that mentioned content we categorized as related to sound and music, including audio signal processing (16) and generation and manipulation of MIDI* data (4). Some of the syllabi reviewed also explicitly mentioned text processing (5), data analysis (2), and artificial intelligence (3).

A substantial number of syllabi included topics related to user interaction (36), which included topics about real-time interaction (12), graphic user interfaces (9), and tangible interfaces (14). One single syllabus also mentioned the topic of live coding.

Over a quarter of all curricular unit plans we reviewed included topics we found related to web programming (34). Of these, most syllabi explicitly mentioned frontend web design and development (30), though some included topics related to server-side programming (9) or content management systems (6). Some syllabi mentioned topics related to networking (17) and databases (13), though not always in tandem with web development. Several CUPs reviewed referred to mobile application programming (5). Though two licentiates were dedicated to digital game design, only 9 CUPs explicitly mentioned game development. In addition, we found 4 syllabi that included the topic of interactive narratives; yet only one did so in tandem with game development.

✦ LANGUAGES AND TOOLS We present programming languages mentioned in the 121 CUPs reviewed in Table 4.2.4 (next page). We include the number of different courses in which each language figures in at least one mandatory unit's syllabus, in addition to the courses in which each language can be learned in an elective unit. We also tally the overall number of mandatory and elective units in each language was taught. In the last two columns, we present our estimates, according to the 2018/19 enrollment figures presented in Table 4.2.1, of the numbers of students who could be expected to have mandatory contact and those given the option to learn a language throughout their studies.

* Musical Instrument Digital Interface, a set of standards for representing musical information (e.g., notation).

***Table 4.2.4.*** *Languages taught in undergraduate arts and design education, 2018/19.*

| *Programming Language* | *In courses + electives* | | *In unit plans + electives* | | *Expected learners + elective available* | |
|---|---|---|---|---|---|---|
| *C* | 12 [a] | *+2* | 12 [a] | *+3* | 1,911 | *+390* |
| *(for Arduino)* | 9 | *+2* | 9 | *+3* | 1,545 | *+390* |
| *C++* | 3 | *+1* | 2 | *+4* | 453 | *+319* |
| *(with OpenFrameworks)* | 2 | *+1* | 2 | *+1* | 269 | *+319* |
| *C#* | 8 | *+1* | 12 | *+1* | 1,255 | *+319* |
| *(Unity scripting)* | 6 | *+1* | 7 | *+1* | 971 | *+319* |
| *CSound* | 1 | - | 1 | - | 64 | - |
| *Flash ActionScript* | 4 [a] | *+1* | 4 [a] | *+1* | 380 | *+127* |
| *Grasshopper* | 1 | - | 1 | - | 991 | - |
| *Haskell* | 1 | - | 1 | - | 52 | - |
| *Java* | 5 | - | 6 | - | 783 | - |
| *JavaScript* | 18 [a] | *+4* | 30 [a] | *+3* | 2,844 | *+405* |
| *(Node.js)* | 1 | - | 1 | - | 81 | - |
| *(P5.js)* | 1 | *+1* | 2 | *+1* | 333 | *+319* |
| *Julia* | 1 | - | 1 | - | 372 | - |
| *Matlab / Octave* [b] | 1 | - | 1 | - | 63 | - |
| *Max* | 6 | *+1* | 9 | *+1* | 740 | *+174* |
| *OpenGL & GLSL* | 2 | *+1* | 2 | *+1* | 243 | *+184* |
| *PHP* | 8 | *+1* | 10 | *+1* | 1,548 | *+127* |
| *Portugol* | 1 | - | 1 | - | 189 | - |
| *Processing* | 16 | *+3* | 23 | *+4* | 2,398 [c] | *+467* |
| *Pure Data* | - | *+1* | - | *+1* | - | *+124* |
| *Python* | 2 | *+1* | 2 | *+2* | 252 | *+124* |
| *R* | - | *+1* | - | *+1* | - | *+119* |
| *Scratch* | 1 | - | 1 | - | 136 | - |
| *SPSS Syntax* | - | *+1* | - | *+1* | - | *+119* |
| *SQL* | - | - | 3 | - | 557 | - |
| *Supercollider* | - | *+1* | - | *+1* | - | *+319* |
| *Swift* | 1 | - | 1 | - | 104 | - |
| *Unreal Engine—Blueprints* | 1 | - | 1 | *+2* | 184 | - |

*Note:* [a] Including the discontinued Communication Technology course at the Bragança Polytechnic.
[b] Octave is a free software language that is highly compatible with Matlab.
[c] Not including students enrolled in Theater—Light and Sound at the Porto Polytechnic.

Visual programming was relatively prevalent: Max, Grasshopper, Pure Data, Scratch, and Blueprints were taught in 16 units (12 mandatory) in 11 courses. Over 2,000 students would be required to use one of these languages,* while nearly 300 others had access to elective units. We note that languages such as R and SPSS Syntax (specialized in statistical applications) or Pure Data and Supercollider (audio applications) were only taught in the context of elective units.

Several programming languages bundle their own development environment software as a part of their installation package, though use of that software is not mandatory (as it happens with Processing, CSound, or Portugol). Conversely, other programming languages are embedded in software applications and cannot be used otherwise—as it happens with visual languages such as Max, Blueprints (embedded in the Unreal Engine game creation system), or Grasshopper (embedded in the Rhinoceros 3D design software). Several syllabi, typically concerned with game and multimedia development, explicitly mentioned authoring software—such as Unity (8), Unreal Engine (3), or Adobe Flash/Animate (3). Two syllabi cited unnamed *"game development engines,"* and two others cited unnamed *"multimedia authoring software."* The plans we analyzed also mentioned integrated development software that supports multiple languages, such as Microsoft Visual Studio (2) or Adobe Dreamweaver (2); two additional syllabi cited unnamed *"development environments."* In addition, we found sole mentions of other software tools, such as Adobe After Effects, Microsoft Excel, an unnamed *"musical notation editor,"* and an unnamed *"3D modeling software."* Two syllabi also explicitly mentioned the use of the Git version control system.

Some curricular unit plans mentioned the study and use of hardware tools, especially the Arduino family of programmable computing boards (12). One syllabus mentioned the Raspberry Pi single-board computer, and another mentioned the study of PC hardware configuration. Three syllabi also mentioned hardware and processes we categorized as related to digital fabrication and 3D printing. Still, content related to hardware was always taught in tandem with at least one programming language.

### 2.2.2 SESSION TYPES AND TEACHING METHODOLOGIES

As we have seen, only 33 curricular units scheduled dedicated theoretical sessions. Yet, in their teaching methodology sections, more curricular unit plans explicitly allude to theoretical content exposition (82 in total) or the demonstration of programming examples (55); we note mentions of both teaching methods co-occurred in most cases (48). One sole CUP laid out the intention of inviting specialists to lead workshops.

Most unit plans we reviewed emphasized students' practice: we found significant mentions of practical exercises (66), project development (23), and practical work in the classroom (45); though the latter often co-occurred with practical exercises (28), it did so less often with project development (7). Many CUPs explicitly mentioned supervision of students' work by the professors (58), which is more than double the number of units that had dedicated tutoring sessions in their schedule. A significant portion of the plans

* Nonetheless, almost half the students having contact with visual programming would be attending a single mandatory curricular unit at the integrated master's in Architecture of the University of Lisbon's Faculty of Architecture (FAUL).

also mentioned the presentation and discussion of students' works before the class (24) as part of the teaching methodology. In addition, two plans mentioned that students' work would be articulated with other concurrent curricular units.

According to our review of the methodology sections of the CUPs, students were also expected to conduct autonomous research and study, either individually (28) or with their peers (4). Besides, eight plans explicitly mentioned e-Learning: of these, most mentioned the use of a Learning Management System (6), though two CUPs had an explicit reference to a 'flipped classroom' learning methodology.

### 2.2.3 EVALUATION METHODOLOGIES

Concerning the evaluation methodologies, we reviewed 40 curricular unit plans that mentioned continual evaluation. Of these, most specified grading students' attendance and participation during sessions (37), though some specified the continual evaluation of students' classroom practice (13).

Nonetheless, a periodical evaluation was favored in most CUPs under review (96). Over half of these units required students to complete multiple practical exercises (49), while another large portion required students to develop projects weighing over 50% of the grade throughout the semester (43); of the latter, several CUPs explicitly mentioned the development of group projects (6). Some of the unit plans specified additional requirements for periodical evaluation, such as formal presentations (30) or written reports (12) concerning the practical work. We also found one specific instance in which a project led at another concurrent curricular unit was to be considered for periodical evaluation. Some of the curricular units that favored periodical evaluation also specified the completion of tests (39) or written essays (4) as minor requirements, though several required students to complete a final written test or exam, weighing around 50% of the students' periodical evaluation grade (14). It is noteworthy that 13 of the units requiring the completion of a final exam were classified as relating to computer science. [*]

※

The panorama reviewed throughout this chapter sets the stage for our empirical research of how students and professors at Portuguese public undergraduate arts and design courses accept, teach, and learn computer programming. In the next chapter, we will present our research objectives, design, and methodology. In chapter **6**, we will present the results of our preliminary acceptance surveys of students and professors. In chapter **7**, we will present the findings of a collective case study of six curricular units about computer programming at six distinct courses.

[*] In addition, all units requiring students to complete a final exam were found at the Bragança Polytechnic Institute and at the University of Coimbra. The evaluation requirements may thus be related to institutional policy or tradition.

*(blank page)*

*(blank page)*

PART TWO

# *RESEARCH (DESIGN, METHODOLOGY);*

*(blank page)*

# 5

# *Research design and methods*

★ SYNOPSIS We state our research objectives and describe said research's theoretical framework, design, and methodology. In section §1, we set the researcher's reasons, positioning, and objectives for the investigation. In section §2, we present the theoretical underpinnings, and in section §3, we describe our two-stage research design. In section §4, we describe the instrument, the sample, and the data gathering, validation, and analysis procedures for the first research stage, a quantitative study. In section §5, we present the sample and the procedures undertaken for the second stage of our research, consisting of a collective mixed methods case study.

## 1 THE RESEARCHER'S POSITION AND OBJECTIVES

Our research follows from the literature review presented in chapters 2 and 3 and from the review, presented in chapter 4, of computer programming education offered by Portuguese public undergraduate courses in arts and design. As we stated in the introduction to this thesis, this investigation also stems from the author's own past experiences in the subject, teaching introductory computer programming to students attending a public communication design licentiate, as well as to fine arts and design students at a private higher education institution.

From these reviews and personal experiences, we advance four main structural reasons for the saliency of the present research, which underpinned its objectives and research questions: FIRST, the evolving relationship with technology of arts and design students in the convergence era. SECOND, the severe tensions between, on the one hand, professionalization and meeting the demands of the market, and on the other, a heritage of studio experimentation and inquiry, which have shaped the curricula of arts and design education in the new millennium, and most notably past the implementation of the Bologna Process. A THIRD reason emerges from parallel tensions, though rooted in the history of computing, between the view of computer programming as a job skill and the view of programming as emancipatory knowledge. Finally, a FOURTH reason is the growing availability of programming tools designed with an emancipatory mindset and intended for artists and designers.

## 1.1 *Biases and orientation*

We wish to fully acknowledge the role of the researcher's personal biases concerning the investigation efforts described in this chapter. Rather than following Merriam and Tisdell's (2016) suggestion of closing the research methodology chapter with such a statement (p. 298), we believe a summary of the researcher's assumptions, forged by the aforementioned personal academic and professional practices, ought to lead the discussion of his thesis' research framework, design, and methodology. Given the mostly qualitative nature of the research, we heed the notion that a researcher's biography underpins their approach to the three domains of research activity:

> The researcher, multiculturally situated, creates a set of ideas about the world and their self (ontological domain) that specify a set of questions and modes of knowledge (epistemological domain), which are, in turn, examined in specific ways (methodological domain). (Aires, 2011, pp. 14–15)

The researcher's prior teaching experience had advantages and disadvantages. Being an 'insider' (cf. Sikes & Potts, 2008) to the teaching of computer programming to arts and design students may permit easier navigation of the field, yet that status may also complicate the awareness of biases. We should also acknowledge the formative impact of the researcher's undergraduate studies in filmmaking and his adjacent professional practice. Years working in commercial documentary and reportage videography solidified the researcher's (loosely defined) pragmatic stance toward the collecting and presenting of evidence, which is expressed in his research by an orientation towards describing and understanding practices and processes—*how* questions—rather than investigating causality (i.e., *why* questions). As we will see in the coming sections, this predisposition manifests in the researcher's affinity for tempering his research design with a pragmatic, social interactionist worldview.

## 1.2 *Research objectives and contribution*

Given the reasons mentioned and the researcher's position, this investigation's main objective is to further our knowledge of how computer programming is taught and learned at Portuguese public undergraduate arts and design courses. We set multiple goals rooted in this lead objective:

- To identify factors relating to arts and design students and professors' acceptance of computer programming.
- To determine how do curricula and professors interact with the factors that shape students' acceptance and motivation to use computer programming.
- To determine whether and how professors adapt their teaching of computer programming as to make the subject salient to arts and design students.
- To know how professors value teaching these particular students to code (e.g., emancipatory knowledge, an essential literacy, or a specific job skill).
- To contextualize local practices within our review of international developments.

As we will describe in section §3, our two-stage research design is underpinned by an understanding that these research goals may evolve and additional goals may emerge throughout our inquiries.

We hope the present investigation will offer a good contribution to understanding and enhancing the teaching of computer programming in the context of creative areas of expression, which, as we have seen, is a growing educational concern. Besides, we also expect our work to further the field of computer education research. Though its transferability is not a given, we also wish this work may articulate with and inspire the ongoing research on integrating science and technology education with arts education.*

⁂

In acknowledging the researcher's inexperience, much of the following sections proceeded from a review of several manuals, notably John Creswell's *Educational Research* (2013), Glynis Cousin's *Researching Learning in Higher Education* (2009), Luísa Aires' *Paradigma Qualitativo e Práticas de Investigação Educacional* (2011), Sharan Merriam and Elizabeth Tisdell's *Qualitative Research* (2016), as well as the *Companion to Qualitative Research* edited by Flick, von Kardorff, and Steinke (2004). Though published after we began, we were pleased to find a convergence in Tenenberg's (2019) summary of qualitative methods' applicability to computer education research.

* In other words, as to employ the acronyms which are popular at the time of writing: to grow STEM (science, technology, engineering, and mathematics) into STEAM.

Within any context of formal education, teaching and learning computer programming involves at least two distinct types of the interactions posited by van Dijk (2006): those between people (in this case, the interactions between professor and students, or between different students), and those between people and computer systems. A third type of interaction, between people and documents (such as a programming manual), is also likely involved in the act of learning to code, certainly so in the case of autonomous learning. Each of these types of interactions lends itself to a distinct research program. Hence, as presented in Figure 5.2.1, we will articulate two distinct research domains in our investigation:

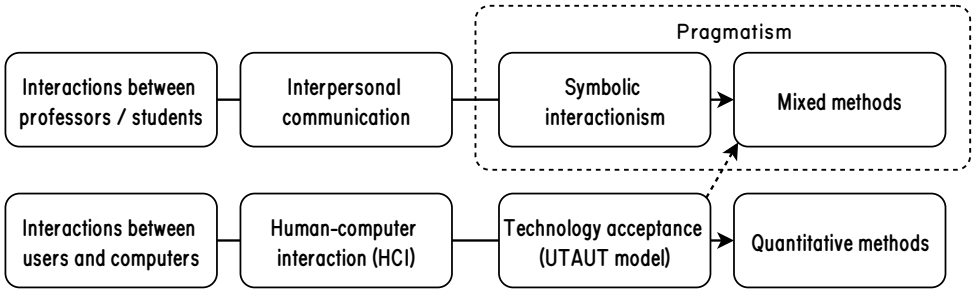



***Figure 5.2.1.*** *Foci and their research domains, theories, and methods (left to right).*

The researcher's pragmatism and teaching experience informs our scrutiny of interpersonal interactions within the classroom. Whereas quantitative research used to hold the positivist view of a stable and measurable objective reality, interpretative qualitative research assumes a socially constructed 'reality,' in the sense that we cannot truly know but one of multiple interpretations, ours (Merriam & Tisdell, 2016, p. 9).[*] It is appropriate multiple philosophies inform interpretivism; we favor the position of symbolic interactionism, which, as we shall present in subsection §2.1, holds that people generate meaning and interpretations through their social interactions.

Computer systems can be configured as a more 'stable' part of reality; hence their study may involve quantitative measurement. The scrutiny of the interactions between people and computers fits perfectly within the Human-Computer Interaction (HCI) research domain and may be underpinned by the field's concern with the dynamics of technology adoption and use. Hence, in subsections §2.2 and §2.3, we will respectively describe models of technology acceptance and the particular model, the Unified Theory of Acceptance and Use of Technology (UTAUT), which we will employ in our research of computer programming acceptance, and which mainly lends itself to the application of quantitative methods.

Nonetheless, we do not see these theories and methods as airtight: in particular, technology acceptance theories factor social interactions and also permit the application of qualitative methods.

[*] Therefore, the interpretative paradigm is not necessarily at odds with the aphorism attributed to science fiction author Philip K. Dick: *"reality is that which, when you stop believing in it, doesn't go away."*

## 2.1 *Symbolic interactionism and the pragmatic paradigm*

Symbolic interactionism is a theory of human society and communication articulated by social psychologist George Mead* and popularized by Herbert Blumer, as Mead's major interpreter, and which follows the American pragmatist philosophical tradition of William James and John Dewey. According to Blumer (1979), symbolic interactionism rests on a set of premises: FIRST, that human action toward things is based on the meanings they ascribe to such things; SECOND, that such meanings arise from social interaction with other people; and THIRD, that those meanings are modified through individuals' reflective interpretations of their interactions with things and with each other. Hence, the same things and events may have different meanings to different people. The web of meanings and human experience succeed from these simple premises, bound by social interaction and self-reflection.

According to Norman Denzin (2004), Mead rejected the notion of unknowable introspection from similar-minded predecessors such as James and favored a view of the individual mind or *'self'* as part of, and interacting with, the social world, and hence an 'object' that can be studied. Furthermore, individuals tend to partially see themselves as others see them (Bogdan & Biklen, 1994, p. 57). Herbert Blumer took Mead's focus on the social act and proposed to describe social interactions at multiple scales, from the interpersonal communication between two persons to those within institutions.

Denzin offers an up-to-date summary of symbolic interactionist research that we found salient to our present research. According to the author, these researchers *"commit themselves to the study and analysis of the developmental course of action that occurs when two or more persons (or agents) [...] join their individual lines of action together into joint action"* (Denzin, 2004, p. 82). The author defends interactionist research from the criticism of lacking clearly defined structures and having emergent theories, noting interactionists value investigation that *"remains closer to the actual experience of people they are writing about"* rather than generalizing theories (p. 83). Hence, Denzin classifies interactionists as *"cultural romantics"* for arguing for a subjective study of human experiences while attempting to build an objective science of human behavior. Yet, he added, interactionists' pragmatism, in which, after Dewey's aphorism, *"what is true is what works,"* remains a viable philosophical position for human sciences researchers (Denzin, 2004, p. 84).

According to Mertens (2010), this underlying pragmatism also provides a philosophic rationale for mixed methods research. According to that author, early pragmatists such as Charles Sanders Pierce and the aforementioned James, Dewey, and Mead rejected the notion that a single scientific method would suffice to reveal the 'truth' about the social world. In contrast, contemporary pragmatist philosophers such as Richard Rorty and Cornel West emphasize practical thinking (pp. 35–36) and "*write extensively of the importance of using mixed methods"* (p. 38).

⁂

* Most notably in his book *Mind, Self and Society*, first published in 1934.

A Google Books search testified to the relevance of symbolic interactionism as a possible theoretical underpinning to our inquiry by yielding sections about the subject in several educational research manuals (cf. Amos Hatch, 2002; Bartlett & Burton, 2016; Bogdan & Biklen, 1994; Cohen, Manion, & Morrison, 2017). Furthermore, we used the software application Publish or Perish 6.32[*] to search academic literature search engines. The search phrases *"'symbolic interactionism' AND 'education'"* yielded over 16,500 literature entries in Google Scholar (excluding citations) up to 2018, as well as 167 papers in the Web of Science Core Collection and 220 papers in Scopus.

Still, Ashwin (2009) noted the third interactionist assumption—how meanings change through self-reflection and interaction—is problematic for educational researchers gathering data from surveys and interviews alone, adding that stand-alone observation would not suffice (pp. 39-40). We took this note as a call for data source triangulation. Ashwin also states, *"symbolic interactionism is helpful in understanding how students and academics negotiate the meaning of particular teaching and learning interactions, and how such negotiations are informed by the careers of academics and students"* (p. 40). Yet, the same author admonishes that interactionism *"does not appear to be helpful in understanding how teaching and learning interactions can facilitate students' critical engagement with the discipline(s)"* (p. 40); hence, there is a need to articulate symbolic interactionism with other theoretical underpinnings.

Several authors sought to do just that with quantitative research in technology and education; e.g., Star (1996) proposed to articulate interactionism and information systems research; Potts (2008) discussed the 'agony' of weaving symbolic interactionism with a positivistic account of the social interactions of academic staff (p. 170); Cowling (2016) shared guidelines on how technology academics doing education research can break from the quantitative mold and move towards interpretivist inquiry.

## 2.2 *Overview of technology acceptance theories*

Given our focus on learning and using computer programming in the arts and design studio, it is appropriate one of our research strands concerns how learners envision their interactions with programming and intend to integrate that newfound knowledge into their practice. A model of acceptance and use of programming will aid our inquiry; to that end, we turned to the literature on technology acceptance.

We used the Publish or Perish 6.32 software to search for *'technology acceptance'* and sort results by the number of citations in academic literature indexes. As of early 2018, Google Scholar yielded as the top three cited papers two articles by Fred Davis presenting the Technology Acceptance Model (Davis, 1989; Davis, Bagozzi, & Warshaw, 1989), with over 60,000 citations in aggregate, and Viswanath Venkatesh, Michael Morris, Gordon Davis, and Fred Davis' paper positing the Unified Theory of Acceptance and Use of Technology (2003), cited over 20,000 times. A search of the Web of Science Core Collection yielded the same top three papers (16,000 citations for Davis' papers and 6,000 for Venkatesh's). Moreover, the top ten most-cited *'technology acceptance'* papers presented by Google Scholar and eight of the top ten found via Web of Science are all

[*] harzing.com/resources/publish-or-perish/

related to either model. In addition, a systematic review of 330 papers about information technology adoption published from 2006 to 2015 found that 168 studies employed the Technology Acceptance Model and that 40 studies employed the Unified Theory of Acceptance and Use of Technology (Rad, Nilashi, & Dahlan, 2018).

### 2.2.1 FROM TAM TO UTAUT: A SHORT CHRONOLOGY

The Technology Acceptance Model adapted to the context of technology adoption a positivist model of individual behavior from social psychology, Icek Ajzen's and Martin Fishbein's (1969, 1972) Theory of Reasoned Action (TRA). The TRA models any action or behavior as the direct consequence of an intention; such a relationship between *Behavioral Intention* and action (e.g., technology usage) remains a characteristic of most technology acceptance theories. The TRA posits that *Behavioral Intention* has two determinants, shown in Figure 5.2.2: individuals' *Attitude* towards the behavior—that is, their belief in a certain outcome; and *Subjective Norm*, defined as individuals' beliefs about others' expectations of them. Such beliefs are posited, in turn, to result from the aggregate of past individual experiences (Ajzen & Fishbein, 1972).

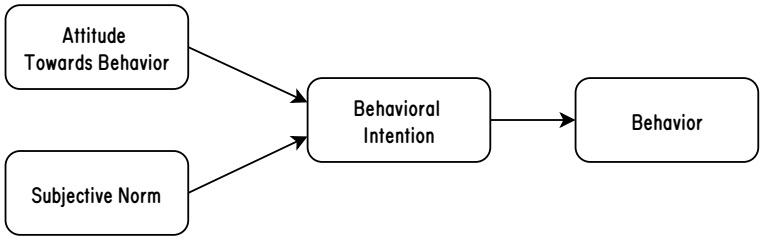



***Figure 5.2.2.*** *The Theory of Reasoned Action* (as in Madden, Ellen, & Ajzen, 1992).

Fred Davis (1986) adapted the Theory of Reasoned Action to the adoption of computer and information systems in the workplace. In his earliest version of the Technology Acceptance Model (TAM), Davis disregarded the role of *Subjective Norm*. He posited two determinants of individuals' *Attitude* towards using a technology: individuals' *Perceived Usefulness*, defined as the *"the degree to which a person believes that using a particular system would enhance his or her job performance,"* and *Perceived Ease of Use*, defined as "*the degree to which a person believes that using a particular system would be free of effort,"* with the latter also posited to be a determinant to the former (Davis, 1989), as shown in Figure 5.2.3:

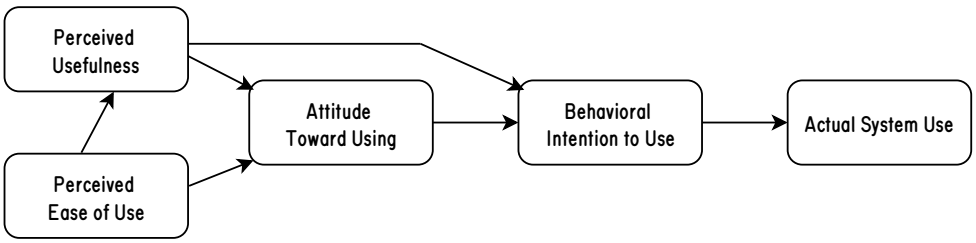



***Figure 5.2.3.*** *An earlier version of the Technology Acceptance Model* (Davis, 1986).

Davis and his colleagues iterated on TAM through empirical tests, and a 'parsimonious' TAM, illustrated in Figure 5.2.4, became the 'canonical' version most cited in the literature. This iterated model further departed from the Theory of Reasoned Action, omitting individuals' *Attitude* towards use and establishing *Perceived Usefulness* and *Perceived Ease of Use* as direct determinants of *Behavioral Intention* (Davis, 1989).

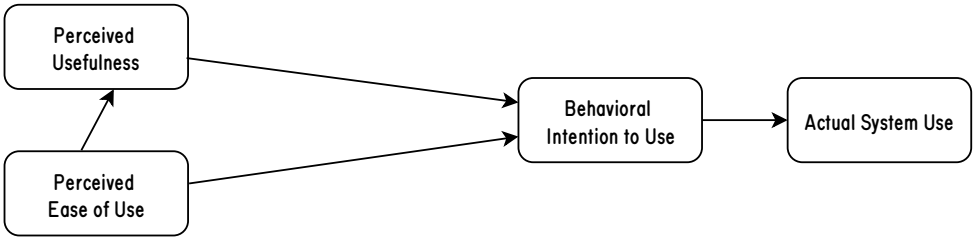



***Figure 5.2.4.*** *'Parsimonious' Technology Acceptance Model* (Davis, 1989).

Iterative development of the Technology Acceptance Model continued apace. Fred Davis and Viswanath Venkatesh proposed the TAM2 in 2000, adding two sets of constructs theorized as determinant of *Perceived Usefulness*; these were related to aspects of social influence (including *Subjective Norm* from the TRA) and cognitive instrumental processes (Venkatesh & Davis, 2000). Venkatesh and Bala proposed the TAM3 in 2008, which added two other sets of constructs theorized as determinant of *Perceived Ease of Use*; these were related to individuals' beliefs about computers (e.g., *Self-Efficacy* and *Anxiety*, which we define in the next subsection) and their perceptions of the facilitating conditions (Venkatesh & Bala, 2008).

Other competing positivistic models of technology acceptance arose in the 1990s: Davis, Bagozzi, and Warshaw's (1992) Motivational Model focused on computer use enjoyment. Thompson and colleagues' Model of PC Utilization saw use as dependent on enjoyment, social factors, facilitating conditions, and separate short-term and long-term perceptions of usefulness and ease of use (Thompson, Higgins, & Howell, 1991). Taylor and Todd combined the TAM and Ajzen's later Theory of Planned Behavior (1991) into a Decomposed Theory of Planned Behavior appliable to technology acceptance (Taylor & Todd, 1995), also foreshadowing TAM2 and TAM3's concern with modeling the secondary determinants of technology use intention and behavior.

Other theories of technology acceptance that circulated in that period were developed outside the positivistic paradigm. The sociologist Everett Rogers' Innovation Diffusion Theory, which still looms large, regards technology acceptance as a cultural diffusion process (cf. Dutton, Rogers, & Jun, 1987). Scholars such as Moore and Benbasat sought to articulate Rogers' model of how technology adoption changes over time with the TRA's model of individual behavior (Moore & Benbasat, 1996). Similarly, Compeau and Higgins rejected TRA and TAM's unidirectional causality and their underlying assumption that individual beliefs result from the sum of past experiences, and proposed a model of technology acceptance articulated with Albert Bandura's Social Cognitive Theory (cf. 1977); Compeau and Higgins put emphasis on the reciprocal relationships between individuals' *Self-Efficacy*, *Anxiety*, enjoyment, expectations, and technology use (Compeau & Higgins, 1995).

## 2.3 *The Unified Theory of Acceptance and Use of Technology*

The increasing technological complexities of the multimedia and the internet era challenged scholars to rein in the technology acceptance theory produced up to and during the 1990s. Apart from working towards integrating aspects of these theories in their newer versions of the Technology Acceptance Model, Viswanath Venkatesh and three other colleagues (including TAM's lead author Fred Davis) proposed a Unified Theory of Acceptance and Use of Technology (UTAUT) from a synthesis of the extant acceptance models (Venkatesh et al., 2003).

The authors thus reviewed two general models of behavior from social psychology—the Theory of Reasoned Action (cf. Ajzen & Fishbein, 1972) and the Theory of Planned Behavior (Ajzen, 1991)—and several technology acceptance theories—the parsimonious version of the Technology Acceptance Model (Davis, 1989), TAM2 (Venkatesh & Davis, 2000), the Motivational Model (Davis et al., 1992), the Model of PC Utilization (Thompson et al., 1991), the Decomposed Theory of Planned Behavior (Taylor & Todd, 1995), as well as Moore and Benbasat's (1996) adaptation of the Innovation Diffusion Theory (cf. Dutton et al., 1987), and Compeau and Higgins' (1995) model of acceptance inspired by the Social Cognitive Theory (cf. Bandura, 1977). According to Venkatesh and his colleagues, all these models understand technology use as the dependent variable while maintaining that intention is the crucial predictor of usage behavior (Venkatesh et al., 2003). Each model postulated between two and seven determinants of intention and actual use of technology, and up to four moderating variables—individuals age, gender, perceived voluntariness of technology use, and experience.

As to synthesize UTAUT, Venkatesh and his co-authors devised four longitudinal studies of the introduction of a new technology in a workplace setting; technology use was mandatory in two such studies (of users' acceptance of an accounting and of a financial portfolio analysis software), voluntary in the other two (of users' acceptance of an audio/video conferencing and of a database software). They designed a questionnaire that aggregated all the validated rating scales from the models they reviewed, except for *Behavioral Intention*—adapted from Davis' (1986) three-item scale—and participants' perceived voluntariness—measured as a single scale, after Moore and Benbasat (1996). Unlike most acceptance research which has, since Davis' earlier TAM studies (1986), relied on participants' self-reporting of technology use, the authors made automated usage measurements by accessing the computer systems' logs. To measure the effect of experience, the researchers applied the survey to each study's participants in three different moments after they completed their software training.

Venkatesh and his colleagues applied statistical regression methods to measure each determinant's direct effect on *Behavioral Intention* (2003) and identified seven latent constructs. The same analysis confirmed the key effect of the four moderating variables. To validate their model, the authors chose the highest-loading questionnaire items from each construct and designed a new shorter questionnaire to test whether the following variables are direct determinants of *Behavioral Intention* or actual technology use (Venkatesh et al., 2003):

- PERFORMANCE EXPECTANCY (PE) is *"the degree to which an individual believes that using the system will help him or her attain gains in job performance"* and a composite of the following constructs: *Perceived Usefulness* (from the TAM); *Job-fit* (from Model of PC Utilization); *Relative Advantage* (from Innovation Diffusion Theory); and *Outcome Expectations* (from Social Cognitive Theory). Validation questionnaire items corresponded to the four highest-loading from *Perceived Usefulness* (1), *Relative Advantage* (2), and *Outcome Expectations* (1).
- EFFORT EXPECTANCY (EE) is *"the degree of ease associated with the use of the system"* and a composite of the following constructs: *Perceived Ease of Use* (from the TAM); *Complexity* (from the Model of PC Utilization); and *Ease of Use* (from Innovation Diffusion Theory). Validation questionnaire items corresponded to the four highest-loading from *Perceived Ease of Use* (3) and *Ease of Use* (1).
- SOCIAL INFLUENCE (SI) is *"the degree to which an individual perceives that important others believe he or she should use the new system"* and a composite of the following: *Subjective Norm* (from the TRA, also part of the TAM2 and the Theory of Planned Behavior); *Social Factors* (from the Model of PC Utilization); and *Image* (from Innovation Diffusion Theory). Survey items corresponded to the four highest-loading from *Subjective Norm* (2) and *Social Factors* (2).
- FACILITATING CONDITIONS (FC) is *"the degree to which an individual believes that an organizational and technical infrastructure exists to support the use of the system"* and a composite of the following: *Perceived Behavioral Control* (from the Theory of Planned Behavior); *Facilitating Conditions* (from the Model of PC Utilization); and *Compatibility* (from Innovation Diffusion Theory). Validation survey items corresponded to the four highest-loading from *Perceived Behavioral Control* (3) and the MPCU's *Facilitating Conditions* (1).
- ATTITUDE TOWARD USING A TECHNOLOGY (ATT) is *"an individual's overall affective reaction to using a system"* and a composite of the following constructs: *Attitude Toward Behavior* (from the TRA, also part of the Theory of Planned Behavior); *Intrinsic Motivation* (from the Motivational Model); *Affect Toward Use* (from the Model of PC Utilization); *Affect* (from the Social Cognitive Theory). Validation survey items corresponded to the four highest-loading from *Attitude Toward Behavior* (1), *Affect Toward Use* (2), and *Affect* (1).
- SELF-EFFICACY (SE) was transposed from Compeau and Higgins' adaptation of Social Cognitive Theory to technology acceptance, wherein these authors define *Self-Efficacy* as the *"judgement of one's ability to use a technology to accomplish a particular task"* (Compeau & Higgins, 1995). UTAUT's validation survey items corresponded to the four highest-loading items proposed by those researchers.
- ANXIETY (ANX) was also transposed from Social Cognitive Theory by Compeau and Higgins, who defined the construct as *"evoking anxious or emotional reactions when it comes to performing a behavior (e.g., using a computer)"* (Compeau & Higgins, 1995). Again, UTAUT's validation survey items corresponded to the four highest-loading items proposed by those researchers.

UTAUT's original authors hypothesized *Performance Expectancy*, *Effort Expectancy*, and *Social Influence* directly determine *Behavioral Intention*, which in turn directly determines actual use alongside *Facilitating Conditions*. The authors hypothesized the three other latent constructs—*Attitude*, *Anxiety*, and *Self-Efficacy*—do not determine either intention or use directly, as their effects would be masked by either *Performance* or *Effort Expectancy* (Venkatesh et al., 2003). In devising the empirical validation, the authors maintained the scales measuring *Behavioral Intention* and participants' perceived voluntariness from the model comparison studies, as well as the automated procedure for measuring system use. Venkatesh and colleagues conducted two cross-validation studies (one involving the mandatory use of a customer service application, the other involving the voluntary use of a financial analysis software) that followed the same timeline as the previous model-comparison studies (Venkatesh et al., 2003). The authors confirmed their hypotheses and hence the model illustrated in Figure 5.2.5:

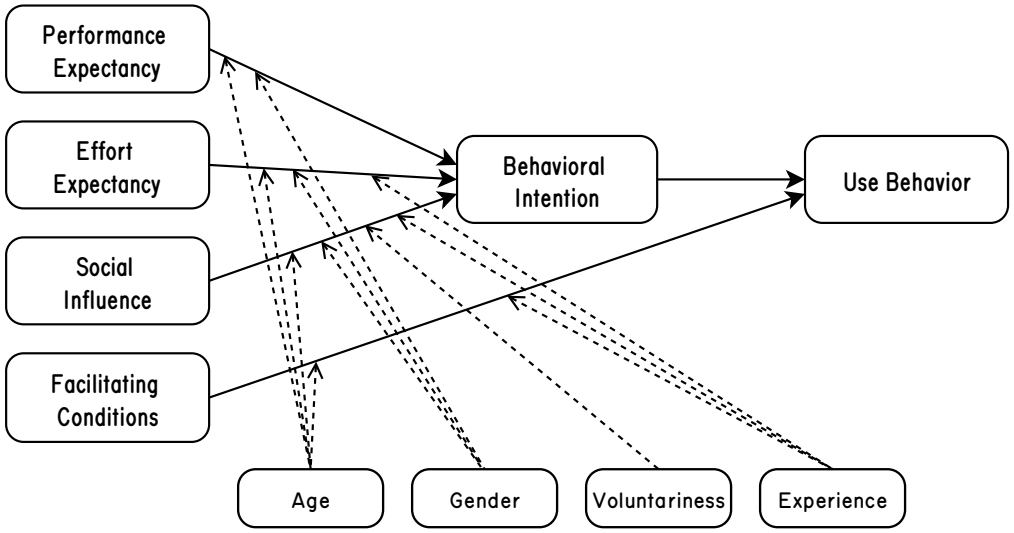



***Figure 5.2.5.*** *Unified Theory of Acceptance and Use of Technology, as validated in the original research paper* (Venkatesh et al., 2003).

As validated in the original paper, UTAUT establishes that age, gender, experience, and perceived voluntariness moderate how *Performance Expectancy*, *Effort Expectancy*, *Social Influence* determine an individual's *Behavioral Intention* to use a technology and the *Facilitating Conditions* determines their actual technology use (Venkatesh et al., 2003). Discussing their findings, the authors questioned whether gender differences would remain noticeable in the future and whether the role of *Social Influence* is dependent on the specific context of technology use.

⁂

As with the Technology Acceptance Model, there was continual development of the UTAUT, yet along different lines. Venkatesh and his colleagues James Thong and Xin Xu proposed the UTAUT2 model of individual acceptance of consumer technology (2012). Unlike in Venkatesh's TAM research about secondary determinants (Venkatesh & Bala, 2008; Venkatesh & Davis, 2000), UTAUT2 posits additional direct determinants of *Behavioral Intention* or actual use of a technology: *Price Value*, *Habit*, and *Hedonic*

*Motivation* (i.e., enjoyment). Besides, given the focus on consumer technology, the UTAUT2 disregards the role of individuals' perceived voluntariness (Venkatesh et al., 2012). Hence, whereas TAM2 (Venkatesh & Davis, 2000) and TAM3 (Venkatesh & Bala, 2008) deepen the parsimonious TAM model (Davis, 1989) without modifying the central relationships therein posited, UTAUT2 widens the original model but also modifies its assumptions about the context of application.* Hence, it is understandable scholarship related to organizational and educational contexts favors the original UTAUT model.

### 2.3.1 UTAUT IN LITERATURE

In 2015, Williams, Rana, and Dwivedi conducted a systematic review of 174 articles applying the UTAUT model and identified 40 studies conducted with students. In the literature they reviewed, the technologies assessed for user acceptance ranged from general (*'ICT,' 'Digital learning'*) to specific (*'Microblogging application'*). Yet, no studies were identified either involving arts education, computer programming, or a specific language or tool, stating to our research gap (Williams, Rana, & Dwivedi, 2015). Still, such UTAUT studies, though rare, are not unprecedented, as testified by the research on the acceptance of tablet computers by arts and design professors at universities in Cyprus and the United Kingdom (Souleles, Savva, Watters, & Annesley, 2017) or of Augmented Reality technologies by arts students at a Spanish university (Cabero-Almenara & Roig-Vila, 2019), as well as the research on the general acceptance of computer programming by students at a Nigerian university (Dasuki, Ogedebe, & Idemudia, 2016) or Malaysian engineering students' acceptance and use of a 3D graphics programming language (Binti Shamsuddin, 2009).

Most of the studies Williams, Rana, and Dwivedi (2015) reviewed had applied the UTAUT instrument in a heterodox manner. Several studies introduced additional direct determinants of *Behavioral Intention* and technology use, others introduced secondary determinants, and others yet both widened and deepened the model; there was also a great heterogeneity in the studies' application of the moderating variables. Another literature review by Venkatesh, Thong, and Xin (2016) identified three types of extant contributions to UTAUT research: application of the model (as-is) in new contexts, contextualization of the model through new moderation mechanisms, and extension of UTAUT through integration with other theories and addition of new constructs.

Williams and colleagues (2015) frequent limitations in the studies they reviewed: limited sample sizes, self-reporting of actual technology use, not accounting for the effect of the moderating variables, and a paucity of longitudinal studies. The authors also found discrepancies between the original validation of UTAUT carried out by Venkatesh and colleagues (2003) and subsequent research, noting the only robust relationships throughout the literature are those between *Performance Expectancy* and *Behavioral Intention* and between the latter and actual usage (Williams et al., 2015). The authors also found several studies that reexamined to what extent *Self-Efficacy* (21 occurrences), *Attitude* (20), or *Anxiety* (12) relate to intention or behavior.

* Hence, we believe UTAUT2 naming is misleading. UTAUCT (C for 'consumer') would more adequately reflect the model's placement among technology acceptance models.

Indeed, a subsequent meta-analysis, authored by some of the same researchers, validated *Attitude Towards Use* as a direct determinant of *Behavioral Intention* (Dwivedi, Rana, Jeyaraj, Clement, & Williams, 2017) while repositioning individuals' *Performance* and *Effort Expectancy* as direct determinants of *Attitude*, hence going back to Fred Davis' original Technology Acceptance Model (1986), much like contemporary TAM research (cf. Marangunić & Granić, 2015). Dwivedi and colleagues thus reversed the original assumption holding *Attitude* as 'masked' by an individual's expectancies (Venkatesh et al., 2003). Furthermore, a distinct meta-analysis, which did not examine variables not included in the original model, also questioned whether *Facilitating Conditions* could be direct determinants of *Behavioral Intention* (Khechine, Ndjambou, & Lakhal, 2016).

### 2.3.2 CRITICISMS AND LIMITATIONS

Though the Unified Theory of Acceptance and Use of Technology was found to predict as much as a 70% variance in the individual intention to use a technology (Venkatesh et al., 2003), Brown, Dennis, and Venkatesh himself (2010) criticized the Unified Theory of Acceptance and Use of Technology for lacking explanatory power, as conclusions from stand-alone UTAUT research are difficult to translate to practical interventions. The authors called for continued theoretical development of contextual UTAUT models adapted to specific technologies (Brown et al., 2010), echoing the similar calls in TAM scholarship (cf. McFarland & Hamilton, 2006).

The UTAUT's original exclusion of individual psychological processes such as *Attitude, Self-Efficacy*, and *Anxiety*—has been a point of contention. As we have seen, recent literature reinstated *Attitude* as a determinant of *Behavioral Intention* (Dwivedi et al., 2017), while Kreijns and colleagues argued for a new model of technology adoption that posited *Attitude, Subjective Norm,* and *Self-Efficacy* as direct determinants of intention (Kreijns, Vermeulen, Kirschner, & Buuren, 2013).

Parameswaran, Kishore, and Li (2015) have criticized the lack of research about the UTAUT survey instrument's measurement invariance. These authors established individual respondents' interpretation of the survey items, though invariant regarding the other moderators such as age or gender, are dependent on their prior engagement with the specific technology under scrutiny, thus adding a layer of contextual specificity to each UTAUT application. Still, the same paper also established the model's scales as invariant to usage behavior, hence preserving the model's assumption of unidirectional relationships (Parameswaran et al., 2015).

### 2.3.3 UTAUT'S APPLICABILITY TO THE PRESENT RESEARCH

Acknowledging its explanatory limitations (cf. Brown et al., 2010) and the contradictions exposed in the literature regarding the model's unidirectional relationships between variables—a reason for us to favor the mixed methods design we shall present in the next section—we nonetheless saw UTAUT accepted as a reliable instrument (Lewis, Fretwell, Ryan, & Parham, 2013; Williams et al., 2015). Given the habitual use of the model in educational technology acceptance research and precedents of its application in the context of arts education and to the subject of computer programming, we saw

UTAUT as a valuable component of our research framework and as a valid instrument to include in our research design, which we shall detail in the next section.

Among its strengths, UTAUT is a relatively parsimonious model that allows for a concise questionnaire given the breadth of the measurements we intended to make. We found the model's sensitivity to perceived voluntariness quite important, as our research will comprise educational situations in which students are voluntarily seeking contact with computer programming (as in the case of elective curricular units), or in which an evaluation methodology through group projects may permit certain students to adjust their commitment to programming.

The constructs hypothesized and determined by Venkatesh and his colleagues (2003) to lack a direct causal relationship with intention or use—*Attitude Toward Use of Technology*, *Self-Efficacy*, and *Anxiety*—either incorporated or were fully transposed from Compeau and Higgins' adaptation of Social Cognitive Theory to technology adoption (1995), which posited reciprocal relationships between acceptance, intention, and use. We note such reciprocity was a possibility left unexamined in the formulation of UTAUT. We also note how standing of *Attitude* has been rethought and reshuffled in literature employing the UTAUT model (Dwivedi et al., 2017) and the TAM family of models (cf. Marangunić & Granić, 2015).

This 'reshuffling of cards' can be seen as a weakness in the underlying assumption of causality that underpins most technology acceptance theory. From our pragmatic standpoint, however, we found that *Attitude*, *Self-Efficacy*, and *Anxiety* are in themselves salient to the context and the subject of our research. Though an interrogation of the UTAUT model is not an immediate research objective, we found its frequent application in acceptance research to encourage adaptation to contextual and topical needs. Though from a position of relaxed regard for unidirectional causality, we perceived usefulness in employing the UTAUT as a research instrument and as a descriptive model to scrutinize the interactions that professors and students have with computer programming.

# 3 RESEARCH DESIGN

> Do not let the tail wag the dog—research methods are in the service of the researcher, not vice versa. (Cousin, 2009, p. 2)

As described in the previous section, our investigation focuses on the two main types of interactions involved in teaching and learning computer programming within a formal environment: those between the professor and the students or between colleagues, and those between learners and computer systems. As we have seen, we believe these two types of interactions lend themselves to exploration through two distinct theoretical strands. Though there is significant variability in the design of computer systems and the programming languages taught to arts and design students (as seen in 4.2), we believe said systems nonetheless constitute objective realities that may serve as a focal point for applying quantitative acceptance models and methods. On the other hand, we believe that investigating teaching and learning interactions and the consequent changing interpretations of said experiences by those involved require multiple qualitative methods.

The dynamics of learning leads us to mind Glynis Cousin's (2009) advice against a linear view of empirical research, as the activities of literature review, question formulation, data gathering, data analysis, and writing should not necessarily be understood as sequential steps but rather as *"dynamically linked and continually enlivened by an engagement with a wide reading"* (p. 3).

⁂

Though our research activities are not to be construed as airtight and as sequential as a 'waterfall' process, we nevertheless devised a two-stage research process which we shall describe in this section. Given our investigation's overall qualitative slant, as we aim our contributions at the educational process rather than computer systems development, we will dedicate the following subsection to examine the main characteristics of qualitative inquiry and a discussion of mixed methods research. In subsection §3.2, we will then detail our research design.

## 3.1 *Characteristics of qualitative research*

Merriam and Tisdell (2016) summarize qualitative research as the *yang* to quantitative *yin*: whereas the latter aims at providing causal explanations or measurable descriptions, qualitative research aims at *"understanding how people interpret their experiences, how they construct their worlds"* (p. 6). They specify four major characteristics of qualitative inquiry: FIRST, that aforementioned focus on interpretation; SECOND, that researchers themselves are the primary instrument of data collection and analysis; THIRD, that research is inductive, and its goal is to generate theory rather than to test hypotheses deductively; and FOURTH, that the outcome of qualitative research is highly descriptive (pp. 15–18). The authors also argue the ideal qualitative research design is *"emergent and flexible, responsive to the changing conditions of the study in progress"* (p. 18).

The third of the aforementioned characteristics merits additional discussion. Most research manuals we reviewed (cf. Creswell, 2013; Flick et al., 2004) root the principle of generating theory from the inductive analysis of phenomena rather than the testing of hypotheses in the arguments put forward by Barney Glaser and Anselm Strauss in their 1967 book *The Discovery of Grounded Theory*, which were themselves influenced by symbolic interactionism (cf. Blumer, 1979). Meinefeld (2004) contrasts the need for conscious structure and accounting for biases in quantitative methodology with the need for appropriate categorization and 'openness' towards the field in qualitative research; nonetheless, the author also rejects the ideal of an unprejudiced and objective researcher unencumbered by previous knowledge (p. 154).

Indeed, to the mind of the particular researcher writing this thesis, the idea that objective qualitative research could be possible recalls the unsophisticated notion of objectivity in documentary filmmaking or reportage brought forth by movements such as *cinema verité* and direct cinema in the 1960s, which proposed such naïve objectivity as a reaction against didacticism or propaganda (cf. Aufderheide, 2007; Beattie, 2004). Even so, one notable early practitioner of that style of *'fly-on-the-wall'* documentary enabled by portable equipment, Frederick Wiseman, came round to describe his films as *"reality fictions"* (Aufderheide, 2007). This realization has a historical parallel with the crises of representation and legitimacy in qualitative social science research, and a turn towards linguistic and interpretative procedures, which emphasizes lived experience *"is created in the social text written by the researcher"* (Aires, 2011, p. 12).

Hence, Meinefeld's (2004) comment that *"the first setting up of data is already an active undertaking on the part of the researcher and is based on the individual's research interest and prior understanding"* (p. 155) echoes that commonplace rebuttal of objective filmmaking—that whether to bring a tripod reflects an opinion. Likewise, Meinefeld concurs *"the idea of a 'direct' record of social reality [is] untenable from an epistemological viewpoint"* (p. 155). Moreover, observations will be given structure and meaning from researchers' previous knowledge and personal biography; as Aires (2011) put it, *"there is no clear window into people's interior life—any sight is filtered by language, gender, social class, ethnic group"* (p. 17).

Cousin (2009) quotes Silverman* to raise another important point: a researcher's *"tacit sense of the sort of appearance or shape of a worthwhile piece of research,"* hence the degree to which "*research is informed by an aesthetic"* (p. 12). Cousin thus advises that research texts should not be given too much determining power (p. 12), advocating a 'middle path.' In the same vein, Meinefeld (2004) advocates reconciliation with hypothesizing from a distinction *"between the principled methodological openness and the explicitness with which prior knowledge is reflected and expressed"* (p. 156).

* Silverman, D. (1997). Towards an aesthetics of research. In D. Silverman (ed.), *Qualitative Research: theory, method and practice* (pp. 239–253). London: Sage.

### 3.1.1 MIXED METHODS RESEARCH

According to Luísa Aires (2011), qualitative research is *"multiparadigmatic in focus, and its practicing researchers are sensitive to the value of a mixed methods approach"* (p. 13). Indeed, mixed methods are salient to our research objectives. Merriam and Tisdell (2016) place the dawn of mixed methods research in the early 1970s after researchers started to combine quantitative surveys with interviews (p. 45). The authors explain how this practice soon led to debates regarding a contradiction between qualitative methods and the positivist epistemology of quantitative research. As they recount, though purists argued against mixing research methods, pragmatists relished the benefits of engaging multiple paradigms to investigate a problem (pp. 45–46). Hence, after those initial debates, the last two decades have been characterized by *"an expansion of possibilities of mixed methods approaches [… and] a perception of such approaches as methodology in their own right"* (Merriam & Tisdell, 2016, p. 46).

Technology acceptance models offer useful instruments to investigate how professors and students react towards computer programming, yet the quantitative methods do not offer a way to look at the *'unknown unknowns'* we expect from arts and design studio practice. As Flick, von Kardorff, and Steinke (2004) put it:

> Standardized methods need […] some fixed idea about the subject of the investigation, whereas qualitative research can be open to what is new in the material being studied, to the unknown in the apparently familiar. (p. 5)

Far from standing their putative positivistic ground, technology acceptance theorists Venkatesh, Brown, and Bala (2013) published guidelines for conducting mixed methods research that generally meet the recommendations we found in the literature reviewed, notably quoting from John Creswell (cf. 2013) and Donna Mertens (2010).

## 3.2 *Our research design*

We believe Uwe Flick is correct to warn against undervaluing what resources will be available in designing research (Flick, 2004a). One researcher carried out the present investigation under the time and budgetary constraints of the doctoral grants. Moreover, our research was constrained by the academic calendars of the higher education institutions possibly hosting our research and the help granted by the gatekeepers, themselves under understandable constraints (Mertens, 2010, pp. 329–330). Moreover, our empirical investigation started immediately before and continued after the implementation of the EU's General Data Protection Regulation (GDPR) in May 2018, which brought constraints due to higher education institutions' rightful caution in ensuring adherence to the new regulations if they were to aid our investigation. Given the time constraints, we strove to build the preservation of participants' anonymity into our design, as to ensure institutions' timely approval.

Our research design may be described as similar to what Creswell (2013) names as an *explanatory sequential mixed methods design*, defined as a two-phase model in which quantitative data is collected first as to build toward a general answer to the research questions, and qualitative data is gathered afterward as to refine those general findings.

Yet, the author notes a primacy of the quantitative over the qualitative, in that the latter is supplemental and is not aimed at answering additional research questions. On this crucial point, our research design departs from the explanatory model and instead shares characteristics Creswell (2013) assigns to *multiphase designs*, which *"address a set of incremental research questions that all advance one programmatic research objective"* (p. 547). The following figure presents a diagram of our research design:

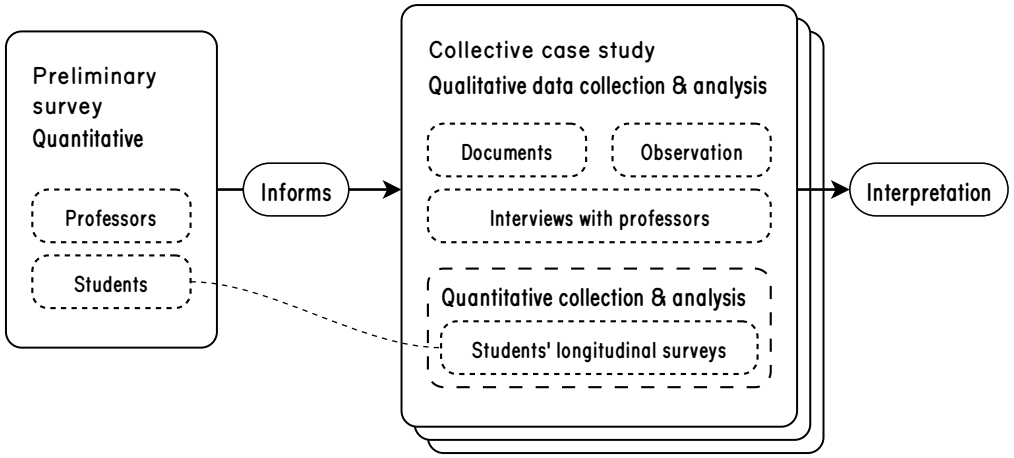



***Figure 5.3.1.*** *Our two-phase research design (dotted line denotes a shared instrument).*

Building a provisional answer to our first research question—to identify factors relating to arts and design students and professors' acceptance of computer programming—was the lead objective of the preliminary phase of our investigation. The identification of relevant factors raised questions and informed the second phase, offering a map for navigating the collective case study—nonetheless, not disregarding the need for openness towards the field. These two research phases feature particular designs, which we shall detail in the following subdivisions.

### 3.2.1 PRELIMINARY SURVEY

Our research's preliminary phase consisted of a quantitative *cross-sectional survey* (Creswell, 2013, p. 377) of professors and students' computer programming acceptance and use. We applied survey instruments based on the Unified Theory of Acceptance and Use of Technology (§2.3), following the procedures we will describe in section §4. As discussed, UTAUT was intended as a predictive instrument by its authors (Venkatesh et al., 2003), though some subsequent literature may question the model's assumption of robust unidirectional relationships. Hence, this phase also meets Creswell's (2013) definition of an *explanatory correlational* design (p. 340): our analysis of the survey went beyond descriptive reporting and aimed to identify relationships between participants' individual characteristics and their measures of acceptance and use of programming, as well as between their latter measures.

### 3.2.2 CASE STUDIES

According to John Creswell (2013), *"a case study is an in-depth exploration of a bounded system (e.g., activity, event, process, or individuals) based on extensive data collection"* (p. 465). Following the author's typology, the second stage of our research consisted of a *collective case study* comprised of multiple *instrumental cases* to be compared.[*] As we will describe in section §5, each such case corresponded to a specific curricular unit in which computer programming is taught to arts and design undergraduate students.

These case studies were carried out during both terms of the 2018/2019 academic year. As mentioned, we were worried about our GDPR compliance and about assuring said compliance to the institutions we contacted as to host our studies, as to receive timely authorizations to carry out the studies. We strove to integrate the protection of students' anonymity into our research design, even though we recognize this brought limitations to our studies. Therefore, though interviewing students or organizing focus groups would have been valuable, we substituted such data collection methods with a longitudinal survey of students (carried out at the start and at the end of the semester), employing the same UTAUT survey instrument as the preliminary survey, in order to ensure expedient approval and institutional support.

Hence, given the multiple data gathering methods involved, each instrumental case study may itself conform to an *embedded mixed methods design* (Creswell, 2013), as quantitative data collection—the longitudinal survey panel of students—supported the collection of qualitative data—gathering documents and artifact, field observation in the classroom and online, and interviews with the professors.

As we will further detail in §5.3, the simultaneous gathering of data from multiple sources called for triangulation in analysis, both as to cross-validate research results and enlarge *"perspectives that permit a fuller treatment, description and explanation of the subject area"* (Kelle & Erzberger, 2004, p. 174).

⁂

Triangulation also plays a role in the overall analysis and discussion of findings from the two stages of our research. Not only we collected data from different sources, but we also employed different methods. Given this last point, *theory triangulation* is also, to a certain extent, inevitable (Flick, 2004b).

[*] With that in mind, we felt more comfortable referring to our case studies in the plural while writing this thesis.

# 4 QUANTITATIVE RESEARCH

Teaching, learning, and using computer programming in an educational environment presents a challenging and heterogeneous research context. As seen in the previous sections, we find a technology acceptance model useful to scrutinize the interactions between learners or educators and computer systems. Therefore, the quantitative strand of our investigation aims at obtaining an answer to questions that immediately succeed from our research objectives: FIRST, how do arts and design students facing contact with computer programming accept and use that technology? SECOND, how do professors involved in arts and design education envision and adopt computer programming? As seen in §2.3, Venkatesh's (2003) Unified Theory of Acceptance and Use of Technology (UTAUT) permits additional queries: ONE, how do individuals' characteristics—age, gender, and computing experience—shape their engagement with programming? TWO, to what an extent do individuals' perception of voluntariness in regard to their learning or use of computer programming shape their acceptance?

In this section, we shall describe the instruments (§4.1), the samples and data collection procedures (§4.2), and the data validation and analysis procedures (§4.3) we engaged in the quantitative strand of our investigation.

## 4.1 *The survey instruments*

We devised questionnaires for professors and students in which we adapted the items identified by the original UTAUT authors (Venkatesh et al., 2003) as the most reliable and consistent measurements of *Behavioral Intention* and of its respective determinant constructs—*Performance Expectancy*, *Effort Expectancy*, and *Social Influence*. We also adapted the items measuring the *Facilitating Conditions*, conceived to determine actual use, as well as the items measuring *Attitude Toward Use of Technology*, *Self-Efficacy*, and *Anxiety*, which were either suggested to determine intention in subsequent literature (cf. Dwivedi et al., 2017) or are salient to our specific research topic. We thus rewrote the items as to relate to computer programming. As the items were to be presented as Likert rating scales, we pondered Krosnick, Judd, and Wittenbrink's (2005) discussion of the optimal number of rating points, after which we settled for 7-point scales.

We wrote two variants of each set of items, adapting the phrasing to participants' prior familiarity with programming, in recognition of Parameswaran and colleagues' (2015) suggestion that these constitute distinct contexts of application. In the *indicative variant*, applied to participants familiar with programming or having their first contact at the time of the survey, we wrote items in the indicative mood. We exemplify this variant of the questionnaire as destined for students in Table 5.4.1 (next page). In the *conditional variant*, applied to participants lacking any programming familiarity, we phrased the items in the conditional mood; we also removed one *Facilitating Conditions* item about system compatibility (FC3). We exemplify the conditional variant items of the survey destined for professors in Table 5.4.2 (overleaf). Though the questionnaires intended for completion by professors and students were similar, we made slight modifications to the items related to career or the workplace (PE4, SI3).

***Table 5.4.1**. Retroversion of the acceptance and intention items (indicative mood)—students with a familiarity with programming or learning for the first time.*

| *Construct* | *Items* | |
|---|---|---|
| *Performance Expectancy* | PE1 | Knowledge of computer programming is useful for my job. |
| | PE2 | Knowledge of computer programming allows me to perform my tasks faster. |
| | PE3 | Knowledge of computer programming increases my productivity. |
| | PE4 | Computer programming increases my chances of scoring higher grades or earning more money in the future. |
| *Effort Expectancy* | EE1 | My interaction with the programing languages I'm using is clear and understandable. |
| | EE2 | It's easy for me to master a programming language. |
| | EE3 | I think computer programming is easy. |
| | EE4 | Learning a programming language is easy for me. |
| *Attitude Toward Use* | At1 | Computer programming is a good idea. |
| | At2 | Computer programming makes my work more interesting. |
| | At3 | Computer programming is fun. |
| | At4 | I like to program computers. |
| *Social Influence / Subjective Norm* | SI1 | People who influence my behavior think I should use computer programming. |
| | SI2 | People who are important to me think I should use computer programming. |
| *Social Influence* | SI3 | The professors in this course help me learn and use computer programming. |
| | SI4 | In general, this school supports learning and using computer programming. |
| *Facilitating Conditions* | FC1 | I have the resources required to program computers. |
| | FC2 | I have the knowledge required to program computers. |
| | FC3 | The programming languages taught at my school are not compatible with other systems or languages I'm currently using. * |
| | FC4 | Someone is available for assistance should I experience any difficulties. |
| *Self-Efficacy* | | *I can complete a work or task using computer programming…* |
| | SE1 | Even if no one is available to guide me step-by-step. |
| | SE2 | If I can ask someone for help in case I got stuck. |
| | SE3 | If I have a lot of time to complete it. |
| | SE4 | If I only have access to help files or the programming language documentation. |
| *Anxiety* | An1 | I'm apprehensive about computer programming. |
| | An2 | I'm scared to think I could lose a lot of information if I make a simple mistake. |
| | An3 | I hesitate to program for fear of making mistakes I cannot correct. |
| | An4 | Computer programming is somewhat intimidating to me. |
| *Behavioral Intention* | BI1 | I intend to use computer programming in the next 12 months. |
| | BI2 | I predict I would use computer programming in the next 12 months. |
| | BI3 | I plan to use computer programming in the next 12 months. |

* This question was solely presented to students reporting some familiarity with computer programming.

***Table 5.4.2.*** *Retroversion of the acceptance and intention items (conditional mood)—professors without any knowledge of programming.*

| *Construct* | *Items* | |
|---|---|---|
| *Performance Expectancy* | PE1 | Knowledge of computer programming would be useful for my job. |
| | PE2 | Knowledge of computer programming would let me perform my tasks faster. |
| | PE3 | Knowledge of computer programming would increase my productivity. |
| | PE4 | Knowing computer programming would increase my chances of career progression or of getting a raise. |
| *Effort Expectancy* | EE1 | My interaction through a programing language would be clear and understandable. |
| | EE2 | It would be easy for me to master a programming language. |
| | EE3 | I think computer programming would be easy. |
| | EE4 | Learning a programming language would be easy for me. |
| *Attitude Toward Use* | At1 | Computer programming would be a good idea. |
| | At2 | Computer programming would make my work more interesting. |
| | At3 | Computer programming would be fun. |
| | At4 | I would like to program computers. |
| *Social Influence / Subjective Norm* | SI1 | People who influence my behavior think I should learn programming. |
| | SI2 | People who are important to me think I should learn computer programming. |
| *Social Influence* | SI3 | The board of the institution would provide for my learning of programming. |
| | SI4 | In general, my school would support learning and using programming. |
| *Facilitating Conditions* | FC1 | I would have the resources required to program computers. |
| | FC2 | I would have the knowledge required to learn to program computers. |
| | – | *Omitted.* |
| | FC4 | Someone would be available to provide assistance should I experience any difficulties programming. |
| *Self-Efficacy* | *I would be able to complete a work or task using computer programming…* | |
| | SE1 | Even if no one was available to guide me step-by-step. |
| | SE2 | If I could ask someone for help in case I got stuck. |
| | SE3 | If I had a lot of time to complete it. |
| | SE4 | If I only had access to help files or the programming language documentation. |
| *Anxiety* | An1 | I would be apprehensive about computer programming. |
| | An2 | I would be scared to think I could lose a lot of data from a simple mistake. |
| | An3 | I would hesitate to program for fear of making mistakes I cannot correct. |
| | An4 | I would be intimidated at the thought of computer programming. |
| *Behavioral Intention* | BI1 | I intend to learn computer programming in the next 12 months. |
| | BI2 | I predict I would learn computer programming in the next 12 months. |
| | BI3 | I plan to learn computer programming in the next 12 months. |

As we will describe in further detail in §4.1.2, the questionnaires we applied were written in Portuguese (see Annexes); hence we conducted pilot tests of our translation through focus groups and by comparing a professional translator's retroversion against the reference UTAUT questionnaire by Venkatesh and colleagues' (2003).

### 4.1.1 CHARACTERIZATION AND PROGRAMMING USE QUESTIONS

We devised a set of questions that were presented to students and professors before the main questionnaires. We asked participants their gender and age, following the original authors' recommendation for the latter to be measured as a continuous variable (Venkatesh et al., 2003). After the suggestion IT engagement may also constitute a UTAUT moderator (Parameswaran et al., 2015), we asked participants to rate their daily computer use from a set of intervals. In addition, we asked students their seniority (i.e., the curricular year they were presently enrolled in) out of suspicion it might have a moderating effect similar to age, as well as to identify their course and whether they had already attended or expected to attend a unit about computer programming. Professors, on the other hand, were asked to fill out their number of years of teaching experience. Students and professors alike were also asked to alike to fill out their institution.*

Surveys' participants were asked about their present knowledge of programming, which we equated with the UTAUT moderating variable related to their experience: professors were presented with a binary option, while students were given a ternary option, which allowed for the possibility they would be learning to code for the first time at the time of the survey. Based on this answer, we designed the survey as to request participants without knowledge of programming to complete the questionnaire written in the conditional mood (exemplified in Table 5.4.2).

The remaining participants were presented with additional questions about their programming use and knowledge: two optional multiple-choice questions asked these participants to describe the places or situations where they had learned to code and the programming languages they knew how to use. We also asked professors whether they would be teaching programming during the current academic year and, if that were the case, which programming languages were part of their syllabi. Hinting at the next stage in our investigation, we also presented professors with an open question about how they value teaching programming to arts and design students.

Following the UTAUT model, we presented those participants having any contact with computer programming, including first-time learners, with questions devised to assess their perceived voluntariness and actual use of programming. Following the original authors' use of Moore and Benbasat's scale (1996; Venkatesh et al., 2003), we asked participants to rate their perception of programming use within their work or studies on a 7-point scale (from fully mandatory to fully voluntary). Understanding self-reporting is a frequent limitation in UTAUT studies (Williams et al., 2015), our procedures precluded direct usage measurements; hence we followed the suggestion of Davis' earlier research (1986) and asked participants to rate computer programming use via two measurements: choosing the appropriate frequency interval (six options, from 0: never and 1: less than weekly, to 5: multiple times a day) and rating their programming use along a 7-point intensity scale (from as little as possible to as much as possible).

We thus summarize the characterization questions included in each questionnaire variant in Table 5.4.3:

* As professors may be affiliated with multiple institutions, our question asked the name of the institution from whence they received the request for participation in the survey.

***Table 5.4.3.*** *Participant characterization and computer programming usage questions.*

| | *Students' surveys* | *Professors' preliminary survey* |
|---|---|---|
| *All participants* *(mandatory response)* | Gender<br>Age *(Continuous integer)*<br>Higher Education Institution<br>Daily computer usage *(Intervals: < 1 hour, 1–2, 2–4, 4–8, > 8 hours)* | |
| | Course<br>Seniority *(Year at the course)*<br>Have you attended or will attend any unit about programming?<br>*Case studies only:*<br>Are you repeating this unit? | Teaching experience *(Years, continuous)* |
| | Do you know how to program?<br>*(Yes / Learning for the first time / No)* | Do you know how to program?<br>*(Yes / No)* |
| *Participants that know programming* *(not mandatory)* | Where did you learn to program? *(multiple choice)*<br>Which languages can you use? *(multiple choice)* | |
| | *Case studies only:*<br>How would you rate your knowledge?<br>*(Basic / Reasonable / Proficient)* | Do you teach programming during the present academic year?<br>*In case of a positive answer:* Which languages do you teach?<br>How do you reason teaching arts and design students to code?<br>*(open text question)* |
| *Participants that know or are learning to program* *(indicative mood items)* | Perceived voluntariness of use<br>*("Programming has been entirely mandatory – voluntary," 7-point scale)*<br>Computer programming use frequency<br>*(Intervals: never, less than weekly, weekly, times a week, daily, times a day)*<br>Computer programming use intensity<br>*("I use programming as little – as much as possible," 7-point scale)* | |

In the specific contexts of the surveys applied to students in our case studies, we made a few modifications to the mandatory questions. The participants reporting a prior knowledge of programming were asked to rate said knowledge as basic, reasonable, or proficient. In addition, participants were asked whether they were attending the study's specific curricular unit for the first time.

### 4.1.2 TRANSLATION AND PILOT TESTS

Given the two different populations—students and professors—which we intended to survey and the need to account for participants having yet no contact with computer programming, we thus wrote four distinct versions of the survey instrument. We wrote our questionnaires in Portuguese, which was to be the application language, and we thus applied two procedures to verify our translation against the original items and to test our measures' applicability to our research contexts.

FIRST, we employed the services of a professional translator to retrovert our questionnaire variants back into English. After comparing these reverse translations against the original items, we consulted with the translator in making the necessary adjustments to the Portuguese questionnaires to eliminate discrepancies.

SECOND, following the advice of Creswell (2013) on pilot testing, we consulted with groups of students and professors to verify they found the language in their respective questionnaires to be clear and that these groups shared our understanding of the meaning of each item. The researcher organized a focus group with three students at the Music Composition course of the Porto Polytechnic Institute and two students at the Arts & Intermedia course of ESAP (Escola Superior Artística do Porto), a private higher education institution. These participants were given fifteen minutes to read hardcopies of the questionnaires and write down any comments and difficulties, and then each item was discussed regarding its clarity and what students understood. Concerning the professors' survey, the researcher contacted multiple docents at ESAP, which offers arts and design courses that include programming units. Though failing to organize a timely focus group, we managed to obtain four professors' comments on their specific questionnaires via email. We note that we asked the students and the professors we consulted to abstain from completing the survey, should they come across a request to do so.

## 4.2 *Sample and data collection procedures*

As explained in section §3, we applied our quantitative survey instrument in two distinct research phases. We first employed it to conduct preliminary studies of the acceptance of computer programming of students and professors' taking part in Portuguese arts and design undergraduate courses. In our second research phase, we applied a similar instrument to longitudinal panels of students participating in our case studies. Given the embeddedness of quantitative research within our case studies, we shall describe that procedure in the next section (see §5.1.5).

### 4.2.1 PRELIMINARY SURVEY

After our first cursory review of the curricular unit plans of Portuguese public arts and design undergraduate studies, we identified 50 courses that we either confirmed or found likely to include computer programming in their curricula, as described in 4.2. In doing so, we had collected the publicly available contacts of the courses' coordinators and host institutions' academic management. As to conduct our preliminary survey, we made initial contact with each course's coordinator via email in early March 2018. Upon request or lacking a timely response, we contacted institutions' academic management bodies to request their permission and their aid in forwarding the survey to the courses' students and academic staff. Several institutions required assurance of compliance with the EU's General Data Protection Regulation (GDPR) and of ensuring participants' anonymity in writing, in which case we complied.

We prepared two online surveys using Google Forms, one for students and the other for professors. In doing so, we leveraged the tool's capability to program the survey according to participants' answers as to follow the structure we detailed in §4.1.1. As shown in Figure 5.4.1, we placed links to either survey in a webpage that explained our research and our privacy policy (also presented in the survey forms). The site allowed us to share one single web address in all of our communications about the survey.

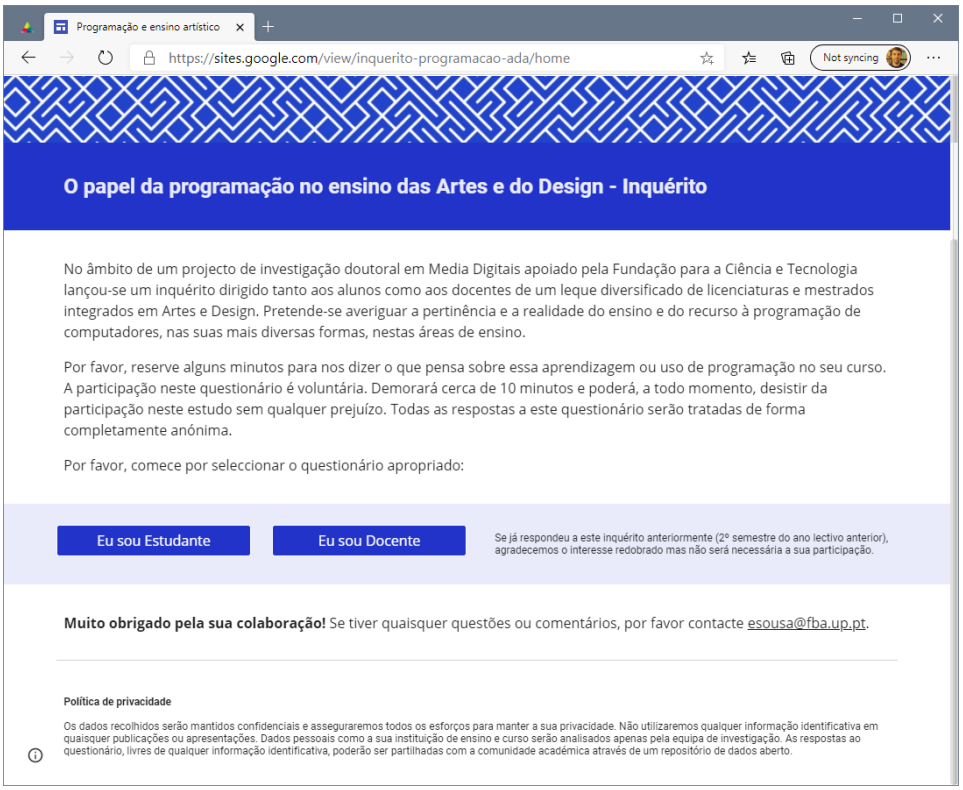



***Figure 5.4.1.*** *Survey research presentation and privacy policy webpage.*

Reliance on each individual course or host institution to disseminate our surveys led us to receive responses at a staggered pace. After renewed contact attempts with nonrespondent courses and academic institutions in early April, we kept collecting responses until early May, setting the online forms not to accept further responses after not receiving any for a week. In a two-month period, we obtained 68 responses from professors and 242 responses from students at sixteen different institutions. Given the dissemination method, many participants were outside of the desired population (e.g., graduate students or attending licentiates unrelated to our research). After excluding those, we were left with 187 valid student participations.

In early September 2018, coinciding with the beginning of a new academic year, we again set our online forms as open for participation and issued renewed requests to course coordinators and academic bodies from which we had not obtained any response during the first survey period. Hence, and until the end of October 2018, we obtained an additional 12 responses from professors and 102 responses from students (83 of which valid) at seven different institutions. In total, we obtained 270 valid participations from students at 27 different courses and from 80 professors at eighteen different institutions. Out of 165 we sent, 132 email messages sat without a reply, which testifies to a systemic issue with institutions' accessibility to educational researchers.

### 4.2.2 CASE STUDIES' SURVEYS

As noted, refer to §5.1.5.

## 4.3 *Data analysis procedure and validation*

After conducting either the preliminary surveys or the case studies' surveys of student acceptance, all quantitative data we obtained was inserted and coded in the IBM SPSS 24 software package for analysis. We verified the internal reliability and consistency of the item groups corresponding to each measurement construct in our acceptance and use model, following the advice of the statistics specialists we consulted. We made separate reliability tests for each survey we conducted.

Following the suggestions such analyses are highly specific and measurements are variant in regard to engagement with the focus technology (Parameswaran et al., 2015; Taber, 2014, 2018), we conducted separate tests for each such group of responses. We set the threshold for constructs' reliability at Cronbach α>.7 and having all items yielding a Corrected Item-Total Correlation or a Principal Component Analysis (PCA) extraction over .5 (Nunnally, 1978). In the next subdivisions, we will present the outcomes of such validations of measurements obtained from our preliminary surveys. The validation of the quantitative surveys embedded in our case studies will be discussed in §5.3.

### 4.3.1 STUDENTS' PRELIMINARY SURVEY VALIDATION

In Table 5.4.4, we present the results of our reliability and consistency tests of the valid responses (*n*=270) to our preliminary acceptance and use questionnaire, after exclusion of participations from outside the target student population. As formulated initially, several constructs were either unreliable or inconsistent.

***Table 5.4.4.*** *Students' questionnaire item reliability and consistency.*

| *Item groups* | *No knowledge* (*n*=72) | | *Learning* (*n*=77) | | *Prior knowledge* (*n*=121) | | *Computed construct* |
|---|---|---|---|---|---|---|---|
| | Cronbach α | Extract. ≥ | Cronbach α | Extract. ≥ | Cronbach α | Extract. ≥ | |
| PE1 PE2 PE3 PE4 | .85 | .47 [a] | .89 | .66 | .89 | .64 | *Performance Exp.* |
| EE1 EE2 EE3 EE4 | .86 | .50 | .91 | .71 | .93 | .77 | *Effort Expectancy* |
| At1 At2 At3 At4 | .93 | .78 | .92 | .66 | .94 | .77 | *Attitude to Use* |
| *SI1 SI2 SI3 SI4* | *.77* | *.34* | *.75* | *b* | *.85* | *.62* | – |
| SI1 SI2 - - | .95 | .95 | .95 | .95 | .90 | .91 | *Subjective Norm* |
| *FC1 FC2 FC3 FC4* | - | - | *.56* | *b* | *.70* | *.03* | – |
| FC1 FC2 - FC4 | .73 | .59 | .78 | .56 | .87 | .76 | *Fac. Conditions* |
| SE1 SE2 SE3 SE4 | .74 | .46 [a] | .80 | .45 [a] | .81 | .51 | *Self-Efficacy* |
| *An1 An2 An3 An4* | *.80* | *.55* | *.70* | *.33* | *.85* | *.61* | - |
| - An2 An3 An4 | .74 | .65 | .72 | .52 | .82 | .61 | *Anxiety* |
| BI1 BI2 BI3 | .91 | .84 | .91 | .84 | .93 | .82 | *Behav. Intention* |

*Notes:* [a] Items with the least extractions still yielded Corrected Item-Total Correlations exceeding .5.

[b] Factor analysis yielded two components.

*Social Influence* was inconsistent among respondents without programming knowledge and first-time learners: among the former, the question of whether *"my school would support learning and using programming"* (SI4) yielded poor extraction; among the latter, PCA yielded two distinct factors (SI1+SI2 and SI3+SI4). Items SI1 and SI2 correspond exactly to *Subjective Norm*, defined as *"the person's perception of the social pressures put on them to perform the behavior in question"* in Ajzen's Theory of Planned Behavior (1985). Given the similarity to the definition of *Social Influence* and that two items suffice to compute a latent construct (Steenkamp & Baumgartner, 1998), we substituted *Subjective Norm* for *Social Influence* in subsequent* analyses, as did several other UTAUT studies reviewed by Williams and colleagues (2015).

The *Facilitating Conditions* item about compatibility (FC3) was problematic when applied. Therefore, the construct was computed from the other three items. The *Anxiety* item regarding *"apprehension"* (An1) was problematic as well, in particular among first-time learners, and was also removed from further calculation. Our subsequent analyses were underpinned by the model we illustrate in Figure 5.4.2:

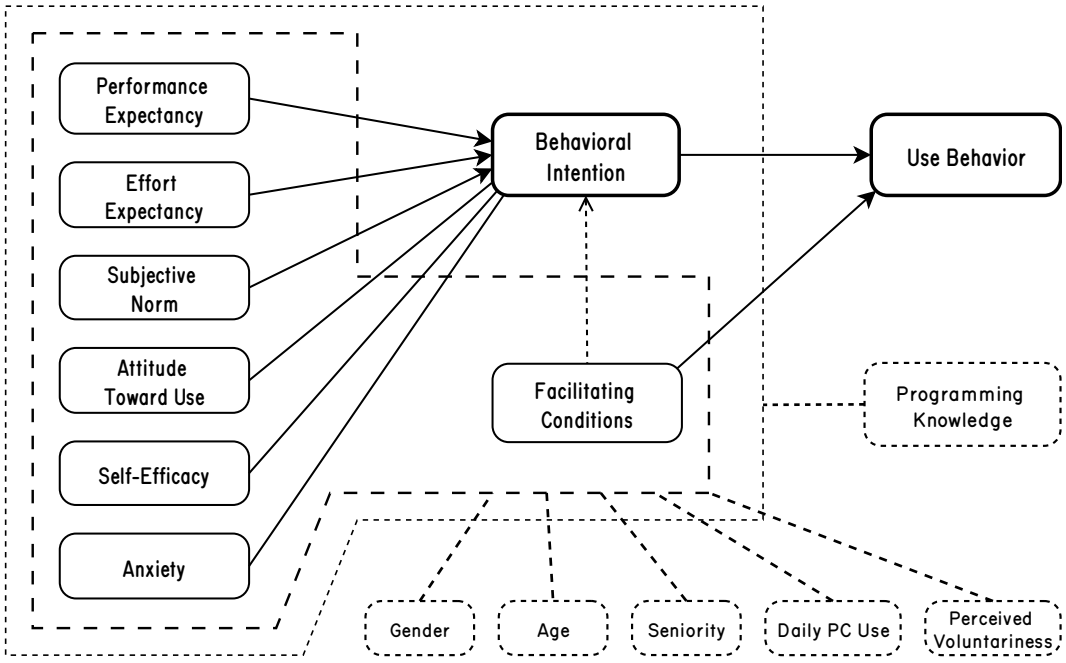



***Figure 5.4.2.*** *Our qualitative research model of students' acceptance and use of computer programming, after UTAUT (Venkatesh et al., 2003).*

Participants' programming knowledge configures distinct contexts for analysis, as suggested by Parameswaran et al (2015). Arrows denote the unidirectional relationships posited by the UTAUT (with *Subjective Norm* in place of *Social Influence*), whereas solid lines denote additional associations we wished to assess. Dashes denote the moderating effects to investigate. We accepted the model's original configuration, as the sample did not meet the threshold for an adequate interrogation of the model using structural equation modeling; Weston and Gore (2006) recommend at least 10 to 20 participants per each measurement and estimation item.

* As to maintain parsimony, we also employed *Subjective Norm* as a substitute for *Social Influence* in our analysis of responses from participants with prior programming knowledge.

### 4.3.2 PROFESSORS' PRELIMINARY SURVEY VALIDATION

Our reliability and consistency tests of professors' responses (*n=80*) to our acceptance and intention questionnaire yielded some similar outcomes to the validation of students' responses. We present reliability and consistency test results in Table 5.4.5:

***Table 5.4.5.*** *Professors' questionnaire item groups reliability and consistency.*

| *Item groups* | | | | *No knowledge* (*n*=50) | | *Prior knowledge* (*n*=30) | | *Computed construct* |
|---|---|---|---|---|---|---|---|---|
| | | | | Cronbach α | Extractions ≥ | Cronbach α | Extractions ≥ | |
| PE1 | PE2 | PE3 | PE4 | .87 | .49 [a] | .92 | .60 | Performance Expectancy |
| EE1 | EE2 | EE3 | EE4 | .90 | .58 | .92 | .65 | Effort Expectancy |
| At1 | At2 | At3 | At4 | .96 | .89 | .94 | .77 | Attitude Toward Use |
| *SI1* | *SI2* | *SI3* | *SI4* | *.70* | *b* | *.84* | *B* | – |
| SI1 | SI2 | - | - | .79 | .83 | .93 | .93 | Subjective Norm |
| *FC1* | *FC2* | *FC3* | *FC4* | - | - | *.61* | *b* | – |
| *FC1* | *FC2* | - | *FC4* | *.68* | *.46* | *.66* | *.54* | – |
| *SE1* | *SE2* | *SE3* | *SE4* | *.89* | *.68* | *.80* | *.37* | – |
| SE1 | - | SE3 | SE4 | .82 | .65 | .85 | .76 | Self-Efficacy |
| *An1* | *An2* | *An3* | *An4* | *.78* | *.41* | *.64* | *.20* | – |
| - | An2 | An3 | An4 | .80 | .67 | .72 | .61 | Anxiety |
| BI1 | BI2 | BI3 | | .98 | .95 | .95 | .86 | Behavioral Intention |

*Notes:* [a] Items with the least extractions still yielded Corrected Item-Total Correlations exceeding .5.

[b] Factor analysis yielded two components.

*Social Influence* was inconsistent as PCA yielded two distinct factors (SI1+SI2 and SI3+SI4) in both sets of responses. We substituted *Subjective Norm* (SI1+SI2) for *Social Influence* in proceeding with our analyses, given the similarity in both constructs' definitions, as explained in §4.3.1. Similarly, the item concerning whether participants felt *"apprehensive about computer programming"* (An1) was also revealed inconsistent within the *Anxiety* group in both questionnaire forms, and the measurement was therefore computed from the three other items.

As in the preliminary survey of students' acceptance, PCA yielded the *Facilitating Conditions* item about compatibility (FC3) as a separate factor in professors' responses to the indicative mood questionnaire. Yet, reliability and consistency tests after removal of FC3 did not meet our threshold; hence we did not compute *Facilitating Conditions* nor include the construct in our subsequent analyses.

Principal Component Analysis of the *Self-Efficacy* group revealed inconsistency with the item about whether respondents *"can complete a task if [they] can ask someone for help"* (SE2) among participants familiar with programming. Therefore, *Self-Efficacy* was computed from the remaining items, which yielded good reliability and consistency in our subsequent analyses.

Our subsequent analyses of professors' surveys were underpinned by the model we present in Figure 5.4.3. As in our analyses of students' surveys, we followed Parameswaran and colleagues' (2015) suggestion that participants' familiarity with programming configures distinct contexts for analysis.

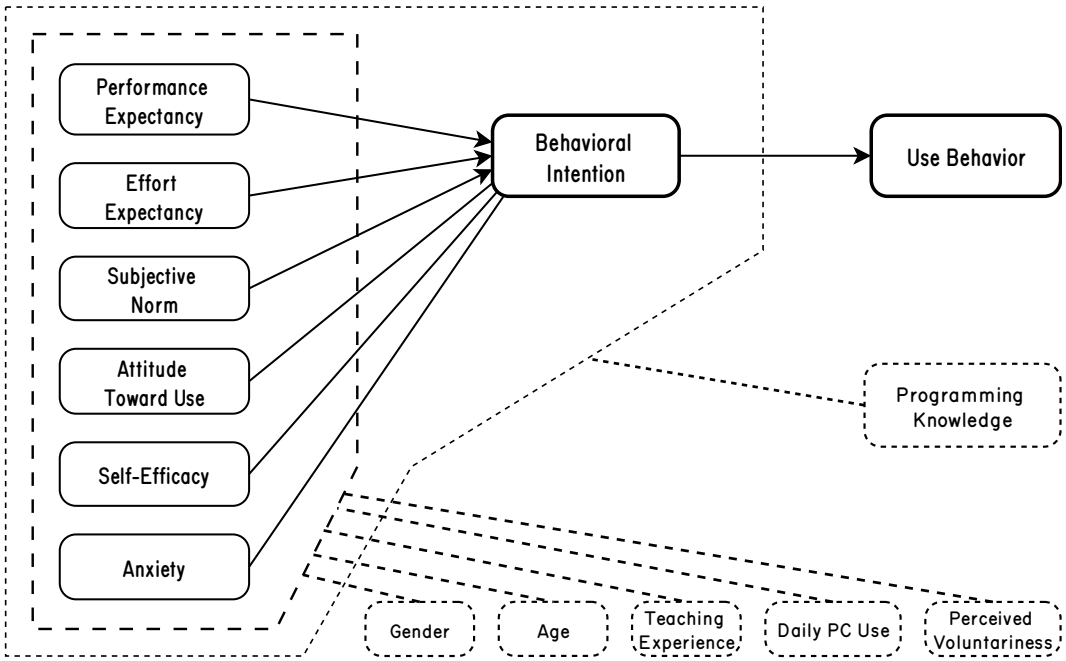



***Figure 5.4.3.*** *Our qualitative research model of professors' acceptance and use of computer programming, after UTAUT.*

Solid arrows denote the original unidirectional relationships posited by the UTAUT (Venkatesh et al., 2003), though with *Subjective Norm* in place of *Social Influence* and removal of *Facilitating Conditions* given the inconsistency found. Solid lines denote the possible additional associations we desired to evaluate, whereas dashes denote the moderating effects to investigate.

### 4.3.3 CASE STUDIES' SURVEYS

As noted, refer to §5.3.

# 5 CASE STUDY RESEARCH

As we described in §3.2, we conducted case studies to scrutinize the activities and attitudes of professors and students teaching and learning computer programming in undergraduate arts and design courses, as well as to interrogate and further the findings from our preliminary quantitative surveys. In the first subsection, we shall describe our sample and the studies' data gathering procedures, and in subsection §5.2, we will describe the analysis procedure for the qualitative data. As explained, we embedded quantitative surveys of students' views of computer programming within our case studies; we will describe their data validation and analysis procedures in §5.3.

## 5.1 *Sample and data collection procedures*

The experiences we had conducting our preliminary surveys, described in §4.2.1, shaped our approach to qualitative sampling for our case studies. We looked at the 27 courses from where we obtained responses and had opened a channel of communication with the institutional gatekeepers and reexamined their curricula. We balanced the potential relevance of each possible case study against its feasibility in the face of the available resources. Therefore, we deprioritized contacting institutions where observation would require either time-intensive or onerous travel to a location far from any other potential case-study where observation would also take place.

One takeaway from our panorama of the programming education offered to arts and design undergraduate students (see 4.2), is that we would have great difficulty identifying a 'typical case.' There is high variability in how apparently similar courses (e.g., in music or communication design) integrate computer programming in their curricula, to say nothing of the diversity of subfields within arts and design. Hence, we saw what Creswell (2013) describes as a *maximal variation sampling* of cases as inevitable. Furthermore, though the author sees maximal variation as just one among several sampling approaches in case study research, Aires and Merkens see such sampling as desirable (Aires, 2011, p. 22; Merkens, 2004, p. 167). We believe it is possible *theoretical sampling* could be desirable. Still, investigating cases until saturation would be unpragmatic given the limited time and resources.

To carry out the case studies, we began by leveraging the contacts made as part of our efforts to conduct the preliminary survey, as described in §4.2.1. Few institutions, courses, and professors were willing to host a case study during the 2018/2019 academic year and grant the necessary authorizations. As noted, several institutions were rather sensitive to privacy issues in the wake of the implementation of the EU's General Data Protection Regulation (GDPR), though we assured anonymity in collecting data related to students and committed to maintaining participants' anonymity when reporting our findings. Hence, given our limited resources and gatekeepers' conditions, our sampling approach may be described as *opinionated* (Aires, 2011; see also Merkens, 2004). We present a summary of the studies we conducted in Table 5.5.1, along with our chosen codewords for each curricular unit and host institution, following our commitment regarding the participants' anonymity.

***Table 5.5.1.*** *Overview of our case studies.*

| *Unit code* | *Host institution* | *Course area* | *Level – topic* | *Year* | *Participants* |
|---|---|---|---|---|---|
| PRIMER | RED (at a university) | Multimedia Production | Introduction | 1st - 3rd * (mandatory) | 180 students / 4 classes 3 professors |
| GRAPHIC | RED (at a university) | Design | Introduction – graphics | 1st - 3rd (elective) | 15 students (elective) 1 professor |
| SKETCH | LIME (at a university) | Design | Introduction | 2nd | 61 students / 2 classes 1 professor |
| WEAVE | BLUE (at a university) | Multimedia Production | Intermediate – web development | 2nd | 91 students / 4 classes 2 professors |
| TOUCH | ROSE (at a polytechnic) | Multimedia Production | Intermediate – tangible interfaces | 3rd | 19 students 1 professor |
| FLOW | AMBER (at a polytechnic) | Music | Introduction – dataflow coding | 1st | 5 students 1 professor |

There were two additional cases that we decided not to carry out, given the conditions proposed by the respective institutions: one explicitly refused observation, and another conditioned the case study on hosting the research project itself—a demand which was incompatible with the ongoing doctoral investigation.

As it can be surmised from Table 5.5.1, our six instrumental case studies focused on a diverse group of curricular units from six distinct licentiate courses at five different host institutions, which we feel present a good panorama of how programming is taught and learned in arts and design undergraduate courses. Three of the host institutions were faculties integrated into larger universities, and two others were schools that were part of a polytechnic institute. We also anonymized the names of the licentiates; three were mostly dedicated to audiovisual and multimedia production, two were dedicated to communication design, and one was dedicated to music.

We followed the same general blueprint for data collection in each case study, which we shall describe in greater detail in the coming pages: FIRST, we obtained documentation about the host institution, the course, and the curricular unit; SECOND, we requested guest access to online supporting resources, such as a learning management system (LMS); THIRD, we requested to attend sessions and observe the classroom; FOURTH, we asked professors to request their students to complete our quantitative surveys both at the start and at the end of the semester; finally, FIFTH, we requested individual interviews with each professor at the end of the term in which the study took place. In addition, though we had not planned for it given the anonymity and privacy issues that could arise, we were given opportunities to study students' exercises and projects, either by being invited to attend exhibitions or presentations or by being forwarded students' work submissions.

In Table 5.5.2, we present a summary of the data we collected in our case studies.

* The curricular unit coded as PRIMER was mandatory but attended by students in either year of their course under a transitional curriculum (see 7.2).

***Table 5.5.2.*** *Overview of the data we collected in each case study.*

| *Case* | *Session Observations* | *Online support & resources* | *Students' exercises* | *Student surveys* | *Professor interviews* |
|---|---|---|---|---|---|
| PRIMER | 13.5 hours – weeks 7 & 9 | Moodle LMS *(limited access)* | Presentation session (6 hrs.) | 76 p. – wk. 4<br>10 p. – wk. 15 | 2 via Skype,<br>1 via email |
| GRAPHIC | 6 hours – weeks 8 & 10 | Shared folder | Prof. shared submissions | 6 participants *(discarded)* | Via Skype |
| SKETCH | 18 hours – weeks 2, 7, 9, 12 | Moodle LMS, weblog, video | Presentation session (6 hrs.) | 51 p. – wk. 2<br>17 p. – wk. 15 | Face-to-face |
| WEAVE | 8 hours – weeks 10 & 13 | Professor's website | *No access* | 50 p. – wk. 3<br>12 p. – wk. 15 | 2 via email<br>(1 declined) |
| TOUCH | 4 hours – week 6 | Shared folder *(limited access)* | Exhibition, Present. (4 hrs.) | 17 participants – week 6 | Face-to-face |
| FLOW | 10 hours – wks. 1, 6, 10, 13, 18 | Professor's website | *No access* | 5 p. – wk. 1<br>4 p. – wk. 18 | Face-to-face |

Each study presented its specific constraints, either set by the institution, the course, the professors, or else due to operational limitations such as delays or scheduling conflicts (GRAPHIC, SKETCH, TOUCH, and FLOW took place at distinct locations in the same semester).

We note the units we refer to as PRIMER and SKETCH were attended by many students, which were divided into separate classes. During our visits to the respective faculties or schools, we sometimes had the opportunity to attend multiple sessions in the same week. In the case we refer to as WEAVE, we were able to attend distinct theoretical lectures and practical studio sessions during each visit to the respective faculty. We did not include the project presentations we attended in our tally of observed sessions—i.e., we made three distinct visits related to the TOUCH study, though we only attended one regular session in the studio, for we also attended a public exhibition of students' intermediate projects and a session dedicated to students' formal presentation of their final projects.

In the next subdivisions, we will describe in further detail the gathering procedures for each type of data we collected while carrying our collective case study.

### 5.1.1 DOCUMENTATION AND ONLINE CONTENT

We gathered public records relating to the courses and the relevant curricular units in order to establish the panorama, reported in chapter **4**, of programming education and use within Portuguese public arts and design undergraduate courses. Selection of which institutional contacts to perform as to conduct the case studies had followed from our analysis of those documents.

In performing such contacts, we requested gatekeepers, most often the courses' academic coordinators, to provide us with updated documents, notably the curricular unit plans (often not available until after the start of the academic year) as to ascertain each potential case study's relevance. We thus gathered the following public documents relating to each specific case's curricular unit:

- Website texts or brochures about the courses and their host institutions.
- The course's curricular plans.
- Government statistics about the courses (e.g., DGEEC, 2019).
- The curricular units' plans (CUPs), which state each unit's learning objectives, syllabus, teaching and evaluation methodology, and bibliography.
- Historical course plans and CUPs, obtained in host institutions' websites, in the Government Gazette—dre.pt—or in filings with the Accreditation Agency—a3es.pt.

When put in contact with the professor responsible for the curricular unit that would be under scrutiny, we requested its schedule and lesson plan so we could prepare our visits. We also requested any other materials the professor felt pertinent to share. We also inquired whether the professors would be using a learning management system (LMS) or any other online platform for sharing educational resources and whether we could be given guest access. Though the level of access we were granted was highly variable, we gathered or referenced the following types of 'private' documents and artifacts over the six case studies:

- Schedules, lesson plans, and summaries.
- Lecture slides and lecture notes.
- Recorded video lessons.
- Exercise or project instructions.
- Notices and LMS forum posts made by the professors.
- Blog posts by the professors.
- Project examples (e.g., made by students at a previous instance of the curricular unit).
- Exercises submitted by students (see §5.1.3).

As Merriam and Tisdell (2016) note, online content may count as an observation rather than as a document (p. 159); video lessons, blog posts, and forum discussions are basic e-Learning tools, after all. Furthermore, the authors rightly point out online observation allows for the researcher to be truly invisible. Besides, certain types of resources, such as the lecture slides and notes authored by the curricular units' professors, may be observed as unidirectional interactions between the professors and their students and were appreciated as such.

### 5.1.2 CLASSROOM OBSERVATION

According to Aires (2011), *"qualitative observation is fundamentally naturalistic; practiced in the context of the occurrences between actors interacting naturally"* (p. 25). The author values observation for *"avoiding the artificial distortion of the experiment and the 'numbing' dimension of the interview"* (p. 27), though also warning about potential pitfalls stemming from the researcher's own biases (discussed in sections §1.1 and §3.2), being a potential influence in the group under observation, or developing emotions toward the group that cloud critical abilities. As to wean concerns about the validity of observation, Aires recommends triangulation analyses (as described in the next subsection) and careful and precise description of the observed situations.

Our visits to attend sessions of the curricular units under scrutiny were scheduled with the agreement of the responsible professors and after examination of their lesson

plans. As can be surmised by Table 5.5.2, we faced a few initial delays given the aforementioned reticence of institutions and professors regarding our commitment to respecting participants' privacy and ensuring their anonymity. In some cases, agreement to observation required further negotiation. The professor responsible for the WEAVE study asked that our observation was limited to the sessions he led, and thus we were not authorized to visit the classes led by his assistants. One professor negotiated access to the raw data from our quantitative survey of his students, which we facilitated after redacting any information (e.g., students' age) that could permit identifying particular respondents.

Given his experience with that approach while working in documentaries or video reporting, the researcher took the role of a nonparticipant observer—*"an 'outsider' who sits on the periphery or some advantageous place (e.g., the back of the classroom) to watch and record the phenomenon under study"* (Creswell, 2013, p. 215). The researcher arrived at the location of each session prior to its start, and the professors leading the session were asked to briefly present the researcher to the class prior to beginning their lecture or workshop session. On his first visit to each location, the researcher asked to address the class for a moment to state the aims of the study and to thank the professor and the students for their contribution. The researcher would sit down on the seat he had chosen beforehand among the students' (placing his coat in the chair before students entered the room), which he had judged to grant a good view of the room or, in practical workshop sessions, a good position from where to stand and move around the room.

Though constrained by time, the non-participating role allowed the researcher to grow invisible through familiarity, as to enable closer observation of practical work and of students' interactions with the professor and among themselves. After taking a few minutes to write down initial notes about the classroom conditions and, in initial visits, to measure and draw sketches of the surrounding environment discreetly, the researcher tentatively moved around: first, to pace around the room and catch glimpses of students' computer screens, and later, to get closer to the professor and listen to their conversations with students. Crucially, notes would be written down as immediately as possible but at a discreet remove from the participants. Like a documentarist holding a wide-angle camera by the chest and away from the face, the researcher meant to present himself as an interested witness, not as a reporter.

Following the advice of Creswell (2013), the researcher wrote down descriptive and reflexive field notes. In addition, upon leaving each site of observation, the researcher wrote down a personal reflection about the session just witnessed as a first step toward finding the emergent themes from observation.

### 5.1.3 STUDENTS' PROJECTS AND EXERCISES

Though not a part of our original design due to concerns on how to assure the anonymity of the authors, we were able to observe the artifacts produced by students in the context of public exhibitions (as in the TOUCH study) and formal presentation sessions which the responsible professors invited us to attend. We wrote down notes to document the artifacts. Though students' work responded to professors' instructions, often these gave

students great latitude. Hence it was valuable to employ our notes to find emergent categories of artifacts.

In the particular study we refer to as GRAPHIC, the responsible professor gave us access to a shared online folder where students submitted the source code for two exercises. This gave us access to two types of artifacts produced by the students: the code students produced lent itself to description and analysis, as the researcher could not overlook his expertise in the programming language employed. The researcher was also able to run the code and gather the resulting visual artifacts. Nonetheless, in respect to students' anonymity, no efforts were made to link any particular exercise to any other datum in a way that could undermine said anonymity.

### 5.1.4 INTERVIEWS WITH THE PROFESSORS

Aires (2011) tells us the interview is *"an artificial process in which the researcher creates a specific situation"* which *"is far from a spontaneous social exchange"* (p. 29). However, the author adds, interviews comprise an ongoing *"interaction that creates and captures meaning and which is decisively influenced by the interviewer and interviewee's personal characteristics"* (p. 29) and configure a process of socialization. Hence, though the minimal interview situation arises from the researcher's objectives and their preliminary interview protocol or structure, *"it is desirable that throughout the interview [researcher and interviewee] create a dynamic relationship that generates the themes being studied"* (p. 33). The author concludes, *"therefore, the interview is not an interrogation situation, but rather a 'confession' situation"* (p. 33). Agreeing that interviews are generative and not merely data gathering situations, Cousin (2009) defends the interactionism of Miller and Glassner* in granting researchers the authority to look beyond meanings generated at the site of the interview in their shaping and interpretation of the interview (p. 78).

We asked the professors taking part in our case studies to grant us one-on-one interviews at the end of the respective terms. Ideally, we would have scheduled face-to-face interviews with all participants, yet some were reluctant. Of the ten professors involved in the curricular units corresponding to our cases, three agreed to schedule a face-to-face interview, and another three agreed to a Skype call. Three professors (one assistant in the PRIMER study, and the lead professor and one assistant in the WEAVE study) only made themselves available to answer questions via email; another assistant at WEAVE declined to respond to any questions.

We devised a semi-structured interview protocol; questions were open-ended and shaped by our experiences observing sessions during the preceding weeks and by the outcomes of our preliminary study. Our main questions were based on the following:

- How do you look back at the semester, in regard to students' meeting your objectives for the curricular unit?
- How would you describe students' commitment to learning programming? What were their strengths and weaknesses doing so?

* Miller, J. & Glassner, B. (1997). The 'inside' and the 'outside': finding realities in interviews. In D. Silverman (ed.), *Qualitative Research: theory, method and practice* (pp. 99–112). Sage.

- How do you value learning to code? How do you see programming regarding digital literacy?
- Since you started to teach the subject, were there any changes in students' interest in computer programming? And what changes did you see in students' learning autonomy?
- Is it usual for students to display fear or anxiety towards learning to code? If so, how do you deal with that?
- Do you think students of different genders or age groups display different attitudes toward programming? How do you explain those differences?

Though we posed the above questions in each specific interview (see Annexes), we did so in a flexible manner and adapted to what we witnessed in each specific case. There were also significant and expected differences between the synchronous interviews and the email interviews.

✦ FACE-TO-FACE AND SKYPE INTERVIEWS We obtained the explicit consent of participants to record the interviews, whether using an audio recorder—in face-to-face interviews—or using Skype's recording feature in those interviews conducted over a call. After getting their prior agreement to the procedure, we began our interviews by taping participants answering affirmatively to a question on whether we had their consent to record the conversation. In addition to the audio recording, we followed the advice of Creswell (2013) and wrote down notes during the interviews, either to shape the ongoing interview or to aid analysis by noting the emergent themes and noteworthy observations made by the interviewees.

Though there was a familiarity with the interviewees, which had been established during our visits to observe their classrooms (e.g., in conversations during breaks or over lunch prior to or after a session), we followed Cousin's (2009) advice about building rapport (p. 76). We began our interviews with a few questions devised to put participants at ease, asking them to describe their earliest recollections of contact with computers and engaging them in a nostalgic dialogue about the computing technology of the time. We then guided the conversation into questions about their professional and academic backgrounds and how they related to computing.

In our interview with the professor responsible for TOUCH, we were given a time limit and therefore had to be mindful of gathering answers to our core questions. Apart from that specific interview, we probed participants to elongate their answers, only interrupting and asking to move to another question if we felt they were repeating themselves. We consulted our interview notes to pose requests for clarification or follow-up questions that could have arisen from the professor's statements.

✦ EMAIL INTERVIEWS We sent an email with our questions tailored to each of the three participants who desired to be interviewed in this manner. We were careful to write the questions as clearly open-ended to elicit richer responses, and in our first email, we also asked participants whether they would be available to respond to follow-up questions. Nonetheless, email is a much poorer interview medium. Given the time participants took to respond, we were not able to take these interviews beyond the aforementioned 'minimal situation' and elicit deeper themes.

### 5.1.5 QUANTITATIVE SURVEYS OF STUDENTS

We employed the instrument described in §4.1 to carry out longitudinal surveys of the acceptance and use of programming by students attending each of the curricular units corresponding to our cases. As we did for the preliminary surveys, we prepared a Google Form for each survey application, which we likewise prepared to follow the structure described in §4.1.1 depending on students' answers to whether they had prior familiarity with programming or were learning to code for the first time.

As to investigate possible changes in students' acceptance and use, we intended to apply the survey instrument twice during each case study, both near the beginning and at the end of the corresponding term. In complying with the requests for discretion and anonymity made by host institutions or the professors responsible for each individual class, completion of the surveys was made voluntary. We also devised a mechanism to match each participant's set of responses without asking for identifying information: we added a numeric field to our online surveys so participants could provide a five-digit sequence of their own choosing* while completing the first application; participants were asked to provide the same sequence of digits in the second survey application.

We were dependent on each curricular unit's professors to require their students to complete the surveys. We provided each professor the appropriate link to our online surveys at the start of the semester, yet their initial application and exact procedure varied on a case-by-case basis. We faced delays in obtaining the proper authorizations to proceed with our studies of the curricular units we dubbed PRIMER and TOUCH. Some professors asked students to complete the surveys during a session, as witnessed while observing SKETCH or TOUCH classes; yet others preferred to forward a request to students' institutional email or posted a notice with the survey link in the unit's learning management platform (e.g., Moodle), which yielded a lower response rate.

Following our instructions, professors later requested students to complete the second survey application during the last week of classes. Again, such requests were made in class, as well as via email or by posting a notice to the curricular unit's LMS. Yet, given the voluntary nature of the surveys and the fact that students were either finishing or had just presented their projects, most declined to complete the second survey application, even when such request was made during sessions we observed, which was a disappointing outcome. A few additional students completed the second survey, yet we were unable to match their participation to the first survey (as they did not provide a matching digit sequence), and we thus excluded their responses. As shown in Table 5.5.3, we obtained an unsatisfactory level of valid student participation in most case studies' surveys.

* The survey form instructed respondents on the purpose of the sequence and asked them to memorize the digits as to complete the second survey.

***Table 5.5.3.*** *Overview of students' survey participation in our case studies.*

| *Unit code* | *Area / Institution code* | *Year* | *First application* | | *Second application* | |
|---|---|---|---|---|---|---|
| | | | *week* | *responses* | *week* | *responses* |
| PRIMER | Multimedia Production *at RED (university)* | 1st - 3rd *(mandatory)* | 4 | 76 (42%) | 15 | 10 (6%) |
| GRAPHIC | Design *at RED (university)* | 1st - 3rd *(elective)* | 8 | 6 (40%) | *(not conducted)* | |
| SKETCH | Design *at LIME (university)* | 2nd | 2 | 51 (83%) | 15 | 17 (28%) |
| WEAVE | Multimedia Production *at BLUE (university)* | 2nd | 3 | 50 (56%) | 15 | 12 (13%) |
| TOUCH | Multimedia Production *at ROSE (polytechnic)* | 3rd | 6 | 17 (89%) | *(not conducted)* | |
| FLOW | Music *at AMBER (polytechnic)* | 1st | 1 | 5 (100%) | 18 | 4 (80%) |

We disregarded the first survey application and opted to eschew a second application to students attending the unit we dubbed GRAPHIC, given their small number and their participation percentage. We also decided not to apply a follow-up survey to students attending TOUCH, given the lateness of the first survey application and the group's preexisting familiarity with programming acquired earlier in their curriculum.

## 5.2 *Qualitative data analysis*

According to Aires (2011), the flexibility of qualitative methodologies and their *"diversity of perspectives alongside the frequent paucity of recommendations and systematizations present the most challenging dimension of analysis"* (p. 43). Hence, at the earliest stages of case studies' observation and analysis, we held on to one of the possibilities laid out by Creswell (2013) regarding explanatory design analysis, that of *"typology development through quantitative data collection and the use of this typology as a framework for identifying themes in a qualitative database"* (p. 552). In doing so, we first looked at UTAUT's constructs (see §2.3) as possible themes in our qualitative observations.

However, we took care to approach the data inductively (cf. Cousin, 2009, pp. 35–36) to support a reflexive analysis. In so doing, we moved away from the categorization prompted by giving too much weight to the UTAUT and the questions prompted by the preliminary survey's results. We recognized qualitative data would respond to complementary research objectives from quantitative data and heeded Creswell's (2013) advice about embedded design analysis, keeping the analyses separate (p. 552).

We present in Table 5.5.4 a summary of the main emerging coding categories and subcategories we employed in our analysis of such qualitative data, also noting to whom, from which source, or when they were pertinent.

***Table 5.5.4.*** *Qualitative analyses main coding categories and subcategories.*

| *Category* | *Who* | *Applied to* | *Description* |
|---|---|---|---|
| *Curriculum* | P - | D - - - I | Considerations relating to the course's curricular plan and the role of the curricular unit in it. |
| *Objectives & value* | P - | D L - - I | Considerations regarding the value of learning to code for arts and design practitioners and students. |
| *Students & workgroups* | - S | D L T E I | Student's gender, age, course, seniority. In studio contexts, whether working individually or in a group. |
| *Professors* | P - | D - - - I | Relating to the professors' unit assignments, professional experience, or general individual preferences. |
| *Content* | P S | D L T E I | Relating to the content (e.g., how to use a programming language) taught in the curricular unit. |
| *Pedagogical strategies:* | P - | D L T - I | Relating to professors' teaching methods and general behavior in the classroom. |
| *Explanation & demonstration* | P - | - L T - I | How professors conduct their explanations and demonstrations. |
| *Advice & critique* | P - | - L T E I | Relating to professors' advice, critique, and recommendations to specific students. |
| *Evaluation* | P - | D - T E I | Relating to the unit's evaluation methods and requirements. |
| *Resources* | P - | D L T - I | Use, sharing, and recommendation of educational resources (e.g., bibliographies, tutorials, code examples). |
| *Conditions* | P S | D L T E I | Relating to the unit's working and facilitating conditions (e.g., classroom, equipment, schedule, office hours). |
| *Tutoring* | P S | - - T - I | Considerations about supervision (e.g., consultation time, whether the professor checked back later). |
| *Students' behavior:* | - S | - L T E I | Students' general classroom behavior (e.g., attendance, participation, attentiveness, diligence). |
| *Impact on students:* | - S | - L T E I | Relating to the impact of content, activities, and interventions of students' regard of programming. |
| *Interest* | - S | - L T E I | Relating to students' expressions of interest and motivation (i.e., *Performance Exp., B. Intention*). |
| *Efficacy* | - S | - L T E I | Relating to students' expressions of learning autonomy and *Self-Efficacy*. |
| *Difficulties* | - S | - L T E I | Relating to students' expressions of difficulty and self-criticism (i.e., *Effort Exp.*, *Anxiety*). |
| *Project & exercise characteristics:* | - S | D - T E I | Students' work general characteristics (e.g., requirements, freedom of topic, the general topic, tools used). |
| *Formal characteristics* | - S | D - T E I | Relating to formal exercise characteristics (e.g., graphics, audio, interaction, type of presentation). |
| *Incident* | P S | - L T E I | Related to noteworthy incidents (i.e., vignettes). |

*Legend:*
*S Student*
*P Professor*
*D Course / unit documentation & online resources*
*L Lectures*
*T Tutoring / studio supervision*
*I Interview*
*E Work exhibition or presentation*

We used the Atlas.ti 8 qualitative analysis software to create a database for each case study, organizing the field notes, reflections, interview transcripts, and documents we collected. In the first readings of our database, we assigned tentative categories and memoranda to the relevant blocks of text, which we then grouped and reduced into the emerging analysis categories above.

Following the example of a classic collective case study of high schools by Sara Lawrence-Lightfoot* cited by Merriam and Tisdell (2016, pp. 233–234), we too intended to present our findings as six intra-case 'portraits' followed by an inter-case analysis.

### 5.2.1 INTRA-CASE ANALYSIS

*Intra-case analysis*, as we shall present in chapter 7 (see sections 7.2 to 7.7), proceeds in two incremental stages, according to Aires (2011). The first stage is descriptive and concerned with the *what* and the *how* of the research focus, whereas the second stage is explanatory and concerned with the *why* (pp. 47–48). Therefore, understanding causality is a preoccupation enabled by the temporal nature of qualitative research. Hence, though we approached the causal associations in our quantitative model from a position of skepticism, we believe the analysis of the qualitative data may contribute to establishing causal explanations. Yet, tempered by an acknowledgment of the surrounding context of each case and the researcher's own positioning and prior educational experience, such explanations are not to be construed as linear.

Indeed, Aires (2011) quotes Miles and Huberman† to place causal multiplicity among the characteristics of qualitative analysis, noting multiple causes combine and interact not only among themselves but also with their effects; hence *"causes and effects ought to be configured as networks profoundly influenced by local contexts"* (p. 49); networks which we may envision as closer to Deleuze and Guattari's (1987) notion of the *rhizome* rather than as acyclic graphs.

Given the researcher's aforementioned predilections and positionality, particular care was taken to describe incidents and exemplary vignettes. Besides bringing some narrative 'color' to the systematic categorization of our field notes, we concur with Cousin's (2009) view of vignettes as offering important insights and good leads for interpretation (pp. 121-122).

### 5.2.2 INTER-CASE ANALYSIS

According to Aires (2011), *inter-case analysis,* which we shall present in 7.8 as to inform the discussion in chapter **8**, proceeds from intra-case analysis to address the challenge of *"articulating the specific with the universal, reconciliating one case's unique character with the need to understand general processes"* (p. 50). The same author presents criteria as to validate robust qualitative findings: *Credibility*, the *"isomorphism between reality and the information gathered,"* can be achieved through extensive observation and *triangulation*; *Consistency* as to researcher's gathering and analysis of information as a goal of detailed descriptions of the researcher's stance and the study procedures; and *transferability* of findings, achievable through theoretical sampling and extensive data collection and description (pp. 54–56).‡

---

* Lawrence-Lightfoot, S. (1983). *The Good High School.* Basic Books.

† Miles, M. & Huberman, A. (1994). *Qualitative Data Analysis (2nd ed.)*. London: Sage.

‡ A fourth criteria, *neutrality*, concerns the agreement between multiple researchers and was not appliable in our research.

The aforementioned limited resources available for our research impact the transferability of our findings; hence we ought to qualify our assertions. Nonetheless, we strove towards consistency and credibility in our research procedures and analysis. As to achieve the latter, we triangulated data sources and methods. Cousin (2009) quotes Hammersley and Atkinson* in advising against a naïve expectation that multiple data sources will unproblematically corroborate each other and in noting *"differences between sets or types of data may be just as important and illuminating"* (p. 115).

Kelle and Erzberger (2004) note results may be *convergent*, yet agreement ought to relate to the same set of facts; *complementary*, yet related under a joint theoretical framework; or *divergent*, which, relating to the same facts, testify to methodological or theoretical inadequacies and may motivate further theorizing and research (p. 174–176).

## 5.3 *Quantitative surveys' validation and analysis*

Though keeping present the outcomes of the validation of our preliminary survey of students' acceptance and use of programming, we attempted to conduct additional reliability and consistency tests for each group of responses (according to participants' programming knowledge) to the first application within each particular case study. Case studies' small sample sizes, presented in Table 5.5.3, rendered certain tests unfeasible; hence we discarded the responses of participants in the GRAPHIC and FLOW studies and of the six participants in the TOUCH study not familiar with programming.

We followed Yurdugül's (2008) suggestions for conducting reliability tests with small samples and were able to obtain meaningful indications. The fact we replicated two outcomes from our preliminary survey's reliability and consistency tests also gave a strong signal attesting to their trustworthiness. As then, the four *Social Influence* items factored into two latent constructs across all tests conducted; in addition, SI1 and SI2, corresponding to the *Subjective Norm* measurement, yielded good reliability across our tests ($\alpha > .8$). We also obtained similar outcomes when testing *Anxiety*'s reliability, in which the first item (*"apprehensive"*) yielded poor extraction (<.5) across our tests.

We also found differences in the reliability of our measures across the case studies. Though some may arise from the small samples, we note the specificity of each case study may contribute to differences in interpretation (Taber, 2014), as our observations found explanations for most unreliable items:

- PERFORMANCE EXPECTANCY Whether programming *"increases chances of scoring higher grades or earning more"* (PE4) presented a poor extraction (<.5) from groups of participants attending PRIMER and SKETCH; hence *Performance Expectancy* was computed from the remaining items. Perhaps the question lacked salience to participants in those cases, still far from completing their degrees and attending courses with few mandatory computer programming credits.
- EFFORT EXPECTANCY Whether *"interactions though a programming language would be clear"* (EE1) presented a low extraction value (<.5) among all groups of

* Hammersley, M. & Atkinson, P. (1983). *Ethnography Principles in Practice*. Routledge.

participants in SKETCH. In that case, *Effort Expectancy* was computed from the remaining items and proved reliable. We speculate the difference in interpretation may relate to early autonomous exploration after students attended an event with demonstrations and masterclasses in the week prior to completing the survey.

- FACILITATING CONDITIONS Testing of the responses of participants in the WEAVE study replicated the outcomes of the preliminary survey's tests: the item about compatibility (FC3) showed low extraction and was removed; the construct was then reliable ($\alpha$>.7; extractions >.6). Yet, *Facilitating Conditions* was unreliable among any other group of responses from the other cases. WEAVE students likely felt more qualified to assess conditions, given the combination of their familiarity with the course with their greater familiarity with code.
- SELF-EFFICACY Feeling able to work while solely having *"access to help files or the programming language's documentation"* (SE4) presented a low extraction (<.4) among participants in the WEAVE study; *Self-Efficacy* was computed from the remaining items in that case. Though this difference in interpretation may relate to certain students' past experience with programming in their course, we have no further clues.

But for *Facilitating Conditions*, which we were only able to measure reliably in WEAVE, we obtained reliable and consistent measurements (Cronbach $\alpha$>.7; extractions > .5) of the remaining constructs in our research model (refer to Figure 5.4.2) from our first survey application to participants at all levels of knowledge of programming within the PRIMER, SKETCH, and WEAVE studies, as well as from participants familiar with programming in the TOUCH study.

We note we did not test for the reliability of the responses obtained in our second survey application at the end of each respective semester. Given the smaller sample and the fact we matched response sets, we computed the means for each construct according to the outcomes of our validation of the first survey application.

### 5.3.1 STATISTICAL ANALYSES

After validation, we also employed the IBM SPSS 24 software package for all our statistical analyses. We followed the advice of the Statistics specialists we consulted with, as well as of literature about measures of association (cf. Goodman & Kruskal, 1979; Khamis, 2008; Richardson, 2011), to determine which statistical tests were appropriate to investigate the possible relationships between variables in our models, considering the sample size and the type of data tested for association.

Given that the curricular unit observed in the FLOW case study was attended by just five students, we not we did not calculate latent constructs or look for associations between variables. In that case, we looked for the specific items in which participants unanimously demonstrated either a negative or a positive change in their responses, as to infer some evolutions in the students' acceptance of computer programming.

*(blank page)*

PART THREE

# *FINDINGS (PRELIMINARY, CASE_STUDY);*

*(blank page)*

# 6

# *A preliminary survey of students and professors' acceptance and use of computer programming*

★ SYNOPSIS After our initial review of undergraduate arts and design courses offering contact with computer programming, which we expanded in chapter 4, we conducted preliminary surveys of those students and professors' acceptance and use of computer programming. We describe and present the results obtained from opportunity samples of 270 students (in section §2) and 80 professors (§3).

## 1 OVERVIEW

As presented in chapter 4, we had identified fifty arts and design undergraduate courses offered at Portuguese public higher education institutions during the 2017/18 academic year that potentially offered a contact with computer programming to their students.* As described in chapter 5, we thus designed questionnaires based on Venkatesh and colleagues' Unified Theory of Acceptance and Use of Technology (2003) to measure said acceptance and use of computer programming by students and professors involved in those courses. We relied on each course's academic coordinator or the respective faculties or schools' management bodies to authorize and forward our online surveys, as presented in 5.4.2. The application thus proceeded at an uneven pace. During two application periods—March to April and September to October 2018—we obtained 270 valid responses from students attending 27 different courses; we will report those findings in section §2. We also obtained 80 responses from professors at 18 institutions and will report those findings in section §3.

* As described in chapter 4, we subsequently determined that only 42 arts and design courses offered a real contact with programming in the 2018/19 academic year. We did so after a deeper analysis of the curricular unit plans, some of which were obtained with the help of the coordinators we had contacted for the purpose of conducting these preliminary studies.

## 1.1 *Communications related to these surveys*

At the time of writing, we presented three conference papers related to these preliminary surveys. We analyzed the data collected during the first period—March to early May 2018—and reported our interim findings at the 11th annual International Conference of Education, Research, and Innovation (ICERI). Both papers are indexed in the Web of Science Core Collection:

- Morais, E., Morais, C., & Paiva, J. C. (2018). The Perspective of Higher Arts and Design Educators on Teaching Computer Programming. In L. G. Chova, A. L. Martínez, & I. C. Torres (Eds.), *ICERI2018 Proceedings* (pp. 2043–2050). Seville: IATED.
- Morais, E., Morais, C., & Paiva, J. C. (2018). Computer Programming Acceptance By Students In Higher Arts And Design Education. In L. G. Chova, A. L. Martínez, & I. C. Torres (Eds.), *ICERI2018 Proceedings* (pp. 2058–2065). Seville: IATED.

We reanalyzed the data after adding the responses we received during the second period—September and October 2018—and reported our findings about students' acceptance at the 2019 International Conference on Education and New Developments:

- Morais, E., Morais, C., & Paiva, J. C. (2019). Higher Arts and Design Students' Attitudes Towards Learning Computer Programming. In M. Carmo (Ed.), *Education and New Developments 2019* (Vol. II, pp. 115–119). Porto: InScience Press.

We had to summarize the findings presented in the above papers to meet the conferences' publication requirements. In the next sections, we will explore the responses we obtained from survey participants we obtained in greater detail.

## 2 STUDENTS' ACCEPTANCE AND USE OF PROGRAMMING

We relied on the coordinators of fifty public arts and design undergraduate courses to forward our online survey to the students, as described in 5.4.2. We collected students' participation at an uneven pace during two periods. We gathered 242 responses from students at fourteen different universities and polytechnic institutes between mid-March and early May 2018. After excluding responses outside the scope of our study (e.g., from graduate students), we conducted and presented a preliminary analysis of the 187 valid responses we obtained (Morais, Morais, & Paiva, 2018a).

We renewed our request to the coordinators of courses from wherein we had not obtained any response at the start of the new academic year. We thus gathered another 102 responses, 83 of them valid, from students at eight universities and polytechnic institutes between the middle of September and late October 2018. We conducted and presented a summary analysis of the aggregate 270 valid responses to our survey (Morais, Morais, & Paiva, 2019), which we will now expand in further detail in this section.

### 2.1 *Participants*

We obtained 270 valid participations from students at 27 different courses. According to public data (DGEEC, 2019), those courses were attended by over 4,800 students at the time of the survey.[*] We note that 28 participants attended two courses—Visual Art & Technology at the Lisbon Polytechnic and Fine Arts & Artistic Technologies at the Viana do Castelo Polytechnic—that no longer offered any elective unit concerning computer programming when we revisited their curricula in 2019 for the purpose of compiling the panorama we presented in chapter 4.[†]

In Table 6.2.1 (overleaf), we list the universities or polytechnic institutes as well as the courses attended by valid participants.

⁂

Out of 270 valid participants, 121 (44.8%) reported a prior familiarity with computer programming, 77 (28.5%) reported they were learning to code for the first time at the time they completed the survey, and the remaining 72 (26.7%) reported having no knowledge of the subject. 53 participants (19.6%) reported they did not learn and neither expected to learn to code at their course; we note a slight discrepancy here, as only 46 participants were enrolled at a course in which computer programming was not found in a mandatory curricular unit.

[*] We may note the two Architecture courses share a substantial portion of students, with over 1300 students between them.

[†] Nonetheless, given as these participants may have benefitted from the elective units up to the time of the survey, their responses remain valid.

***Table 6.2.1**. Student participants' institutions and courses.*

| *Institution* | *Course* [a] | *Participants* | *Total students* |
|---|---|---|---|
| *Algarve University* | Communication Design | 4 | 130 |
| *Aveiro University* | Product Design & Technology | 4 | 119 |
| *Coimbra University* | Design & Multimedia | 27 | 227 |
| *Évora University* | *Music* | *8* | *174* |
| *Lisbon University* | Architecture MA (at Architecture Faculty) | 12 | 991 |
| | Architecture MA (at Technical Institute—IST) | 32 | 372 |
| | *Design* (at Architecture Faculty) | *10* | *216* |
| *Minho University* | Product Design | 8 | 119 |
| *Porto University* | Communication Design | 5 | 220 |
| | Fine Arts | 3 | 389 |
| *Trás-os-Montes Univ.* | Multimedia & Communication | 9 | 189 |
| *Bragança Polytechnic* | Communications Technologies [b] | 2 | 32 [b] |
| | Digital Games Design | 15 | 139 |
| *Castelo Branco Poly.* | Musical Electronics & Production | 2 | 33 |
| | Communication & Audiovisual Design | 6 | 110 |
| *Coimbra Polytechnic* | Multimedia Design & Communication | 18 | 136 |
| *Leiria Polytechnic* | Graphic & Multimedia Design | 1 | 333 |
| | Digital Games & Multimedia | 19 | 184 |
| *Lisbon Polytechnic* | *Visual Art &Technology* [c] | *23* [c] | *225* |
| *Portalegre Polytechnic* | Communication Design | 3 | 58 |
| *Porto Polytechnic* | Multimedia | 6 | 81 |
| | Music Composition | 11 | *21* [d] |
| | Theater—Light & Sound | 5 | - |
| | Web Technologies & Information Systems | 2 | 104 |
| *Viana do Castelo Poly.* | *Fine Arts & Technologies* [c] | *5* [c] | - |
| *Viseu Polytechnic* | Multimedia Design & Technologies | 18 | 151 |
| | Fine Arts & Multimedia | 12 | 114 |

*Notes:*
[a] Courses that only offer programming in elective units shown in italics. Units and ECTS credits shown in regular type are mandatory.
[b] As of 2019/20, the course had been discontinued and no longer accepted applications.
[c] As of 2019/20, the course no longer offered elective programming units.
[d] Student data was absent from the DGEEC database (2019) and was obtained from the Porto Polytechnic Institute's website at portal.ipp.pt.

The following table presents valid participants' gender, age, seniority, and self-assessed daily computer use, grouped by their programming knowledge level. For greater clarity, we present age as intervals, though age data was collected as a continuous integer.

***Table 6.2.2**. Students' characteristics and familiarity with programming.*

| | | *No knowledge* (*n*=72) | | *Learning* (*n*=77) | | *Prior knowledge* (*n*=121) | | *Total* (*n*=270) | |
|---|---|---|---|---|---|---|---|---|---|
| | | *n* | *%* | *n* | *%* | *n* | *%* | *n* | *%* |
| *Gender* | Female | 46 | 29.9 | 51 | 33.1 | 57 | 37.0 | 154 | 57.0 |
| | Male | 26 | 22.4 | 26 | 22.4 | 64 | 55.2 | 116 | 43.0 |
| *Age group* | 18 – 20 | 35 | 25.5 | 51 | 37.2 | 51 | 37.2 | 137 | 50.7 |
| | 21 – 24 | 29 | 28.7 | 19 | 18.8 | 53 | 52.5 | 101 | 37.4 |
| | 25 – 34 | 6 | 25.0 | 5 | 20.8 | 13 | 54.2 | 24 | 8.9 |
| | 35 + | 2 | 25.0 | 2 | 25.0 | 4 | 50.0 | 8 | 3.0 |
| *Seniority* * | 1st year | 20 | 27.0 | 34 | 45.9 | 20 | 27.0 | 74 | 27.4 |
| | 2nd year | 21 | 29.6 | 19 | 26.9 | 31 | 43.7 | 71 | 26.3 |
| | 3rd year | 22 | 23.7 | 21 | 22.6 | 50 | 53.8 | 93 | 34.4 |
| | 4th year + | 9 | 28.1 | 3 | 9.4 | 20 | 62.5 | 32 | 11.9 |
| *Daily PC use* | < 1 hour | 3 | 4.2 | 2 | 2.6 | 1 | 0.8 | 6 | 2.2 |
| | 1 - 2 hours | 10 | 13.9 | 5 | 6.5 | 5 | 4.1 | 20 | 7.4 |
| | 2 - 4 hours | 21 | 29.2 | 15 | 19.5 | 16 | 13.2 | 52 | 19.3 |
| | 4 - 8 hours | 25 | 34.7 | 44 | 57.1 | 57 | 47.1 | 126 | 46.7 |
| | > 8 hours | 13 | 18.1 | 11 | 14.3 | 42 | 34.7 | 66 | 24.4 |

We found a moderate Spearman 2-tailed correlation between all participants' seniority and greater daily computer use ($\rho$=.32; $p$<.001). In turn, daily computer use correlated with an increased familiarity with programming ($\rho$=.29; $p$<.001).

### 2.1.1 PARTICIPANTS' PROGRAMMING KNOWLEDGE

We asked participants familiar with programming (*n*=121) to report where they had learned to code. Most mentioned their present course (88), though many also reported learning to code at secondary school (25), at a vocational school (15), or at another higher education course (14). Participants also reported attending paid programming lessons (6), online courses (5), or coding workshops (1). Finally, several claimed they learned to code by themselves (15) or with the aid of a peer (1). We also asked these participants to list the programming languages they felt comfortable using. C# was the most mentioned language (29), possibly due to its application in the Unity game engine. JavaScript (23), PHP (18), Java (18), Python (13), and C++ (12) all had over ten mentions, followed by C (6), Racket (6), Processing (5), and Pascal (5). LISP was mentioned by three participants; Blueprints (Unreal Engine), Max, Matlab, Visual Basic, and Grasshopper were mentioned twice; finally, there were sole mentions of GDScript, PWGL, Ruby, SQL, Swift, and ActionScript. Recall from Table 4.2.4 that Racket, Pascal, and LISP were not mentioned in any curricular plan at the time of our review. Of the 88 participants who answered the question, 43 mentioned one programming language, 29 mentioned two languages, 12 mentioned three languages, and 3 mentioned four languages; one sole participant claimed to code in six different languages.

* Note that only four courses had plans of study longer than three years (the licentiates at the University of Porto and the integrated master's in Architecture).

## 2.2 *Students' acceptance and intentions*

We computed the acceptance constructs according to the outcome of our reliability and consistency analyses (see 5.4.3.1). We present their means and standard deviations, grouped by participants' programming knowledge, in Table 6.2.3. Note we transposed the original 7-point Likert scales to run from -3 (standing for full disagreement) to +3 (full agreement).

***Table 6.2.3.*** *Students' programming acceptance, grouped by familiarity.*

| | *No knowledge* (*n*=72) | | *First time learning* (*n*=77) | | *Prior knowledge* (*n*=121) | |
|---|---|---|---|---|---|---|
| | Mean | St. dev. | Mean | St. dev. | Mean | St. dev. |
| *Performance Expectancy* | 0.15 | 1.43 | 0.68 | 1.43 | 1.07 | 1.43 |
| *Effort Expectancy* | -0.95 | 1.24 | -0.69 | 1.21 | 0.10 | 1.48 |
| *Attitude Toward Use* | -0.35 | 1.64 | 0.25 | 1.59 | 0.91 | 1.74 |
| *Subjective Norm* | -1.49 | 1.52 | -0.55 | 1.57 | 0.06 | 1.73 |
| *Facilitating Conditions* | -0.62 | 1.29 | 0.16 | 1.56 | 0.72 | 1.50 |
| *Self-Efficacy* | -0.80 | 1.17 | 0.26 | 1.31 | 0.75 | 1.27 |
| *Anxiety* (reverse scale) | -0.33 | 1.56 | 0.14 | 1.44 | -0.60 | 1.62 |
| *Behavioral Intention* | -1.99 | 1.21 | -0.08 | 2.04 | 0.25 | 2.11 |

Except for *Anxiety*, positive values can be regarded as more favorable: a positive *Effort Expectancy* reflects an agreement with programming being easy to use or to learn, but a positive *Anxiety* reflects an agreement with programming being scary or intimidating.

⁂

We understood participants' familiarity with computer programming leads to a variant interpretation of the survey measurements (Parameswaran, Kishore, & Li, 2015). Hence, we divided responses into three groups before applying the appropriate analyses to investigate the relationships between respondents' individual characteristics and their acceptance and intention to learn or use computer programming. We found significant associations, which we describe in detail in the following subdivisions.

### 2.2.1 PARTICIPANTS WITHOUT PROGRAMMING KNOWLEDGE

Among students without any knowledge of programming ($n$=72), our 2-tailed analyses found a suggestive Pearson correlation in that older students were likelier to report less *Anxiety* toward programming ($r$=-.30; $p$=.012). We also found strong negative Spearman correlations between students' seniority and both their *Subjective* perception of *Norm* ($\rho$=-.36; $p$<.001) and their *Behavioral Intention* ($\rho$=-.36; $p$=.002). Regardless of age, it is likely that students with consolidated interests and closer to completing their degree feel less pressure to learn a new skill and have a lesser inclination to use it. In addition, we saw a conclusive correlation between participants' daily computer use and their *Performance Expectancy* of programming ($\rho$=.38; $p$=.001).

In addition, we found likely Point-biserial associations between these participants' gender and some of their views: while male students reported more positive views of the *Facilitating Conditions* ($r_{pb}$=.32; $p$=.006), female students were more likely to ascribe greater *Anxiety* to the subject of programming ($r_{pb}$=.28; $p$=.017).

### 2.2.2 PARTICIPANTS LEARNING PROGRAMMING FOR THE FIRST TIME

We found persuasive Point-biserial associations between the gender and the computer programming acceptance of participants learning to code for the first time ($n$=77). As we will present in Figure 6.2.1, male participants were likelier to report better *Effort Expectancy* ($r_{pb}$=.33; $p$=.004), *Attitude* ($r_{pb}$=.34; $p$=.003), a more favorable view of the *Facilitating Conditions* ($r_{pb}$=.32; $p$=.005), and lesser *Anxiety* about programming ($r_{pb}$=.43; $p$<.001). We also found likely evidence of male learners reporting improved *Performance Expectancy* ($r_{pb}$=.27; $p$=.019), *Self-Efficacy* ($r_{pb}$=.26; $p$=.024), and *Behavioral Intention* to code in the future ($r_{pb}$=.26; $p$=.027). In addition, we found a Pearson 2-tailed negative correlation between first-time learners' age and *Anxiety* ($r$=-.31; $p$=.007).

Spearman 2-tailed analysis also yielded multiple compelling correlations ($p$≤.001) between first-time learners' perceived voluntariness of computer programming use and strongly improved *Performance* ($\rho$=-.45) and *Effort* ($\rho$=.51) expectancies, *Attitude* ($\rho$=.55), as well as a diminished *Anxiety* ($\rho$=-.40) and a greater *Behavioral Intention* ($\rho$=.39).

⁂

In Figure 6.2.1, we present the significant acceptance differences between male ($n$=26) and female (51) first-time learners, as well as between male (64) and female (57) participants familiar with programming, as we will report in the next subdivision:

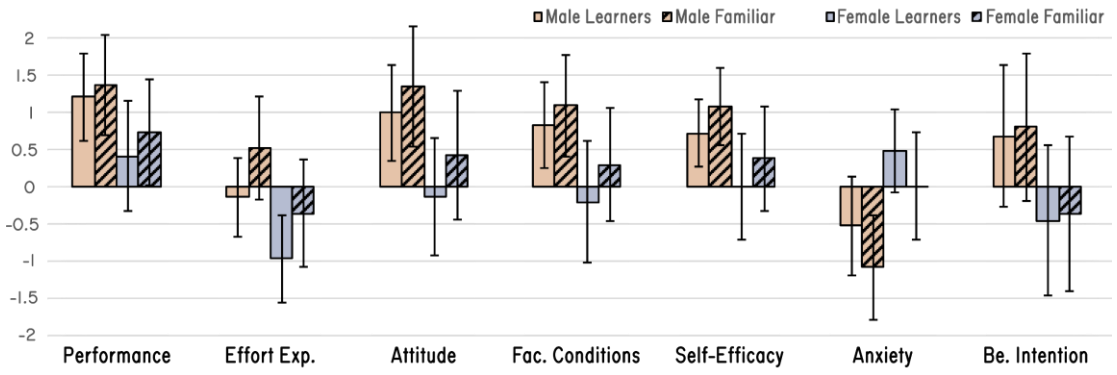



***Figure 6.2.1.*** *Mean programming acceptances and Behavioral Intentions of male and female students. (Error bars equal 1 standard deviation.)*

### 2.2.3 PARTICIPANTS WITH PRIOR PROGRAMMING KNOWLEDGE

Our analyses of the responses of participants that reported familiarity with computer programming ($n$=121) yielded differences between genders' acceptance, as presented in Figure 6.2.1, and seen with first-time learners. We found Point-biserial associations concluding that male participants were likelier to report better *Effort Expectancy* ($r_{pb}$=.30; $p$=.001) and *Attitude* ($r_{pb}$=.26; $p$=.003), a more favorable view of the *Facilitating*

*Conditions* ($r_{pb}$=.27; $p$=.003), improved *Self-Efficacy* ($r_{pb}$=.28; $p$=.002), lesser *Anxiety* ($r_{pb}$=.35; $p$<.001), and a greater *Intention* to use computer programming in the near future ($r_{pb}$=.28; $p$=.002). Male participants were also likelier to ascribe slightly better *Performance Expectancy* ($r_{pb}$=.22; $p$=.013).

Spearman 2-tailed analysis also yielded likely correlations between the daily computer use of these participants and slightly improved views on *Performance* ($\rho$=.21; $p$=.021), *Attitude* ($\rho$=.23; $p$=.011), and *Self-Efficacy* ($\rho$=.20; $p$=.028). In addition, we found conclusive correlations between these participants' greater perceived voluntariness of programming and improved *Effort Expectancy* ($\rho$=.36; $p$<.001) and *Attitude* ($\rho$=.36; $p$<.001), as well as a lesser *Anxiety* ($\rho$=-.27; $p$=.003), and a greater *Intention* to code in the near future ($\rho$=.28; $p$=.002). These participants' perceived voluntariness also showed likely positive correlations with their *Performance Expectancy* ($\rho$=.22; $p$=.016), *Subjective Norm* ($\rho$=.20; $p$=.029), and their *Self-Efficacy* ($\rho$=.22; $p$=.019).

## 2.3 *Students' intentions and use behavior*

We investigated the relationships between participants' acceptance measurements and their *Behavioral Intention*. We found several conclusive Pearson 2-tailed correlations, which we present in Table 6.2.4:

***Table 6.2.4.*** *Pearson correlations between students' acceptance and Behavioral Intention.*

| | *No knowledge* ($n$=72) | | *Learning* ($n$=77) | | *Prior knowledge* ($n$=121) | |
|---|---|---|---|---|---|---|
| | *Correlation (r)* | *Significance (p)* | *Correlation (r)* | *Significance (p)* | *Correlation (r)* | *Significance (p)* |
| *Performance Expect.* | .46 | < .001 | .38 | < .001 | .70 | < .001 |
| *Effort Expectancy* | .52 | < .001 | .45 | < .001 | .76 | < .001 |
| *Attitude Toward Use* | .57 | < .001 | .56 | < .001 | .73 | < .001 |
| *Subjective Norm* | .56 | < .001 | .38 | .001 | .58 | < .001 |
| *Facilitating Conditions* | *Not significant* | | .35 | .002 | .66 | < .001 |
| *Self-Efficacy* | .30 | .010 | .37 | .001 | .68 | < .001 |
| *Anxiety* | *Not significant* | | -.43 | < .001 | -.26 | .005 |

If we are permitted to draw such a comparison, we note the stronger association between acceptance and intention in participants with prior programming knowledge. Though those yet to learn programming may have had difficulties assessing conditions, we find it noteworthy that emotion (i.e., *Anxiety*) did not relate to their intention.

We also investigated the programming use reported by those participants having either a prior familiarity or a first contact with the subject at the time of our survey. We found compelling and strong Spearman 2-tailed correlations between reported use frequency and use intensity, both among the first-time learners ($n$=77; $\rho$=.69; $p$<.001) and the knowledgeable participants ($n$=121; $\rho$=.81; $p$<.001). We chart both groups' reported use frequency and intensity in Figure 6.2.2:

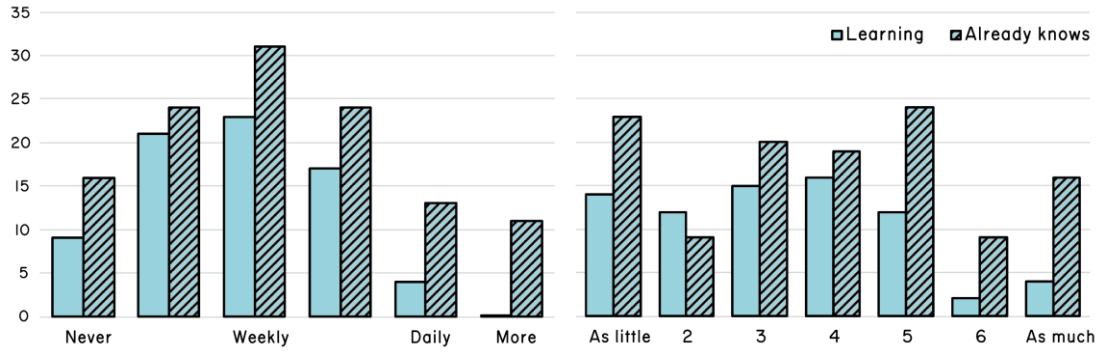



***Figure 6.2.2**. Students' reported programming use frequency (left) and intensity (right, 7-point Likert scale from "as little as possible" to "as much as possible").*

As suggested by the UTAUT model (Venkatesh et al., 2003) discussed in 5.2.3, we investigated to what extent participants' reported programming use correlated with their regard of the *Facilitating Conditions* and their *Behavioral Intention*. We found strong and compelling Spearman 2-tailed correlations at the $p<.001$ level among the responses of both groups of participants, as presented in Table 6.2.5:

***Table 6.2.5.** Spearman correlations between students' programming use and their regard of the Facilitating Conditions and Behavioral Intention (all significant at p<.001).*

| | *First time learners* (*n*=77) | | *Familiar w/ programming* (*n*=121) | |
|---|---|---|---|---|
| | *Use Frequency* (*ρ*) | *Use Intensity* (*ρ*) | *Use Frequency* (*ρ*) | *Use Intensity* (*ρ*) |
| *Facilitating Conditions* | .48 | .43 | .37 | .50 |
| *Behavioral Intention* | .59 | .54 | .52 | .64 |

Among more experienced participants, *Facilitating Conditions* and *Intention* correlated more directly with their intensity than their frequency of programming use.

⁂

In the next chapter, we will report case studies in which we observed and surveyed students learning to code in six distinct arts and design undergraduate courses. As we will see, several dimensions of those students' acceptance and use of programming will diverge from the general findings presented in this section. Those divergences will be discussed in chapter **8**.

## 3 PROFESSORS' ACCEPTANCE AND USE OF PROGRAMMING

As with the survey of students presented in the previous section and following the procedure described in 5.4.2, we relied on the coordinators of the arts and design undergraduate courses we had tentatively identified as offering at least one computer programming unit (see 4.2) to forward our survey to academic staff. We collected responses at an uneven pace, gathering 68 from professors at 16 different institutions between the middle of March and early May 2018, which were the basis for a preliminary analysis (Morais, Morais, & Paiva, 2018b). After a renewed request to the coordinators of the courses from wherein we had not obtained any response, we gathered 12 additional responses from professors at seven institutions during October 2018.

### 3.1 *Participants*

We obtained 80 responses from professors, with teaching assignments at courses that include programming as a part of their curricula, at 18 different public higher education institutions. Of these, 30 (37.5%) reported proficiency in computer programming, and 50 (62.5%) reported no knowledge of the subject. We present participants' gender, age, teaching experience, and daily computer use in Table 6.3.1. For clarity, we show age and experience in intervals, though integer data was collected and analyzed.

***Table 6.3.1.*** *Professors' characteristics and familiarity with programming.*

| | | *No knowledge* (*n*=50) | | *Knows programming* (*n*=30) | | *Total* (*n*=80) | |
|---|---|---|---|---|---|---|---|
| | | *n* | *%* | *n* | *%* | *n* | *%* |
| *Gender* | Female | 20 | 40.0 | 5 | 16,7 | 25 | 31.3 |
| | Male | 30 | 60.0 | 25 | 83,3 | 55 | 68.8 |
| *Age group* | ≤ 34 | 2 | 4.0 | 1 | 3,3 | 3 | 3.8 |
| | 35 – 44 | 22 | 44.0 | 20 | 66,7 | 42 | 52.5 |
| | 45 –54 | 12 | 24.0 | 8 | 26,7 | 20 | 25.0 |
| | 55 + | 14 | 28.0 | 1 | 3,3 | 15 | 18.8 |
| *Teaching experience (years)* | ≤ 4 | 3 | 6.0 | 3 | 10.0 | 6 | 7.5 |
| | 5 – 9 | 8 | 16.0 | 2 | 6,7 | 10 | 12.5 |
| | 10 – 19 | 17 | 34.0 | 16 | 53,3 | 33 | 41.3 |
| | 20 – 29 | 12 | 24.0 | 7 | 23,3 | 19 | 23.8 |
| | 30 + | 10 | 20.0 | 2 | 6,7 | 12 | 15.0 |
| *Daily PC use* | < 1 hour | 0 | 0.0 | 0 | 0.0 | 0 | 0.0 |
| | 1 - 2 hours | 3 | 6.0 | 0 | 0.0 | 3 | 3.8 |
| | 2 - 4 hours | 15 | 30.0 | 2 | 6,7 | 17 | 21.3 |
| | 4 - 8 hours | 25 | 50.0 | 18 | 60.0 | 43 | 53.8 |
| | > 8 hours | 7 | 14.0 | 10 | 33,3 | 17 | 21.3 |

Participants' median age was 43 years; the two youngest were 33, and the two eldest were both 67. We found a conclusive Point-biserial association between participants' youth and their likelihood of knowing how to program ($r_{pb}$=-.31; $p$=.005); participants' daily PC usage also positively correlated with a likelihood of knowing the subject ($r_{pb}$=.32; $p$=.003).

The appropriate analysis also suggested male participants were likelier to know computer programming ($\varphi$=.24; $p$=.029).

### 3.1.1 PARTICIPANTS' PRACTICE AND PROGRAMMING KNOWLEDGE

The participants' median teaching experience was 17 years; one reported as little as one year of experience, while one of the eldest professors reported 46 years of teaching experience. Participants reported their affiliation with the universities of Algarve (2), Aveiro (2), Coimbra (7), Évora (3), Lisbon (12), Madeira (1), Minho (5), Porto (9), and Trás-os-Montes, as well as the polytechnic institutes at Bragança (4), Castelo Branco (2), Coimbra (5), Leiria (4), Lisbon (4), Portalegre (1), Porto (13), Tomar (2), and Viseu (3).

Participants that knew programming ($n$=30) were invited to report where they had learned it. Most mentioned autonomous learning (19) or undergraduate studies (18). Several participants also mentioned vocational school (4), coding workshops (3), postgraduate studies (2), online courses (2), the workplace (2), or paid lessons (1).

We also invited these participants to report the languages they had experience coding. C was the most mentioned language (13, including 4 mentions of Arduino), followed by JavaScript (11), Processing (10), C++ (8), Java (8), Python (8), and PHP (6). Several participants also mentioned experience with C# (5), Max (4), Matlab (3), BASIC (2), LISP (2), Pascal (2), and Visual Basic (2). Other languages were mentioned only once: ActionScript, COBOL, CSound, FORTRAN, Grasshopper, Julia, LabVIEW, Lingo, Lua, P5.js, PostScript, Pure Data, Racket, Scheme, SQL, and TouchDesigner. One participant reported just one language; 7 mentioned two, 6 mentioned three, 2 mentioned four, 6 mentioned five, and 3 reported using six different languages; one additional participant claimed experience in eleven different programming languages.

⁂

Of the thirty professors that reported being proficient at computer programming, 17 (21.3% of all participants in the survey) reported they had been assigned curricular units on the subject at the time of our survey. These professors were affiliated with just nine institutions. Their median age was 41 (ranging from 36 to 58), and their median teaching experience was 15 years (ranging from 2 to 40); only two professors were female.

We invited these seventeen participants to list the programming languages that were part of their teaching assignments. Most mentioned JavaScript (8) and Processing (6), though several mentioned teaching C (4, including 3 mentions of Arduino), C# (3), Java (3), Max (3), Python (3), and PHP (2). Certain languages were mentioned just once: AutoLISP, C++, Grasshopper, P5.js, Pure Data, Racket, Scheme, and TouchDesigner. One participant reported they would be teaching a single language within their assigned curricular units; 7 professors would be teaching two languages, 4 would be teaching three, 2 would be teaching four, and one other professor would be teaching five different languages as part of their assignments. These languages were not necessarily taught to students at arts and design courses, given the likelihood of participants having broader teaching assignments.

## 3.2 *Professors' acceptance and intentions*

We computed the acceptance constructs from the items our analyses validated as reliable and consistent; hence we excluded *Facilitating Conditions* from further analysis (see 5.4.3.2). We present the construct means and standard deviations in Table 6.3.2, grouped by participants' level of programming knowledge:

***Table 6.3.2.*** *Professors' programming acceptance, grouped by familiarity.*

| | *No knowledge* (*n*=50) | | *Know programming* (*n*=30) | |
|---|---|---|---|---|
| | Mean | St. dev. | Mean | St. dev. |
| *Performance Expectancy* | -0.61 | 1.44 | 1.53 | 1.67 |
| *Effort Expectancy* | -0.85 | 1.44 | 1.43 | 1.45 |
| *Attitude Toward Use* | -0.94 | 1.66 | 1.95 | 1.42 |
| *Subjective Norm* | -1.97 | 1.29 | 0.28 | 2.00 |
| *Self-Efficacy* | -0.75 | 1.46 | 1.37 | 1.39 |
| *Anxiety* (reverse scale) | -0.82 | 1.50 | -2.39 | 0.88 |
| *Behavioral Intention* | -2.29 | 1.14 | 1.52 | 2.04 |

For the purpose of our presentation, we transposed the original 7-point Likert scales to run from -3 (full disagreement) to +3 (full agreement). Again, we note positive values stand for more favorable views in most constructs, with *Anxiety* being the exception.

⁂

Following the suggestion that different familiarity with computer programming implies different contexts of interpretation of the survey measurements (Parameswaran et al., 2015), we divided responses into two groups before applying the appropriate analyses to investigate the relationships between participants' individual characteristics and their acceptance and intention to either learn or use computer programming. We found significant associations, which we describe in detail in the following subdivisions.

### 3.2.1 PARTICIPANTS WITHOUT PROGRAMMING KNOWLEDGE

Our analyses yielded few suggestive relationships worth noting among the participants who reported no knowledge of computer programming (n=50). Concerning participants' gender, we found female professors were somewhat likelier to ascribe greater *Anxiety* to the subject ($r_{pb}$=.31; $p$=.031). With regard to participants' teaching experience, we found a suggestive Pearson 2-tailed correlation holding that the longer a participant's career, the lesser *Self-Efficacy* they reported about learning to code ($r$=-.30; $p$=.035).

### 3.2.2 PARTICIPANTS WITH PRIOR PROGRAMMING KNOWLEDGE

In our analysis of the responses of participants that reported an existing familiarity with computer programming ($n$=30), we found persuasive Point-biserial associations. Those who were teaching computer programming at the time of the survey were likelier to report better *Performance Expectancy* ($r_{pb}$=.46; $p$=.014), *Attitude Toward Use* ($r_{pb}$=.44; $p$=.019), *Self-Efficacy* ($r_{pb}$=.53; $p$=.005), and greater *Behavioral Intention* ($r_{pb}$=.54; $p$=.003). Though expected, it is worth noting that those teaching at the time also reported a lesser perceived voluntariness of programming use ($r_{pb}$=-.45; $p$=.017). We also found suggestive Pearson 2-tailed correlations holding that older participants were likelier to report lesser *Performance* ($r$=-.42; $p$=.025) and *Attitude* ($r$=-.40; $p$=.030).*

In addition, we found likely Spearman 2-tailed correlations between this group of participants' perceived voluntariness of programming and worse views regarding *Performance Expectancy* ($\rho$=-.43; $p$=.021) and *Self-Efficacy* ($\rho$=-.43; $p$=.023), as well as a lessened *Behavioral Intention* to code in the near future ($\rho$=-.44; $p$=.016). It is interesting to observe how these professors and the students we surveyed (see §2.2.3) have dissimilar reactions to greater voluntariness.

## 3.3 *Professors' intentions and use behavior*

We looked for relationships between professors' views of computer programming and their *Behavioral Intention* of either learning it or using it in the near future. We present the significant correlations in Table 6.3.3:

***Table 6.3.3.*** *Pearson 2-tailed correlations between professors' programming acceptance and their Behavioral Intention to learn it or use it.*

| | *No knowledge* ($n$=50) | | *Knows programming* ($n$=30) | |
|---|---|---|---|---|
| | *Correlation (r)* | *Sig. (p)* | *Correlation (r)* | *Sig. (p)* |
| *Performance Expectancy* | .63 | < .001 | .78 | < .001 |
| *Effort Expectancy* | .51 | < .001 | .64 | < .001 |
| *Attitude Toward Use* | .69 | < .001 | .81 | < .001 |
| *Subjective Norm* | .71 | < .001 | .50 | .005 |
| *Self-Efficacy* | .44 | .001 | .73 | < .001 |

*Anxiety* evinced no correlations to *Behavioral Intention*; however, *Self-Efficacy* correlated rather strongly with intention among professors with programming knowledge.

Though *Behavioral Intention* showed a compelling and strong Spearman 2-tailed correlation with programming intensity ($n$=30; $\rho$=.70; $p$<.001), we found no significant association between participants' intention and programming frequency ($p$>.05). Still, our analysis yielded a conclusive and strong correlation between participants' programming use frequency and intensity ($\rho$=.59; $p$=.002).

* Likewise, more participants with higher teaching experience reported lesser *Performance Expectancy* ($r$=-.40; $p$=.034) and *Attitude* ($r$=-.42; $p$=.024).

In the following chart, we present counts of participants' reported programming use frequencies and intensities, grouped by whether they were teaching any computer programming class at the time:

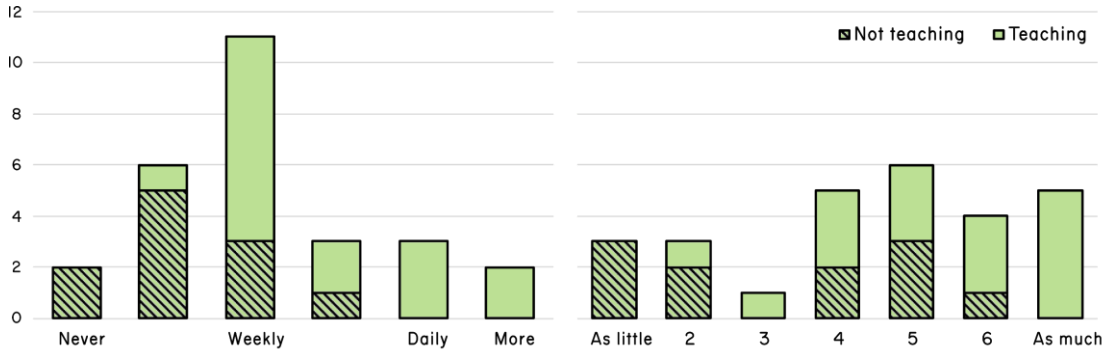



***Figure 6.3.1.*** *Professors' reported programming use frequency (left) and intensity (right, 7-point Likert scale from "as little as possible" to "as much as possible").*

It is quite evident participants who were teaching computer programming at the time of the study report higher frequencies and intensities of programming use.

## 3.4 *Professors' views on teaching to code*

As presented in Table 5.3.3, we asked those professors who reported familiarity with programming an open question about the value of teaching programming to arts and design students.

We obtained eighteen responses. Eight participants noted the expressive potential of programming, such as the professor that believed programming was valuable because *"it makes the use of digital media and the creation of interactive environments accessible to multimedia artists,"* or the respondent that explained, *"programming is an essential meta-medium for digital media manipulation and the creation of new tools and discourses for Design practice."*

Four participants gave concise answers valuing programming as a fundamental technique to solve problems and teach logical thinking. Three contributors wrote about programming as part of *"essential literacies,"* as one such respondent put it, while another professor explained, *"the equipment and software we use in our daily life requires that knowledge."*

Five of the responses emphasized the value of programming as a practical tool, including specific mentions of web design (3) and architectural design (1). One participant explicitly mentioned the job market, noting the value of programming for *"developing projects that generate employment opportunities."* Somewhat tautologically, one respondent justified teaching programming with the subject being a part of their teaching assignments. Three participants revealed their personal interest and enjoyment of matters related to computing.

※

In the next chapter, we will report on a collective case study concerning six curricular units at distinct arts and design licentiates. We will revisit a few of this survey's main findings in our observations and interviews with the professors, which brought to light complementary results. A comparison and discussion of findings will be the focus of chapter **8**.

*(blank page)*

# 7

# *Case studies in programming education within undergraduate arts and design*

★ SYNOPSIS We describe and present findings from six case studies we conducted to characterize how computer programming is taught and learned at public undergraduate arts and design courses. We present an overview of our cases in section §1. From section §2 to §7, we describe and analyze the instrumental case studies we carried out at five different institutions. Each case corresponded to one of six curricular units offered at six distinct licentiate courses on the fields of Audiovisual and Media Production (§2, §5, and §6), Design (§3 and §4), and Performing Arts (in our case, Music, §7). Finally, in §8, we then present an inter-case analysis of all studies.

## 1 OVERVIEW

Following our initial review, presented in 4.2, of Portuguese public arts and design undergraduate courses that potentially offered a contact with computer programming to their students, we carried out surveys of students and professors' acceptance of the subject. Alongside the research framework focused on educational interactions, which we described in 5.2, the findings of those preliminary surveys presented in the previous chapter informed the collective case study herein reported.

As we described in 5.5, we leveraged the institutional contacts we had made during our previous research efforts to confirm the relevance and then request support to carry out instrumental case studies focused on specific curricular units related to the subject of computer programming. We obtained the support of five institutions for carrying out the six case studies summarized in Table 5.5.1, which took place during both terms of the 2018/2019 academic year.

In Table 7.1.1, we restate the codewords we employed to identify each case and summarize the type and length of contact with computer programming which occurred in each curricular unit. As to ensure participants' anonymity, we refer to the courses offering each unit by designations inspired by their main CNAEF classification (see 4.1.3): Audiovisual and Media Production (AMP), Design, or Music.

***Table 7.1.1.*** *Overview of contact with computer programming across our cases.*

| *Case / Institution* | *Area / Year* | *Main contents* | *Students* | *Contact [a] (hours/type)* | *Periodic Evaluation* |
|---|---|---|---|---|---|
| PRIMER (RED) | AMP 1st - 3rd [b] | Project-based introduction to Processing and Unity. | 180 (4 classes) | 54 L | Project (ind./gr.) |
| GRAPHIC (RED) | Design *(elective)* | Introduction to Processing, computer graphics techniques. | 15 | 30 M | 2 exercises (ind.) Essay (ind.) |
| SKETCH (LIME) | Design 2nd | Foundations of programming with Processing. | 61 (2 classes) | 42 M | 2 exercises (ind.) Project (group) |
| WEAVE (BLUE) | AMP 2nd | Brief introduction to JavaScript and PHP techniques. | 91 (4 classes) | 10 T [c] 10 L [c] | Project (ind./gr.) Exam |
| TOUCH (ROSE) | AMP 3rd | 3D graphics with P5.js, brief introduction to Arduino. | 19 | 48 M | Project (group) |
| FLOW (AMBER) | Music 1st | Introduction to Max, algorithmic composition techniques. | 5 | 30 M | Project (individual) |

*Notes:* [a] (T)heoretical lectures; (M)ixed theoretical-practical sessions; (L)aboratory practice sessions.

[b] Second-year students attended the unit under a transitional curriculum. Elective for third-year students.

[c] The subject of computer programming was only introduced in the five last weeks of the term.

We estimated each unit's session time dedicated to computer programming after deducting any suppressed sessions (e.g., due to national holidays or field trips). With the exception of WEAVE, all curricular units dedicated the entirety of their syllabi and sessions to computer programming.

We followed the procedures laid out in 5.5.1 in carrying out each individual case study. We collected and analyzed documentation related to courses and curricular units, observed theoretical lectures and practical sessions, reviewed learning management systems and online educational resources, witnessed and analyzed students' practical works, carried out surveys of students' acceptance and use of computer programming, and conducted individual interviews with the professors. Still, as then testified by Table 5.5.2, there was great heterogeneity in the level of access we were granted and, consequentially, in the data we were able to gather in each case.

At this stage of our investigation, it was crucial for the researcher to establish a rapport with the professors involved in teaching the curricular units, not only to gain access to the unit's activities, but mainly so that the non-participating observation could happen as non-obtrusively as possible. In total, ten professors of varying professional and academic backgrounds and teaching experiences were involved in our cases. In Table 7.1.2, we present these professors, alongside listing the observational contact—during lectures, practical sessions, and project presentation sessions—and the type of interview we were granted. We note that we assigned each docent a codename in order to comply with their requests for anonymity.

***Table 7.1.2.*** *Overview of professors participating in the cases.*

| *Case* | *Codename* | *Background* | *Teaching experience and 2018/19 assignments* | *Obs. contact* | *Interview* |
|---|---|---|---|---|---|
| PRIMER | 'Alan' | Music | 30+ years. 3rd year exclusively at RED. Multiple assignments. | 10.5 hours (4 sessions) | Skype |
| | 'Bill' (Assistant) | Music | 15 years. Newly hired at RED, also teaching at another institution. | 9 hours (4 sessions) | Email |
| | 'Carl' (Assistant) | Fine Arts | 7 years. Newly hired at RED, also teaching at another institution. | 12 hours (4 sessions) | Skype |
| GRAPHIC | 'Douglas' | Computer Science | 3rd year at RED, no prior experience. Assigned 4 units. | 6 hours (2 sessions) | Skype |
| SKETCH | 'Edwin' | Design | 13 years. 2nd year at LIME, exclusive. Assigned 4 units. | 24 hours (8 sessions) | Face-to-face |
| WEAVE | 'Frank' | Computer Engineering | 15 years. 3rd year exclusively at BLUE. Assigned 4 units. | 8 hours (4 sessions) | Email |
| | 'Grace' (Assistant) | Computer Engineering | 30+ years only at BLUE's parent university. Multiple assignments. | *Unauthorized* | Email |
| | 'Hal' (Assistant) | Computer Engineering | Prior experience. Newly hired at BLUE. | *Unauthorized* | *Declined* |
| TOUCH | 'Ivan' | Design | 16 years. 7 years at ROSE, also teaching at another institution. | 8 hours (2 sessions) | Face-to-face |
| FLOW | 'Joel' | Music | Almost 20 years. 1st year exclusive at AMBER. Assigned 6 units. | 10 hours (5 sessions) | Face-to-face |

Except for the professor we refer to as 'Grace,' all the other professors we had contact with were male; hence, our sample had a greater than expected gender imbalance (see 4.1.3.3). We also note that we had no contact with Professor 'Hal,' who assisted with teaching one of the practical laboratory classes at WEAVE.

⁂

In the following six sections, we will describe and analyze each specific case. In section §8, we will then proceed with an inter-case analysis, which will inform the next chapter's discussion.

## 2 PRIMER: MULTIMEDIA PROGRAMMING PROJECTS

This case concerns a curricular unit, which we shall refer to as PRIMER, that offered a practical introduction to computer programming to students attending a course in the field of Media Production ('AMP') at the faculty we shall refer to as RED. The study was conducted under unique conditions, as 180 students from the course's three years attended PRIMER simultaneously due to a curricular transition. In the coming academic years, the unit is to be attended by first-year students.

In subsection §2.1, we will present the host institution and the licentiate offering PRIMER, and in §2.2, we will turn our focus to the curricular unit—its plan and syllabus, session schedule, student make-up, and professors. In subsequent subsections, we will present our findings, starting in §2.3 by describing our classroom observations made during six different sessions; in §2.4, we will report what we were able to gather from the unit's Moodle learning management system, which provided students access to educational resources as well as a communication backbone to the unit's activities; then, in §2.5, we will also report on our observation of final project presentation and debate sessions. In §2.6, we will report findings from the quantitative technology acceptance and use surveys of students. We conducted individual interviews with the unit's professors, which we will report in §2.7. Finally, in subsection §2.8, we will address the successes and the limitations we faced while conducting this case study.

### 2.1 *The course and its host institution*

Alongside the case we refer to as GRAPHIC (presented in §3), the PRIMER study was carried out at RED, a faculty which, though integrated into a large university with tens of thousands of students, has its facilities located at a remove from the main campi. Over 1,500 study at the faculty. Around three-quarters of those students are enrolled at one of the several licentiates offered in the fields of fine arts, media production, and design, while the remainder attends either independent master's courses or doctoral programs related to those fields.

✦ THE COURSE The course offering the curricular unit under study grants a licentiate classified as in Audiovisual and Media Production under the CNAEF (see 4.1.3). Though unrelated to its real designation, we shall henceforth refer to the course by the initials 'AMP'. Prior to 2018, the licentiate had several variants sharing a common first-year curriculum. A unit introducing computer programming was only offered as an elective, and only to third-year students enrolled in the variant dedicated to interactive environments.

After RED promoted a deep revision of curriculum in 2018, the variants were replaced by a single unified curriculum, from which students can then attain one of several 'minor' degrees depending on the elective units they chose to complete during the course. Notably, the curricular unit introducing computer programming, which we refer to as PRIMER for the purpose of this study, became mandatory and was moved to the fall semester of the first year.

✦ STUDENTS AND ACCESS There were over 200 students (68% of whom were female) enrolled in the AMP course according to official data for the 2018/2019 academic year (DGEEC, 2019). Almost 200 new students had arrived at the course in the three years from 2016/17 to 2018/19. Those arriving from secondary school brought relatively high grades: the average application grade for those admitted in 2018/19 exceeded 17 (77th percentile), while the last student admitted under the general contest had applied with a grade that exceeded 16.[*]

We found it noteworthy that 21% of the students enrolled in 2015/16 and 2016/17 dropped out of the course before completing their first year, according to the same data set. We should emphasize these students met the previous curriculum, and thus we may assume that the high drop-out rate had been in the mind of the designers of the new course plan. Over 80 students graduated during the same two-year period, with an average final grade over 15.

## 2.2 *The PRIMER curricular unit*

PRIMER is our chosen codeword to refer to this curricular unit that offers students their first contact with programming. Enrollment became mandatory under the new 2018 curricular plan of the licentiate, which moved the unit to the first semester of the first year. PRIMER is classified as a laboratory practice, and sessions take four and a half hours of students' weekly schedule.

✦ OBJECTIVES AND SYLLABUS The curricular unit plan (CUP) publicly available at the faculty's website[†] was rather brief concerning PRIMER's objectives, stating students were *"to understand and use multimedia tools, looking to provide him/her with the essential knowledge in the field of object-oriented computing for use in artistic fields."* In addition, the CUP included the following syllabus, which explicitly mentioned which specific programming and development tools were to be explored by learners:

> Multimedia elements. Types of information: text, image, audio, video. Formats and standards of representation, compression, and transmission. General notions of programming in multimedia environment context: scripting, algorithms, data structures, response/interactivity. Authoring systems, integrated development environments (IDE). Guided Exploration of Unity and Processing IDEs.
>
> — *from the PRIMER curricular unit plan for the 2018/19 academic year.*

The syllabus thus builds toward project development using either Processing or Unity. Processing, taught in 26 other units comprising the panorama we presented in 4.2, is an open-source programming language and development environment aimed at educators and visual artists (see 3.3.2.3 for more details). The GRAPHIC (§3) and SKETCH (§4) case studies also focused on units in which Processing was taught. Unity, taught in seven other units that were part of our panorama (4.2), is a commercial game creation system

[*] Portuguese secondary schooling and higher education both employ 20-point grading scales.

[†] Reference omitted as to comply with the participants' request for anonymity.

for desktop operating systems (see 3.3.2.5). Unity users can design three-dimensional and two-dimensional environments and then use the included C# language to code the behaviors of objects within those environments.

✦ TEACHING AND EVALUATION METHODS In addition to the guided exploration of tools mentioned in the syllabus, the CUP made explicit that the teaching methodology ought to be *"aiming at a wide learning autonomy,"* and that the Moodle platform was to be leveraged for sharing educational resources and for students' participation.

The evaluation methodology reemphasized student autonomy: a single practical project, developed by individuals or groups of students, weighed 80% of the final grade. The remaining 20% were to result from professors' appreciation of students' attendance and participation in class.

✦ BIBLIOGRAPHY The unit plan also included a section with PRIMER's recommended references, which we present here:


- Lavieri, Edward (2015). *Getting Started with Unity 5*. Packt Publishing.
- Lieser, Wolf (2009). *Arte Digital*. H. F. Ullmann.
- Manovich, Lev (2002). *The Language of New Media*. The MIT Press.
- Manovich, Lev (2013). *Software Takes Command*. Bloomsbury Academic.
- Noble, Joshua (2009). *Programming Interactivity*. O'Reilly.
- Reas, Casey & Fry, Ben (2007). *Processing: a programming handbook*. The MIT Press.
- Reas, Casey & Fry, Ben (2010). *Getting Started with Processing*. O'Reilly.
- Ribeiro, Nuno M. (2012). *Multimédia e Tecnologias Interativas* (5ª edição). FCA.


We found little distinctiveness in PRIMER's bibliography. Reas' and Noble's books were recommended reading for several of the curricular units in the panorama we presented in 4.2. As learning Unity was less prevalent, Lavieri's introduction was recommended in only one other bibliography. The single Portuguese language entry—Ribeiro's panorama of digital media and web technologies—was found in four other CUPs, as was Lieser's picture-rich introduction to contemporary digital art. Of Lev Manovich's theoretical books on digital media, 2013's *Software Takes Command* was the single book recommendation that was unique to PRIMER, while the earlier *Language of New Media* was a part of the bibliographies of two other units.

✦ STUDENTS As we described, PRIMER was moved from the fifth to the first semester of the curricular plan at the time of our study. Hence, we had the opportunity to observe the unit as presented for the first time to first-year students. Under a transitional curricular plan imposed by the faculty's scientific council, second-year students were also required to attend and complete the unit. Third-year students remained under the old curricular plan, thus conserving the option to attend PRIMER as an elective. Thus, even though the course admits a little over sixty students a year, around 180 students were enrolled in the curricular unit, two-thirds of which were female, though we were not given exact figures. Sixty students were newly arrived at the course, while the remaining were either in their second or their third year.

- CLASSES & SESSION SCHEDULE Students were divided into four classes of around forty-five students each. Three of the classes, which we shall refer to as classes I, II, and III, were comprised of a mix of first and second-year students, as students were assigned to classes according to their available schedule depending on their choice of elective units. Class IV was comprised exclusively of third-year students attending the unit as an elective. As mentioned, PRIMER was scheduled for four and a half hours of laboratory practice per week. This time was divided into two non-consecutive sessions for each class—a longer three-hour session and a shorter ninety-minute session.

- THE PROFESSORS Three professors shared lecturing and supervision duties. Before the introduction of the new curriculum, PRIMER had been solely the responsibility of the professor we refer by the pseudonym 'Alan,' given our commitment to participants' anonymity. Professor Alan remained as the curricular unit's coordinator but only led the sessions with class I. Two assistants, who we shall refer to as 'Bill' and 'Carl,' were hired to lecture and supervise the other three classes, though in coordination with Alan. Bill led class II, and Carl led class IV (comprised of third-year students). Class III was shared between the two assistants: Bill led the shorter 90-minute session, and Carl led the longer 3-hour session. We did not obtain an explanation for these arrangements.

  - PROFESSOR 'ALAN' explained he had over thirty years of teaching experience, all while developing a career on the intersection of music and computer technology, though he added that he became interested in multimedia art at the turn of the century. Alan also told us he had taught at several higher education institutions throughout his career before joining RED, where he was now teaching exclusively, two years prior to our inquiry. Alan added he had educational assignments at the faculty's master's and doctoral programs, besides chairing PRIMER and another unit in the same licentiate. He also told us that he was proficient in Max, CSound, and OpenMusic, and that he had been working with Unity in recent years.

  - 'BILL' told us he had fifteen years of consistent teaching experience, adding that at the time, he was also teaching units related to programming at a neighboring polytechnic institute. Bill added that his individual interests concern applications of computer programming to musical composition. He also told us he had working experience with multiple programming languages typically used by musicians and composers, such as Max, CSound, Pure Data, or SuperCollider, in addition to Processing and OpenFrameworks/C++. He also recognized he did not have much experience with Unity at the start of the semester.

  - 'CARL' told us he was also teaching at another university at the time, where he had been employed for the past seven years. He added that he would be leading one other unit introducing computer programming at that other institution. Carl explained he had been invited to lead Processing workshops at RED years prior, though it was the first time he was teaching a curricular unit at the faculty. He mentioned he had a background in Painting and Fine Arts and that he maintained an artistic practice, where he worked, in addition to Processing, with Max or Pure Data, as well as Arduino microcontrollers. Carl also explained he was pursuing his Ph.D. and finding it hard to balance the demands of his research and his work.

## 2.3 *Session observation*

Presumably due to the challenges the transitional curriculum presented at the beginning of the academic year, we faced some delays in obtaining the necessary authorizations. Therefore, we were only able to observe some sessions with the semester already well underway. By the time of our first visit, during the seventh week of the semester, we had missed most content exposition, as sessions had adopted a workshop format to support students' project development. We made six visits to RED to observe those sessions, as summarized in Table 7.2.1.

***Table 7.2.1.*** *Summary of the observed sessions of the PRIMER curricular unit.*

| *Class (yr.)* | *Professor* | *Weeks in* | *Duration* | *Students present* | *Projects supervised* |
|---|---|---|---|---|---|
| I (1st/2nd) | Alan | 7 | 180' | 38 (roll call) | 8 projects (17 students) *– just half the session* |
| I (1st/2nd) | Alan | 9 | 90' | 18 (roll call) | 5 projects (6 students) |
| II (1st/2nd) | Bill | 9 | 90' | 15 (estimated) | 10 projects (10 students) |
| III (1st/2nd) | Carl | 7 | 180' | 40 (estimated) | 20 projects (20 students) |
| III (1st/2nd) | Bill | 9 | 90' | 20 (estimated) | 8 projects (11 students) |
| IV (3rd) | Carl | 7 | 180' | 30 (estimated) | 12 projects (14 students) |

Classes ended before the Christmas holidays, but the final project presentations were set for early January. We were invited to attend a day of presentations, not listed in the above table, which we shall report in subsection §2.5.

### 2.3.1 CLASSROOM CONDITIONS

All sessions we attended took place in the same room, a large rectangular studio about 14 meters long and 5 meters wide. Smaller desks had been put together in two large tables were placed along the center of the studio: one rectangular, about 6 meters long and 2 meters wide, another square, about 2.5 meters on each side. These tables featured electrical outlets and were tied together, so their arrangement was understood as semi-permanent. We produced illustrations based on our rough sketches and measurements, which we present in Figures 7.2.1 and 7.2.2.

Most equipment, including eleven desktop computers and respective monitors (but also three printers, three flatbed scanners, photographic enlargers, and portable light tables), was placed in workbenches set against the long walls. A computer was also provided for the professor and was placed next to its set of speakers at the head of the larger table, closest to the end wall with a projection screen. As illustrated in Figure 7.2.1, the screen was not large enough; we felt most text was unreadable from further than halfway across the classroom, even with good eyesight.

***Figure 7.2.1.*** *A general view of the classroom environment.*

The classroom's spatial arrangement gave many students no choice but to sit with their backs to the professor. During our observations, professors had some difficulties keeping track of the students inside the classroom (Figure 7.2.2).

***Figure 7.2.2.*** *Many students sat either far away or with their backs to the professors.*

There were often over forty persons in the classroom, and not all of them were attending PRIMER, as other students at the faculty could use the studio's equipment even during sessions. These students tended to sit around the smaller table further away from the professor's station, often drawing with the aid of animation devices such as light tables or operating flatbed scanners. The door was often ajar, which led to a chaotic and noisy classroom environment filled with constant movement.

### 2.3.2 PROJECT SUPERVISION

As summarized in Table 7.2.1, we observed as the professors went around the classroom and engaged in dialogue with individual students or groups about their work some 63 times—40 such interactions took place during three longer sessions with Alan and Carl in week 7, and the remaining 23 interactions were observed in week 9, during three shorter 90-minute sessions with Alan and Bill. We observed 78 individual students in dialogue with their professors. Though most conversations took place in Portuguese, in one instance, Alan discussed the project of a pair of students with hearing loss with the help of their Sign Language interpreter, and in another instance, Bill conversed in English with an Erasmus mobility student.

Most interactions concerned individual projects (51), though we also observed the professor engage with pairs (6) and groups of three students (6).* As expected from class composition, the professors had a greater engagement with female students (56 working individually and in groups) than male students (22). Most conversations we witnessed lasted fewer than fifteen minutes, though there were a few that could be described as short (9), lasting up to a couple of minutes, and some that can be described as long (7), lasting up to thirty minutes. In addition, though we made repeat visits to sessions with classes I and III, there were only three instances in which we saw professors reengage, during week 9, with students they had talked to earlier during week 7.

We explored our observations using the Atlas.ti qualitative analysis software and found cooccurrences that confirmed our first-hand impressions in the classroom. As it can also be surmised from Table 7.2.1, Professor Alan supervised more group projects (7) than individual projects (6). We found this to be in contrast to what we observed in the sessions led by assistants: Bill was more engaged in dialogue with students developing individual projects (14) than with groups (4); apart from a single group project, Carl only supervised individual works (31). As mentioned, first and second-year students were assigned to classes I, II, or III depending on the most convenient schedule in relation to their elective curricular units, so it is likely the professor leading their class influenced students' choice to forge ahead with an individual project or to work in a group. Indeed, during week 7 we witnessed Alan asking undecided students to work in groups, while Carl did not issue a similar request to his students.

Many conversations we witnessed concerned students' ideas for their final project (28), though few such conversations involved the discussion of the proposal documents students had been required to submit (3). Some students also showed outside examples of works they considered inspirational while discussing their ideas (6). Predictably, we observed all but one of these discussions about project ideas during week 7. We also witnessed many discussions about the practical project development issues faced by the students (31), often in tandem with the discussion of a working prototype (19), as well as conversations concerning projects' aesthetics (9) and, in one instance, task management issues. Some students showed artwork they had made for their projects (9) or relevant programming experiments (4).

* These numbers do not line up exactly with the number of students observed as there were students absent from their own group's project discussion.

Many ideas and prototypes we saw being discussed concerned game development (28)—especially 3D environment exploration (11), puzzle (7), and platform games (6). Most of the other projects concerned visual artworks (22)—mostly animations (10) and data visualizations (7)—though we also witnessed discussions about audio projects (5). Most discussions also referred to one specific programming tool and were more or less evenly split between Unity (30) and Processing (27); the former was favored for game projects (22), whereas the latter was favored for developing visual artwork (19) and audio projects (5). Though we did not witness discussions relating to all the projects under development, the types we saw at this point matched our later observations during the final presentations, which we will report in §2.5.

✦ STUDENTS' QUESTIONS AND DIFFICULTIES Several students at the earlier stages of projects' development asked professors whether their idea was feasible (7), whether Processing or Unity would be the best tool (5), and how to begin (7). We also witnessed students asking for examples of possible projects (3). In one instance, a group expressed concern with how to manage their tasks. Some students said they felt their knowledge of programming was insufficient (4), going as far as, in one case, confessing they wanted to avoid programming. Others said they were confident they understood the basics but had trouble understanding complex examples (3); another individual student admitted having trouble following a step-by-step tutorial. Still, other students said they were fully confident in their own abilities (5), regardless of their projects' complexity.

Professors were able to address most questions and provide suggestions regarding students' ideas, recommending not just whether to use Unity or Processing, but also useful libraries and assets (3) or instructive tutorials (4). In a few instances, professors criticized ideas as too complex (5), recommending lower expectations (4) or advising students to collaborate with another group (1). In contrast, other project ideas were criticized as too easy to implement (2). We also witnessed professors advising groups to rethink their composition or the roles they had assigned to each element (2).

Some students who were already developing their projects had questions about how to code some part from the beginning (13) or about getting their code to work as desired (12), often when facing bugs or crashes. Students also posed questions about tasks unrelated to programming (8), most often about 3D and scene design in Unity. Such conversations about the completion of simple tasks in Unity tended to be short (6) as the professors could usually provide an answer in two minutes or less. Conversely, discussions of programming in Processing often took more than fifteen minutes (5).

For the most part, professors were able to steer students toward pursuing solutions themselves. In most instances, professors would clearly solve the students' problems by the end of their discussion (13). On three separate occasions, students were also asked to read a resource available on Moodle. On another occasion, Professor Alan asked the student to write a post to the Moodle forum about their problem and its solution so that colleagues could benefit. Still, we witnessed a few instances in which the professors were clearly unable to find a solution to the students' problems (3). In one such situation, the student was asked to rebuild their code during the remainder of the session; the other two were asked to send the professor an email reminder about their problem.

✦ PROFESSORS' EXPLANATIONS The manner in which professors provided their explanations warrants some notes. They often offered theoretical explanations on how students should structure their code (10) rather than advancing more concrete steps (3). Though in one instance Professor Alan used pen and paper, all three professors were willing to take command of students' computers (11), most often their personal laptops, and themselves code the solution to the students' problem (8). We also saw professors pulling up the available documentation for either Processing or Unity using the web browser in the students' personal computer (4). In two of these situations, the professors failed to arrive at a solution and were themselves seen struggling to code (2)—though, in fairness,* it may be the case that students' low-quality code was hard to follow within the constraints of a discussion taking place in a loud and crowded classroom.

The professors also asked questions of their own. Discussing game projects, we saw students were asked to clearly describe the game mechanics (4) or the narrative (1). Professors asked students to clarify whether they were reusing source code found online and where they had found it (2), as well as to clarify their use of ready-made graphics assets for Unity. Still, such inquiries were infrequent. Only once did we observe a professor asking a student to read the Processing language reference and explain aloud how they would use one of the functions. We also witnessed a single instance in which a student was asked to explain aloud the algorithm for their project (a puzzle game).

✦ NOTABLE INCIDENTS We observed other notable incidents during the discussions between students and their professors. We witnessed three different situations in which the students visibly waited for a long time, not engaging in any other class-related task, before the professor came to talk to them. We also witnessed one situation in which a student claimed she had not received any reply to her email regarding a coding problem posed in an earlier session. A few students also told their professor they had trouble understanding their explanations, forcing him to repeat himself (3). In another situation, we heard the professor being asked to explain basic principles we expected to have been addressed at the beginning of the semester (though it was unclear whether the student had attended those sessions). And in yet another situation, we felt the solution the professor wrote in the students' computer, though it solved the problem, could be considered inadequate (repeating statements instead of using a loop).

### 2.3.3 PROFESSORS' BEHAVIOR

During our observations, we found differences between the behavior of Professors Alan, Bill, and Carl, which are worth summarizing:

✦ ALAN Unlike Bill and Carl, Professor Alan addressed the entire class at the start and the end of each session with news and general information about the curricular unit, always emphasizing that students should check out Moodle periodically; he also talked about any new resources he had shared on the platform. In addition, Alan also carried out a roll call halfway through his sessions.

* And according to the researcher's own experience in similar situations.

While supervising students' projects, Alan appeared more comfortable than his assistants in providing students with ideas and examples. As his assistants, Alan was likely to use his students' personal computers during his explanations. Still, the professor seemed more at ease with Unity rather than Processing.

✦ BILL AND CARL Unlike Alan, the assistants did not conduct roll calls nor addressed their classes to make announcements. Though it is understandable that Alan was the responsible professor for PRIMER and Bill and Carl had just arrived as assistants, we note that Bill and Carl were the sole docents for three of the classes. As we saw in §2.3.2, there were differences between the projects developed by Alan's students and Bill and Carl's students, starting with the former students' greater willingness to propose group projects, responding to Alan's specific call on the subject.

Unlike Alan, both Bill and Carl seemed more comfortable with Processing rather than with Unity. We also believe that Bill, perhaps under the pressure of students asking for his attention during the 90-minute sessions, was rather quick to ask students to use their computers and attempt to solve their coding problems all by himself, especially if those problems concerned Processing. Carl, whom we saw during 3-hour sessions, was also likely to use students' laptops, though we also witnessed long verbal explanations; we note we observed those sessions during week 7 when many discussions still concerned the feasibility of students' ideas.

✦ ALAN'S VIRTUAL REALITY DEMONSTRATION Unlike his assistants, Professor Alan dedicated one hour of his week 7 session to class I to demonstrate RED's Oculus Rift headsets and how to use Unity to develop Virtual Reality (VR) content. The brief demonstration was conditioned by the space that was available in the classroom and the relatively small projection screen. As most students remained in their seats working on their projects and ignored the lecture, only eleven gathered around the professor, some bringing chairs. After Alan briefly discussed the history of Virtual Reality, students were allowed to play with the VR headset, at times with genuine reactions of laughter from the colleagues. Alan also put on the headset himself, and he too prompted good laughs from the students following his movements. This attracted a couple more students to try the headset, though they did not stay for Alan's explanation of how to configure Unity to develop VR projects. The professor explained students could develop such projects on their own and then try them on the faculty's equipment.

### 2.3.4 STUDENTS' BEHAVIOR

We witnessed a set of behaviors from students during our observations, regardless of their class or the professor leading the session, which we thus summarize:

FIRST, students did not occupy the space of the classroom evenly. Students usually sat in groups, often separated by gender, and regardless of possible affinities with the projects of colleagues sitting close by. As many students were already in their second year or third year at the faculty, it is likely their preexisting friendships determined their sitting arrangements or their group for the project. Most students chose to work on their laptops, so they were not bound by the placement of the classroom PCs.

SECOND, most students paced their work. We often saw students taking breaks and using their phones, browsing social networks or websites not related to class, or in conversation with colleagues. Still, such frequent distractions did not stop most students observed from either working on their projects or following tutorials.

THIRD, online tutorials played a major instructional role. Some students followed written tutorials while others followed video tutorials, which often required them to wear headphones. Some students followed tutorials shared by the professors on Moodle, while others followed tutorials they had found to meet the needs of their projects.

FOURTH, fewer students attended the ninety-minute sessions, as we can surmise from Table 7.2.1. On the one hand, the shorter sessions we observed were scheduled either in the early morning or in the late afternoon. Given that both Bill and Carl did not make roll calls, it may be the case their students felt more at ease in skipping class. On the other hand, these students had a certain justification for feeling they would have a lower chance of posing questions or discussing their project with the professor during the shorter sessions. Class sizes were too large; even when many students were absent, the professors were still unable to meet each student or group attending a session.

### 2.4 *Online learning management and educational resources*

Given the transitional curriculum that brought an exceptional number of students to attend this particular instance of the curricular unit, the responsible professor and his assistants attempted to implement—though never explicitly naming it as such—a blended learning methodology in which the Moodle learning management system took on a key role. However, RED denied us guest access to Moodle, though Professor Alan provided us with a set of hardcopy prints he took from the principal areas in the unit's Moodle web platform. According to those printouts, PRIMER's learning management platform was organized as follows:

- HEADER SECTION, which adhered to the Moodle installation defaults, with an external link to the curricular unit plan, followed by links to a chat room and two forums: one for news and announcements, and another for general discussion. We are unable to know whether the chat room was used, though we were provided with prints of the forums' index pages, which include each topic's title and author. The announcements forum only included topics started by Professor Alan, mostly concerning submission rules and dates for the project proposal and presentation; all listed as having zero replies. The general discussion index listed one single topic concerning '3D modeling' created by Professor Alan, with a single reply.
- PROJECT PROPOSALS SECTION, which included a forum for project submission and discussion and a page listing all approved projects. We were not provided with any additional printed material concerning these two links, as to protect students' anonymity. From some conversations we witnessed during our observations, we could gather that the project submission and discussion forum was the most active area of Moodle use and that this forum was, alongside email, a key instrument in professors' support of the final projects' development.

- RESOURCES SECTION, which included one link about the software tools and licenses available on campus, as well as a folder titled 'Resources' and a page titled 'Web references,' from which nothing else could be inferred.
- DIGITAL ARTS AND MULTIMEDIA SECTION, which also included a nondescript folder titled 'Artist resources' alongside a page simply titled 'Artist selection.' The section included two external links with titles matching well-known webpages, Thomas Dreher's *History of Computer Art* and the DAM Digital Art Museum.*
- TYPES OF INFORMATION SECTION, which included two external links with titles concerning 'Audio and video coding' and 'Image compression.'
- INTRODUCTION TO PROGRAMMING SECTION, which was in turn divided into two subsections. The first concerned 'Mark-up and Stylesheet Languages' and included links to multiple external webpages simply titled 'HTML,' 'HTML tags,' 'CSS,' and 'KompoZer.' The second concerned 'Programming Languages' and included links to Wikipedia and a 'Java Tutorial.' Given that these subjects are not mentioned in PRIMER's syllabus and that KompoZer is a very outdated and long unmaintained webpage editing software, it is highly likely this section resulted from the migration of a previous instance of the curricular unit, a common practice when setting up Moodle for a new academic year.
- AUTHORING SYSTEMS & GUIDED EXPLORATION SECTION, which was in turn split into two subsections as well, the first concerning Processing and the second concerning Unity. The first included links to external websites: processing.org, Daniel Schiffman's† *Coding Train* YouTube tutorials, and Professor Carl's GitHub profile, and his webpage about Processing. The Processing subsection also included a support forum, from which we were also given a print of its index, though it listed a single topic started by a student (and simply titled 'Question') with two replies. The Unity subsection followed a similar structure, with links to unity3d.com followed by links to video tutorials, though these were just deep links into the official site. The subsection also included a support forum; its index listed six topics started by Professor Alan concerning simple questions (e.g., 'How to place a video in a texture?'), some of which we witnessed being posed by students during the sessions we observed. Apart from the professor's, there was just one single question topic started by a student in the Unity forum.
- MISCELLANEOUS SECTION, which included a link to the website of the Blender 3D modeling software and a link to a page concerning 'Blender examples.' This section also included a page with examples of Game Design Documents (students developing game projects had been asked to use that format in preparing their project proposals). Finally, the section also included a folder of 'Other resources'; we were also given a printout of the index, which listed six PDF documents with titles concerning filmmaking techniques and comic book design, presumably to aid students developing their ideas.

---

* iasl.uni-muenchen.de/links/GCA_Indexe.html, dam.org

† shiffman.net

As it can be surmised, PRIMER's Moodle was not tightly structured. Parts appeared outdated and were probably left from a previous instance of the curricular unit. We also felt some resources were scattered throughout the homepage, while others were placed inside folders for which we were not provided with the indexes. From what we were given to analyze, there were few links to educational resources about Processing or Unity, though it is probable more resources had been shared through the crucial project discussion forum, which we were unable to review. We had the impression that forum concentrated much of the online communication between students and the professors. We note that the printouts we were provided did not mention participations by either Bill or Carl, except the links to the latter's GitHub and webpage.

## 2.5 *Students' projects*

According to Professor Alan, students working individually and in groups submitted 120 proposals to PRIMER's project forum on Moodle, most complying with the deadline that had been set for week 8. Students were required to upload their completed projects to Moodle by a date set for early January, a few days before the presentation sessions. Still, the presentation was mandatory, as students were required to show their projects to all three professors in order to obtain a grade. Hence, 108 projects were scheduled for presentation in four sessions that took place over two days. Professor Alan told us that the remaining projects had not been submitted.

We were invited to attend the first day of project presentations, with the morning and afternoon sessions taking place in the same studio as the classes (see §2.3.1). We saw no discernible criteria for the order of presentations, which were unrelated to students' previous class assignments. Nonetheless, there were between thirty and forty students in the room at the beginning of each session.

Professor Alan told us the presentations were mostly confirmatory, as he and his assistants had already seen and discussed students' submissions. Presentations were thus quite informal. The three professors moved to where each student or group was sitting and asked them to present their work on either their computers or the classroom PCs. In cases in which audio quality was relevant, students picked up their computers and took them to the head table to connect them to the studio's speakers. We felt this procedure distanced the remaining students from their colleagues' work, though we recognize it was time-efficient, given the many projects to be seen. Most students were seen chatting with their colleagues as they waited for their turn to present; only a few were seen completing last-minute testing. The room was noisy at the beginning of each session, yet it grew silent as most students chose to leave after presenting their project.

✦ PROJECT CHARACTERISTICS We witnessed 46 presentations of 34 individual works and 12 group projects; the latter either developed by pairs (5) or groups of three students (7). Only four of these groups were comprised of students of mixed genders. Overall, 64 students (49 female and 15 male) participated in the projects we saw being presented. One student presented two projects—one individual work and one group project with two other colleagues.

Many groups and students developed games (26 projects), notably platform (10), exploration (8), and puzzle games (5). Another large category could be described as visual works (17) and included animations (6), visualizations (6), drawing/design tools (3), and generative visuals (3). Several other projects dealt with audio sequencing and playback (5), multimedia narratives (3), and augmented reality (1). Some projects met more than one topic: e.g., a game that generated music as the player progressed, another game that articulated two- and three-dimensional graphics in two separate screens, or an audio sequencer that produced a graphical visualization at the same time.

We also categorized projects according to their formal characteristics. Most employed two-dimensional graphics (35 projects), but several used three-dimensional graphics (12). Some students also explored audio (5) and video (3) processing and output in their projects, as well as augmented reality mobile development (2), simultaneous use of two screens (1), virtual reality (1), or site-specific art installation (1). Though most projects were developed in either Processing (24) or Unity (21), two projects did not employ one of the tools recommended in the syllabus: one individual work was done in the Visual Basic programming language, and a pair developed theirs with WordPress, a website content management system.

We explored our categorizations for co-occurrences using the Atlas.ti software analysis tool. Although many female students were involved in game projects (19), a sizable proportion was involved in visual works (16); in contrast, their male colleagues overwhelmingly favored games (11 projects) over visual works (1). We also saw an almost equal number of individual projects concerning visual works (16) and games (15), while a strong majority of group projects concerned game development (11). Students working individually were more likely to be preoccupied with two-dimensional graphics (in 28 projects) than three-dimensional graphics (7), while group projects were more evenly concerned with 2D (7) and 3D graphics (5).

We found an undeniable relation between the tools employed by students and the type of projects they developed. Unity is marketed as a 'game engine', even though it is a complete integrated development environment, and indeed it was used by students to build other types of projects, such as the two augmented reality mobile applications we mentioned. Still, the remaining Unity projects were all games (19); around half of which employed two-dimensional graphics (10). In contrast, a sizable proportion of Processing projects concerned visual works (16), though some also developed games (5) and audio projects (5). Except for a single project using 3D graphics, Processing projects employed two-dimensional graphics (23). Processing was favored for audio projects (5), as well as most works that employed means of interaction other than the mouse or keyboard, such as a microphone (4) or a webcam (1).

We may set aside, for now, the discussion on whether it is students' project type preferences that determine which tool they choose or whether it is the other way around. Male students were more likely to be involved in projects made with Unity (9) than Processing (2), while female students were somewhat more likely to be involved in projects using Processing (in 22 projects) than Unity (15). Unity was also much more likely to be used in group projects (10), while most individual projects were developed using Processing (23), though a sizable number of individual students used Unity (11).

✦ PRESENTATIONS AND DISCUSSION Most presentations were quite brief, and most of the ensuing conversations between students and the professors concerned superficial aspects of the work. As mentioned, most presentations we confirmatory; we gathered the impression that the jury knew the questions they were going to ask before students presented their work. Nonetheless, we witnessed one situation in which the jury had to troubleshoot a student's difficulties running their program. We also felt most students lacked confidence in their work, as we only witnessed three presentations in which the students—all of whom were female and had done individual projects—expressed self-assurance. In an informal conversation with the researcher during a break, Professor Alan said he indeed felt the students lacked presentation skills. Nonetheless, we believe the informal environment, the overcrowded room, and colleagues' lack of interest did not favor students' showmanship.

Some projects were characterized by their authors as incomplete prototypes (8); one such student, working individually, admitted to giving up on a previous project. Several students admitted facing difficulties in developing their work (9 occurrences); most concerned programming (5), but some had difficulties with the Unity software (2), understanding how to use a certain library in Processing (1), or optimizing a Processing sketch that ran slowly (1). Though we saw more projects that had reused code or third-party graphical assets, few students acknowledged it in their presentations; some indeed admitted to reusing code from examples or tutorials (7) or reusing visual assets (2). Some of the students that had worked solo were candid about their projects: one said she thought it took too much time to develop, another said she saw it as nothing more than mandatory coursework. Still, another student talked about how her project could improve, and another stated she intended to use Processing in future projects.

From the three professors, we witnessed 20 instances in which they clearly praised students' work and 17 instances in which they clearly criticized it. Students were praised for their ability to code and solve problems (10), for their projects' attention to detail (3), or for their autonomy (2). In individual instances, the jury also praised a student for their work documenting their project's development, for the mechanics of their game project, for the completeness of their project, for the motivation throughout the semester, or for how the students were able to demonstrate their command of programming. In a few instances, the jury generally praised the projects (3) without specifying notable facets.

Conversely, the professors also criticized students for avoiding programming (4)—in one such case noting there had been no programming—as well as for reusing most of their code (3), for showing little attention to detail (2), for work they found 'simplistic' (2), or for 'wasting' an idea with greater potential (2). The jury criticized one student's project for lacking the desired interactivity and another student's project for its little progress since first seen. The professors also criticized a group that had shown hesitations during development and another group for admittedly lacking enthusiasm.

The jury also made remarks to students that were not loaded with any particular praise or criticism. Often these concerned projects' aesthetic or artistic merit (10), which were not under evaluation in PRIMER. The professors offered neutral remarks regarding game projects' mechanics (6) and narratives (2), as well as on the projects' programming (5) and reuse of third-party content (4). The jury also made comments about possible improvements (5) or future project possibilities (5).

## 2.6 *Students' acceptance and use of computer programming*

Per our instructions, Professor Alan used Moodle and institutional email to forward students our survey questionnaire (see **5**.4) at the beginning and at the end of the semester, following the procedure described in **5**.5.1.5. Due to the delay faced in obtaining permission to conduct the study, the first application took place during the fourth week of classes, a moment we nonetheless considered acceptable. Still, we were disappointed only 76 students (42%) completed the first survey application. Of these participants, however, only ten completed the second survey application during the final presentation week. Though that prevented robust longitudinal analysis, we still matched these students' responses to their first survey and conducted tentative comparisons after testing the differences for normality. Still, neither Paired Samples t-tests nor Wilcoxon Signed Ranks tests revealed significant changes in those ten students' acceptance and use of computer programming at the end of the semester.

⁂

We will, therefore, focus our analysis on students' responses to the first questionnaire, completed four weeks in the semester (*n*=76). At that point, 30 participants (39.5%) claimed prior familiarity with computer programming, while 40 participants (52.6%) submitted they were learning to program for the first time. Only six (7.9%) submitted they had no knowledge of computer programming. In Table 7.2.2, we present the first survey participants' gender, age group, seniority (year at the course), and self-assessed daily computer use, grouped by their programming knowledge.

***Table 7.2.2.*** *Students' characteristics and familiarity with programming (week 4 survey).*

| | | *No knowledge* (*n*=6) | | *Learning* (*n*=40) | | *Prior knowledge* (*n*=30) | | *Total* (*n*=76) | |
|---|---|---|---|---|---|---|---|---|---|
| | | *n* | % | *n* | % | *n* | % | *n* | % |
| *Gender* | Female | 5 | 83.3 | 29 | 72.5 | 24 | 80.0 | 58 | 76.3 |
| | Male | 1 | 16.7 | 11 | 27.5 | 6 | 20.0 | 18 | 23.7 |
| *Age group* | 18 – 20 | 5 | 83.3 | 37 | 92.5 | 23 | 76.7 | 65 | 85.5 |
| | 21 – 24 | 0 | 0.0 | 3 | 7.5 | 5 | 16.7 | 8 | 10.5 |
| | 25 – 34 | 1 | 16.7 | 0 | 0.0 | 1 | 3.3 | 2 | 2.6 |
| | 35 + | 0 | 0.0 | 0 | 0.0 | 1 | 3.3 | 1 | 1.3 |
| *Seniority* | 1st year | 2 | 33.3 | 20 | 50.0 | 12 | 40.0 | 34 | 44.7 |
| | 2nd year | 4 | 66.7 | 17 | 42.5 | 15 | 50.0 | 36 | 47.4 |
| | 3rd year | 0 | 0.0 | 3 | 7.5 | 3 | 10.0 | 6 | 7.9 |
| *Daily PC use* | < 1 hour | 0 | 0.0 | 2 | 5.0 | 2 | 6.7 | 4 | 5.3 |
| | 1 - 2 hours | 0 | 0.0 | 6 | 15.0 | 3 | 10.0 | 9 | 11.8 |
| | 2 - 4 hours | 2 | 33.3 | 11 | 27.5 | 12 | 40.0 | 25 | 32.9 |
| | 4 - 8 hours | 3 | 50.0 | 19 | 47.5 | 9 | 30.0 | 31 | 40.8 |
| | > 8 hours | 1 | 16.7 | 2 | 5.0 | 3 | 10.0 | 6 | 7.9 |

We conducted the appropriate analyses given the sample and found no relationships between participants' gender, age, or course year, and their reported programming knowledge or daily computer use, or between these two latter items.

Our survey form was designed to present additional questions to participants claiming familiarity with programming ($n$=30). Of these, six described their knowledge of the subject as 'reasonable,' whereas 24 admitted it was 'basic.' When asked to report the places and situations where they had learned to code, twelve participants mentioned secondary school, and five mentioned another higher education course; vocational school, paid lessons, and self-education were each mentioned once. In addition, though not all eligible participants answered that question, eight mentioned being familiar with Java programming and four specified Processing; JavaScript, C++, and Visual Basic were mentioned three times; C and C# were mentioned twice, while Matlab, Pascal, PHP, and Python were each mentioned once.

### 2.6.1 STUDENTS' ACCEPTANCE AND BEHAVIOR

We computed means and standard deviations for the acceptance constructs' that tested as reliable, leaving out any problematic items as described in 5.5.3, and which we present, grouped by participants' familiarity with programming, in Table 7.2.3. We transposed the 7-point Likert scales to run from -3 (standing for full disagreement) to +3 (standing for full agreement). We note that, with the exception of *Anxiety,* positive values reflect a more favorable opinion.

***Table 7.2.3.*** *Students' programming acceptance, grouped by familiarity (week 4).*

| | *No knowledge* ($n$=6) | | *First time learning* ($n$=40) | | *Prior knowledge* ($n$=30) | |
|---|---|---|---|---|---|---|
| | Mean | St. dev. | Mean | St. dev. | Mean | St. dev. |
| *Performance Expectancy* | 0.39 | 1.10 | -0.18 | 1.58 | 0.53 | 1.46 |
| *Effort Expectancy* | -1.21 | 1.13 | -1.31 | 0.93 | 0.04 | 1.36 |
| *Attitude Toward Use* | -0.08 | 1.61 | 0.05 | 1.34 | 0.83 | 1.44 |
| *Subjective Norm* | -0.83 | 2.41 | -1.23 | 1.64 | -0.50 | 1.71 |
| *Self-Efficacy* | -0.29 | 1.11 | -0.02 | 1.29 | 0.86 | 1.12 |
| *Anxiety* (reverse scale) | -0.61 | 1.39 | 0.08 | 1.44 | -1.07 | 1.42 |
| *Behavioral Intention* | -0.28 | 1.73 | -0.53 | 1.73 | 0.78 | 1.70 |

We asked participants either learning for the first time or already familiar with computer programming ($n$=70) how they perceived the voluntariness of its use in PRIMER. We found a Point-biserial relationship in which male participants were more likely to regard programming as voluntary ($r_{pb}$=.19; $p$=.020). Though we did not find such an association among participants reporting prior familiarity with code, male first-time learners were more likely to perceive voluntariness of use ($n$=40; $r_{pb}$=.43; $p$=.007). Mann-Whitney U tests yielded similar significance levels, confirming the relationships.

Kendall tau analyses yielded a compelling correlation between first-time learners' greater perceived voluntariness of programming and better *Effort Expectancy* ($n$=40; $\tau_b$=.42; $p$=.001). Among those familiar with programming ($n$=30), we found compelling correlations between their perceived voluntariness and better *Performance Expectancy* ($\tau_b$=.50; $p$<.001) and *Attitude Toward Use* ($\tau_b$=.51; $p$<.001).

Among students learning to code for the first time, we found moderate Pearson correlations between older age and a less favorable *Behavioral Intention* ($n$=40; $r$=-.36; $p$=.035). Yet, we found moderate positive correlations between these students' perceived voluntariness and their such *Intention* ($\tau_b$=.30; $p$=.015). We did see a similar correlation among experienced students, though above the .05 significance threshold.

✦ CORRELATIONS Our Pearson 2-tailed analyses found strong correlations between measures of participants' programming acceptance and their *Behavioral Intention*, as suggested by the UTAUT model (see 5.2.3) and presented in Table 7.2.4. We may add that no significant associations were found involving participants' reported *Anxiety*.

***Table 7.2.4.*** *Significant Pearson correlations between students' acceptance and their Behavioral Intention (week 4).*

| | *First time learning* ($n$=40) | | *Prior knowledge* ($n$=30) | |
|---|---|---|---|---|
| | *Correlation (r)* | *Sig. (p)* | *Correlation (r)* | *Sig. (p)* |
| *Performance Expectancy* | .63 | < .001 | .64 | < .001 |
| *Effort Expectancy* | .52 | .001 | .48 | .008 |
| *Attitude Toward Use* | .72 | < .001 | .78 | < .001 |
| *Subjective Norm* | .52 | .001 | | *Not significant* |
| *Self-Efficacy* | .55 | < .001 | | *Not significant* |

Using the appropriate analysis, we also looked for correlations between participants' intention and their self-reported computer programming use. Among students already familiar with programming, their *Behavioral Intention* clearly correlated with frequency ($n$=30; $\tau_b$=.40; $p$=.008) and intensity ($\tau_b$=.43; $p$=.003) of computer programming. At about the same time they completed the first survey application, participants had been given programming exercises as homework. Hence, we also found conclusive correlations between first-time learners' *Behavioral Intention* and use frequency ($n$=40; $\tau_b$=.38; $p$=.003) and intensity ($\tau_b$=.40; $p$=.001).

## 2.7 *Interviews with the professors*

At the end of the term, the researcher requested individual interviews with Professors Alan, Bill, and Carl, which followed the procedure laid out in 5.5.1.4. Professors Alan and Carl granted interviews via Skype video conferencing a couple of weeks after the end of the semester. Professor Alan spoke with the researcher for nearly ninety minutes, while Professor Carl spoke for about forty minutes. Professor Bill only consented to an email interview and took some time to respond to our questions; therefore, Bill's answers were more structured and concise than his colleagues'.

✦ LEARNING TO CODE AND DIGITAL LITERACY In Professor Alan's long career as an educator, he found students had a general difficulty with logical expression; though more evident when teaching computer programming, the professor said the gap is

noticeable in other areas. He was afraid the focus on digital literacy distracts from the real problem of needing to *"go through Cartesian thinking"* first, which he believed is missing from the Portuguese educational system. Alan noted few students were aware of lacking logical thinking skills; moreover, many students did not feel that addressing that gap would be important for their practice.

Professor Bill noted digital literacy is part of the wider socio-cultural literacy, yet he believed computer programming is an important but rather specialized element. He regarded the value of computer programming to arts students as an open matter, noting many students at the AMP course were insensitive to the value of learning to code. Still, Bill also believed schooling ought to promote contact with programming at some point, as to grant students the opportunity to discover and develop a vocation.

Professor Carl agreed with Bill that computer programming is a specialization. Still, he went further and considered that familiarity with code is essential for digital media practitioners. He criticized those only using ready-made software, noting that students and artists that understand and can manipulate the underlying algorithms are able to take their work to a higher level.

✦ HIGHER EDUCATION AND THE CURRICULUM Professor Alan took some time to discuss his views on higher education and his stance toward the revised curriculum. Bill and Carl, who recently arrived at the faculty, were more circumspect about these matters. Alan believed there is a contradiction at the heart of higher education in the present—a desire for *"broadband but specialized courses."* As the professor put it, two fallacies have contradictory demands on curricula: on the one hand, a need to meet volatile job market requirements; on the other, an understanding that students ought to be *"prepared for everything,"* given the trend away from lifelong employment. Alan believed that nowadays, *"people are often forced to choose [their course of study] when in fact they haven't yet made up their minds, and that shouldn't be considered a bad thing."*

Alan believed wider curricula were desirable within higher arts education and that students ought to be given greater contact with the sciences. He explained the divide starts in earlier schooling, giving the example of Multimedia students in secondary schools being taught Mathematics 'B' disciplines that leave them unprepared and unable to apply for Engineering courses at a university. The professor lamented the absence of Mathematics and Physics from the AMP curriculum, noting how they would relate to the work made within the course—e.g., how linear algebra and analytic geometry relate to 3D modeling or how trigonometry relates to computer graphics or signal processing. Alan *"does not understand why would [students] be denied part of the knowledge,"* and noted it would be equally beneficial for students in the Sciences to learn Art History. The professor admitted *"there is a difficult equilibrium, but it is not impossible at all"* to implement such interdisciplinarity in the curricula.

Professor Alan explained the new AMP curriculum tried to follow the evolution, *"not of the market, but of the students that arrive,"* stating, *"I read the book, those after me saw the movie, and now these guys played the game."* Still, Alan emphasized the need for maintaining the faculty's unique characteristics: within RED, 'multimedia' is not to be necessarily understood as referring to digital multimedia, but rather in the tradition of intermedia art dating back to the 1960s. As an example, the professor explained, RED

would be the right place for anyone *"wanting to make a multimedia installation with tape recorders and a slide projector."* Still, he reckons that computer art is a field approaching 70 years of existence, and thus the need to address digital art is undeniable.

The main objectives of including a unit like PRIMER in the curriculum were, according to Alan, to *"create a profound awareness"* of technical possibilities and to address any prejudices about technology students may bring. He believed the decision to move students' first contact with programming from the third year to the beginning of the AMP plan of study brought many advantages, as it empowers students to explore the tools and techniques more thoroughly throughout their studies.

Professor Carl told us he believed the faculty had been nevertheless conservative in maintaining most content related to computer programming in a single curricular unit. He hoped future curricular revisions would introduce more specific units, giving examples of physical computing or game design. Professor Alan advanced that new elective units would be offered in the coming year, including one concerning physical computing. Still, he recognized the new curriculum could have gone further, and ideally, PRIMER would have been split into two separate units—one solely dedicated to computer science foundations and another dedicated to project development.

✦ THE UNIT PLAN Professor Alan also felt more at ease than his assistants to discuss his views about PRIMER's syllabus and session planning. He emphasized the need for *"theorizing by making and making by theorizing,"* in which practical experimentation and mastering the tools are crucial and go hand in hand. He criticized *"learning tool after tool,"* and noted the need for balance in choosing those that best support the concepts to be taught. Alan explained that Processing's ease of use, large user base, and relation to the popularization of creative computing make it a good language for teaching the basics of programming.* He added he liked Unity because of its balance between visual representation and mathematical analysis. The professor pointed out that both Unity and Processing, despite their differences, obey the same 'film projector' paradigm.†

When Alan arrived at RED, the unit under discussion was still a third-year elective under the old curriculum. In his first year teaching, he had introduced HTML formatting during the first few sessions, so students got comfortable using a text editor while seeing immediate results, then moved on to computer programming proper with Processing, leaving Unity for the end. He told us he understood it was desirable to teach Unity first, starting with the 3D modeling and composition parts of the software. Students were comfortable with it because of their previous experience with design software, and in addition, that offered a way around their lack of scientific preparation. Professors could introduce Unity's C# programming afterward, and after that, Processing could be taught in parallel, given the similarities between the languages. Bill and Carl, newly hired for 2018/2019, were not as comfortable with Unity as with Processing; hence Alan and his assistants agreed to reprioritize Processing in this specific instance of the curricular unit, so that Bill and Carl had the time to get themselves up to speed on Unity.

---

* Adam noted that he does not consider Processing very efficient for professional work, especially in comparison to more robust tools such as OpenFrameworks.

† We take this comment as referring to how both tools render cyclical animations in which the contents of each frame result from the execution of a main loop function.

Carl confirmed it was the first time he taught Unity, though he had experience with similar 'game engine' tools and with 3D modeling software. In his teaching, he was used to emphasizing basic notions of programming, using Processing *"to teach how to create things from the ground up."* He respectfully disagreed with Adam, as he believed students should understand the basics before moving on to complex tools such as Unity.

Professor Alan wished to address how he sees PRIMER's syllabus evolving in the coming years. He often participates in the juries of Professional Aptitude Tests and has been witness to great progress in the IT competencies of the secondary school students that typically go on to apply to RED. Alan predicted that *"in two or three years we may not need to teach this unit starting with the tools from the beginning,"* as most students may arrive at the course with prior knowledge of Unity, Processing, or similar tools.

✦ THE WORKING CONDITIONS Professor Alan explained the key issues raised by the Higher Education Assessment and Accreditation Agency (A3ES) quality inspections concerned the material conditions. Noting most of those issues had been solved in the meantime, Alan admitted the studio was not appropriate to accommodate forty students simultaneously, noting the faculty is planning to build additional facilities.

Alan also noted that neither the faculty nor the course require students to own or bring their personal laptop computers to class. Though classrooms ought to be equipped with enough workstations to meet students' needs, he believed it would be unwise to *"invest in forty computers that sit unused,"* as most students do prefer to work on their own laptops. Alan explained the faculty's policy is now to purchase fewer but higher-end workstations, such as the computers supporting the Oculus Rift kits, as to provide access to unique equipment students are less likely to own. The professor told us the faculty had also recently acquired electronics kits and physical computing equipment.

Professor Bill told us his classes' sizes had been excessive. He thought there could have been distinct schedules for lectures and studio workshop sessions, which could run throughout the entire semester. Though agreeing, Professor Alan saw some virtues in large classes, as the systematization and discipline required of the professors benefited students. Still, he admitted some students were, in a word, *"perplexed."*

✦ INSIDE THE CLASSROOM We asked the professors about the first few sessions of the semester, which we had been unable to witness. Carl obliged and explained that during those first few sessions, students had been asked to learn by example and follow the professors' coding demonstrations, replicating the code displayed through the projector on their own computers. After each such demonstration, professors would circle the studio and check on students' exercises. During those first few weeks, students had also been asked to complete simple coding exercises for homework, which Professor Alan compared to *"learning the multiplication table."* Afterward, he went around asking individual students for what they liked; he made them understand that professors were merely demonstrating tools and that students were free to explore and to use to develop whichever project they desire.

Professors felt transparency about coursework was key to demystify the subject of computer programming. According to Alan, that principle led to their emphasis on maintaining an open studio environment—which he described as a *"natural workspace"*

—during project development and their final presentations. He also tried to persuade students away from their misunderstanding that professors were looking for particular objectives in their work. Alan was dismissive of issuing precise project instructions, explaining that professors should instead find out how students' own artistic interests can benefit from programming:

> One tries to understand what students want for themselves, putting some distance from the curricular unit, so then one can go back to the unit. – *Alan*

Though professors were unable to check on all students during the sessions we were able to observe, Professors Alan and Carl believed the studio sessions went rather well and that they were able to provide support to their students and discuss their ideas. Still, Carl admitted the size of the classes made tutoring exhausting. He noticed students were less likely to attend in the early morning, noting the absenteeism on Fridays. Professor Carl was also convinced some skipped class to avoid confessing their little work during the previous week. Unlike his assistants, Professor Alan was used to conducting roll calls, which he described to us in jest as a *"bad habit."* He believed that might lead students to feel a greater obligation to attend his sessions. Alan explained he still finds it strange to see students using their phones in class even though he understands students use their phones to take notes or document their work.

Carl told us a few students *"really hated programming, did not want anything to do with it, and were only there because of the curricular requirement."* He explained he persuaded those students of programming's relevance by portraying it as just another artistic technique like painting, sculpting, or drafting. Nonetheless, Professor Carl felt it should have been obvious for those students that they were going to need notions of computer programming to complete a course dedicated to multimedia production. Professor Bill told us he addressed his students' programming anxiety by keeping his demonstrations simple and reminding students that the unit *"is only an introduction and they don't need to sign a lifetime contract."*

According to Carl, some students became anxious because they found themselves 'stuck' while striving to obtain good grades. In such situations, he tried to find source code examples that were pertinent to students' ideas, and then he would read the code along with them, *"explaining how it works so they can adapt to their needs."* Professor Carl also had to manage the lofty ambitions of motivated students during the discussion of their project ideas, which he found was difficult to do without harming these students' motivation. In such situations, Carl told us, he would nudge students toward completing basic tutorials, so they slowly appreciated the complexity of their ideas. In later sessions, he would then recommend realistic directions and more specific references. Still, the professor praised students' initiative, which he considered essential.

Professor Carl admitted to lacking the Unity experience to answer some of the questions posed by students, in which case he helped them with their web searches. In contrast, he sometimes took the opportunity to demonstrate to students some of the Processing techniques he felt more comfortable with (e.g., audio-visual interactions or camera tracking). He said the demonstrations were fruitful as there were students that used those techniques in their projects.

✦ THE STUDENTS Professor Carl felt some students, despite their lack of experience, *"were dazzled by the technologies and really wanted to explore"* their potential to develop innovative projects. He saw this as a 'double-edged' situation, as one semester would not be enough to learn everything required. Professor Bill commended students' persistence and willingness to research, yet he noted most had been uninterested in thinking about historical computer art references. Professor Alan did not expect licentiate students to show much autonomy and, within typical variations, was pleasantly surprised by PRIMER's. Still, Alan noted the students admitted to the course had good grades in secondary school (see §2.1) and were likely effective and motivated. He also believed that second-year students, surprised by the mandatory requirement to attend PRIMER under the transitional curriculum, were less motivated than their colleagues. Professor Carl disagreed and believed they were as motivated as their colleagues.

Professor Alan told us that students over the age of thirty often struggled with digital technologies and *"sought refuge in analog subjects,"* though he mostly noticed these differences when teaching in master's degrees. He also noticed some differences among his licentiate students, noting those yet to turn twenty seem more effective and at ease with technologies than their colleagues just five years older. Still, Alan believed students are more immature nowadays, though noting that does not mean they are more irresponsible—rather than *"students are getting younger, as one is young for longer."* Though Professor Carl agreed, he emphasized that older students carry greater personal and professional pressures and cannot commit as much time to study. Writing specifically about PRIMER, Professor Bill told us he did not perceive any differences in attitude toward programming between students of different ages.

Alan also used to notice differences between the attitudes of female and male students, which he ascribed to historical and upbringing factors; the professor noted such differences had faded in recent years. Professor Carl was *"nicely surprised"* by the little difference in outcomes between genders and admitted such an appraisal shows his own bias in still thinking *"male students are more at ease in subjects like programming engineering."* Carl thought that female students, when struggling with programming, were likelier to take greater care with the aesthetics of their projects. Nonetheless, he was confident female and male students both stood on equal footing at RED, which had not been the case in the past at other institutions.

Professor Carl would *"risk a slight 'yes'"* to our question of whether there had been changes in students' motivation since he started to teach. He noted that when he gave his first programming workshops, around 2002, those who attended were highly motivated *"by a strong desire to make certain types of work they couldn't do with other technologies."* Carl still saw the same highly motivated students, and the evolution laid in the greater massification of computer programming. In contrast, Professor Bill told us his answer would be an emphatic 'no.' Still, Bill recognized that students interested in learning to code are increasingly autonomous because of the wider availability of information and resources online. Professor Alan was more skeptical and noted that younger students often do not know what to search. He told us that educators should focus on developing students' autonomy rather than *"battling all over the place."* Alan believed one way to do that would be to teach students *"how to find information, and how to make it 'information' rather than mere 'data'."*

✦ STUDENTS' PROJECTS AND EDUCATIONAL OUTCOMES Professor Alan explained that students had been required to post their projects to a Moodle forum instead of submitting them via the typical 'work submission' module. That way, students could see the work of their colleagues, according to Alan's principle of a transparent environment. The professor explained the presentation sessions were *"verifications of the projects submitted to Moodle,"* emphasizing they were meant primarily for the benefit of each student's colleagues. Alan wished one final opportunity to lead students to understand how their colleagues *"were facing the same problems and would benefit from greater socialization."* He also clarified that in previous instances, when the unit was an elective available to third-year students, the presentations were much more formal.

Though some students learned the tools *"as little as it is possible to be able to express themselves"* and meet the minimal objectives Professor Alan had set, he told us others went further. Some students overcame their earlier difficulties and presented above-average projects at the end of the semester. As he expected, more senior students displayed greater aesthetic quality, even in their homework exercises. Alan noted the better outcomes of third-year students, who had elected to attend PRIMER, in comparison to their second-year colleagues, caught under the transitional curricular plan imposed by the faculty's scientific council.

Professor Carl agreed with Alan that most students worked hard, presented good projects, and that many overcame an earlier lack of motivation. Professor Bill also had a positive assessment of the outcomes, noting that he considered some projects to be excellent. Bill also emphasized the good rapport that had been established between the professors and the students, while Alan found the diversity of projects noteworthy. In contrast, Carl believed the massification of creative code tools and online resources had led the average students to present similar types of projects. According to him, few had presented bolder or more experimental work.

✦ OTHER REMARKS Professor Alan told us education is going through changes while simultaneously demanding more profound changes. He explained the integration of new tools and ways of interacting with technology is a major challenge for the future of arts education.

Alan clarified why he had championed Virtual Reality throughout the semester. The professor criticized the sluggishness of mouse-based user interfaces and told us that screens and touch technologies bring about a *"sensory breakdown."* Alan believes those user interfaces *"prevent virtuosity in the use of our tools"* and are therefore inadequate for artistic education and practice. Instead, he would like to see the kind of gestural interface demonstrated in the science-fiction film *Minority Report*. The professor quoted the composer Edgard Varèse:

> *I dream of instruments obedient to my thought and which lend themselves to the demands of my inner rhythm.* —Edgard Varèse (1883–1965)

## 2.8 *Successes and limitations faced in this case study*

The unique nature of this instance of the PRIMER curricular unit caused exceptional stress on the teaching conditions, and thus this case study can be criticized as not being representative of the expected general functioning of the unit. Still, we believe this case study was valuable due to the substantial number of student participants, who were at different stages in their course. We were also able to compare the approaches of three different professors. We thus summarize the successes and the limitations we faced:

***Table 7.2.5.*** *Successes and limitations faced in the PRIMER case study.*

| | *Planned* | *Unplanned* |
|---|---|---|
| *Successes* | Collected most types of data sought. | The exceptional number of students and the opportunity to compare the approaches of three professors.<br>Opportunity to attend students' project presentations. |
| *Limitations & Disappointments* | Significant early delays; missed expositive lectures.<br>Limited access (printouts only) to Moodle and the online content.<br>Very low participation percentage in the second students' survey. | Overcrowded and noisy classroom difficulted observation. |

Presumably due to the issues presented by the transitional curriculum and the hiring of assistants, we only obtained the approval to visit the faculty and to attend sessions a few weeks after the beginning of the semester, and thus we missed most of the expositive lectures and students' first reactions to computer programming. In addition, students' low participation in our acceptance survey at the end of the semester had a negative impact on our ability to conduct longitudinal analysis.

## 3 GRAPHIC: COMPUTER VISUALS AND ANIMATIONS

For this case study, which we shall refer to as GRAPHIC, we went back to RED during the second semester. Like PRIMER, GRAPHIC offered an introduction to computer programming but aimed at students at a Design licentiate. However, there were great differences between the units. Unlike PRIMER, which had been led as a workshop where students were given the opportunity to develop a free exploratory project, GRAPHIC pointed toward the specific domain of computer graphics techniques. In addition, it was an elective unit, newly offered after a curricular revision of the Design course, but also made available to other students at RED.

We will present the licentiate in the first subsection. In §3.2, we will focus on the GRAPHIC curricular unit—its plan, students, and professor. In §3.3, we will present findings from our observations, and in §3.4, we will report on the exercises developed by students. In addition, we conducted an interview with the professor, which we will present in §3.5. We were unable to gather a significant number of responses to our qualitative survey; in §3.6, we will address that, as well as the other limitations we faced while conducting this case study.

### 3.1 *The course and its host institution*

As described in §2.1, RED is a medium-sized faculty that is part of a large university attended by tens of thousands of students. The course which administers the unit under scrutiny is related to the field of Design, and thus we shall henceforth refer to the course as 'Design' though that is not its real designation. The course has been offered as a three-year licentiate since the faculty implemented the Bologna Process, though its curriculum did not feature computer programming until RED promoted a deep reform of the study plan in 2018. That reform introduced a new elective unit introducing computer graphics programming, open to all students at the faculty (as long as their curricular plans assign the necessary credits to electives), and which we shall refer by the codeword GRAPHIC.

✦ STUDENTS AND ACCESS There were over 200 students (a full 80% of whom were female) enrolled in the Design licentiate according to data for the 2017/18 academic year (DGEEC, 2019). In the three years spanning 2016/17 to 2018/19, a little less than 200 new students had enrolled. Those arriving from secondary schooling brought high candidacy grades: the average admission grade in 2018/2019 neared 18 (76th percentile), a grade similar to that of the last student admitted under the general contest.*

According to the same data, 9% of students admitted in 2015/2016 and 2016/2017 dropped out before completing their first year, while over 100 students graduated during the same period with an average final grade nearing 15. We note these outcomes predate the curricular revision.

* We should note the candidacy grade average includes the grades of students who benefited from special statutes or applied through special contests, which may – and often does – lower the average grade of new students below the lowest grade in the general contest.

## 3.2 *The GRAPHIC curricular unit*

The unit which we refer to by the codeword GRAPHIC is available in the second semester to students attending any year of the Design licentiate, as well as to all other students at RED, as long as their courses' curricula include a 6 ECTS credit slot for elective units. Though faculties can adjust the roster of elective units annually without approval from the competent official entities, we note that GRAPHIC, which was newly offered at the time of our study, was also tied to the new AMP curriculum as a requirement for obtaining two of its minor degrees; the AMP study plan for the second and third year includes the necessary elective slots. The weekly GRAPHIC sessions were three hours long and classified as theoretical-practical.

✦ OBJECTIVES AND SYLLABUS According to the curricular unit plan (CUP) available at the faculty's website,* GRAPHIC had the objective of providing students the following set of competencies:

- Historically contextualize graphics programming within the creative community.
- Master the fundamental concepts of imperative programming.
- Learn techniques for generative drawing, procedural animation, and interaction.
- Develop a critical perspective on the role of programming within designers' skills.
- Develop novel computational tools for graphics creation.
- Envisage the application of the acquired skills in new areas.

— *from the GRAPHIC curricular unit plan for the 2018/19 academic year*

In order to achieve these learning objectives, the syllabus included a *"historical contextualization,"* followed by a topic on the *"fundamentals of imperative programming,"* in turn followed by the main computer graphics topics—i.e., "*generative drawing;" "procedural animation"*—and on *"interactive systems."* Introducing *"physical displays, drawing machines, and digital fabrication"* concluded the syllabus.

Though the CUP did not mention a specific tool in neither the objectives nor the syllabus, the use of Processing was heavily implied by the bibliography, and indeed, we observed that to be the case. As discussed in 4.2, Processing was mentioned in 26 other curricular unit plans concerning computer programming within Portuguese public arts and design licentiates. The PRIMER (§2) and SKETCH (§4) cases also concerned units teaching and exploring Processing, which we described in 3.3.2.3.

✦ TEACHING AND EVALUATION METHODS The unit plan did not include any section about the teaching methods, yet it did include some details about evaluation. The CUP emphasized that students were to articulate their personal research on the history of computer graphics with practical programming exercises. Therefore, an essay about a *"historical, technical, or cultural aspect of computer graphics"* was to weigh 30% of the final grade, and individual practical exercises submitted throughout the semester would add up to 50% of the grade. Students were also granted the opportunity to revise their work at the end of the semester for an additional 10%. Finally, the professor's assessment of students' attendance and participation during sessions was to weigh 10%.

* Reference omitted as to comply with the participants' request for anonymity.

✦ BIBLIOGRAPHY The unit plan also included a list with GRAPHIC's bibliographic references, which we reproduce here:


- Parent, Rick (2012). *Computer Animation: algorithms and techniques*. Morgan Kaufman.
- Reas, Casey & Fry, Ben (2007). *Processing: a programming handbook*. The MIT Press.
- Reas, Casey & McWilliams, Chandler (2010). *Form+Code in Design, Art, and Architecture*. Princeton Architectural Press.
- Shiffman, Daniel (2012). *The Nature of Code*. Self-published.
- Taylor, Grant (2014). *When the Machine Made Art*. Bloomsbury Academic.


In comparison with the unit plans we analyzed in 4.2, GRAPHIC had, but for the oft-recommended Processing manuals, a distinct bibliography. Grant Taylor's examination of the cultural debates surrounding computer art was not mentioned in any other CUP, and neither was Parent's technical animation textbook. And though Shiffman's book about programming physics and natural system simulations with Processing can be read online for free, it was mentioned in only two other CUPs (one being SKETCH's, see §4), possibly due to the greater complexity of its subject matter. We believe it is worth noting neither recommendation was available in Portuguese.

✦ THE PROFESSOR The curricular unit was the sole responsibility of the docent to whom we shall refer by the pseudonym 'Douglas' to comply with the anonymity requests. The docent told us he had been teaching at RED for three years and admitted he had no prior teaching experience. He told us that at the time of our inquiry he was leading another curricular unit at the licentiate, was not related to programming, alongside two units at an independent master's Design program.

Douglas told us he had a formal background in computer science. He also talked at length about how he became interested in the 'demo scene' as an undergraduate and thus decided to obtain a master's degree in computer graphics. He went on to start a small studio afterward, working on data visualization as well as game creation projects. He explained that he was proficient in coding in C and C++, especially when in articulation with OpenFrameworks. He also used the Pure Data visual programming tool for other projects.

Douglas added he had become more interested in research in recent years and was writing his Ph.D. thesis at the time of our study; he told us he found it hard to conciliate his research with his freelance work, his teaching, and his family life.

✦ THE STUDENTS Douglas told us there were fifteen students enrolled in GRAPHIC and that he estimated a majority were learning to code for the first time. From what we could gather during his roll calls, there were five first-year students, eight second-year students, and two third-year students. Three students were studying AMP and had attended PRIMER during the first semester. Another student was studying Painting, while the remaining students were studying Design. Out of the fifteen, only two students were male; thus, the unit had an even greater gender imbalance than either course.

✦ LEARNING MANAGEMENT The professor did not employ a learning management system (e.g., Moodle) to support his lessons. He had set up a Google Drive folder so that students could upload their exercises, which he also employed to share the code he

wrote during lessons and the instructions for the exercises. The professor shared this folder with the researcher so we could analyze students' work; we will present our findings in §3.4.

## 3.3 *Session observation*

Though the faculty and the responsible professor granted us authorizations for our study before the beginning of the semester, we then faced some difficulties in articulating our study with the professor. The professor was incommunicado for long periods, which difficulted the scheduling of our visits; these difficulties were compounded by an unfavorable second-semester calendar. Hence, we were only able to observe two GRAPHIC sessions, corresponding to the eighth and tenth lessons.

### 3.3.1 CLASSROOM CONDITIONS

Sessions took place in a long and narrow classroom, which we estimated as 10 meters long and between 3 and 4 meters wide. We made renderings based on our approximate measurements and sketches, which we will present in Figures 7.3.1 and 7.3.2.

There were twelve small desks inside the classroom, each just wide enough to be shared by two students. At the beginning of both sessions, the professor and the students rearranged the desks into their preferred layout. Some were placed in rows facing the stage at the end of the room, while the remainder were put together by students into a bigger table or just left in the nooks by windows. The small stage had a small desk holding sound equipment and a computer, though the professor preferred to use his own laptop. A video projector hung from the ceiling, and a relatively large screen hung from the stage wall, as seen in Figure 7.3.1.

***Figure 7.3.1.*** *A general view of the classroom environment.*

We felt text was readable from the opposite end of the room, though the professor often adjusted the font size in the Processing development environment. There were no other computers in the room, implying students should bring their own laptops, as shown in Figure 7.3.2. The classroom also lacked a whiteboard, a fact the professor complained about when we interviewed him (see §3.5). The room also had a high flat ceiling and a ceramic floor, and therefore its acoustics were quite poor.

***Figure 7.3.2.*** *A view of the classroom from the professor's stage.*

As also shown in Figure 7.3.2, a few students would sit down at the 'big table' by the doorway across the room from the stage. We had the impression these students were on their own 'island' away from the rest of the class and were likely to leave the session early —indeed, some students faced scheduling conflicts with other curricular units.

### 3.3.2 THE LECTURES

We attended two sessions of the GRAPHIC curricular unit on the eighth and the tenth week of classes. On our first visit, the professor told us he had fallen two weeks behind in his lesson plan. He dedicated the first half of that session to supervising students' exercises, as we will report on in §3.3.3. The second part of that session, as well as the entirety of the week 10 session, were dedicated to lectures, which we will thus narrate:

- VECTOR INTERPOLATION AND ANIMATION, WEEK 8 Only five students attended Professor Douglas' lecture during the second half of the session, after a break. Several students who had been in the classroom during the first half had asked the professor about the topic of the lecture before deciding whether to stay, as they faced scheduling conflicts with another curricular unit.

  The professor explained the use of vectors to store coordinates and how to calculate interpolated values between two points as to animate shapes. Only two students paid their full attention to the lecture and reproduced the code Douglas wrote, seen in the projection screen. The others were either distracted or working

for other units. After completing his first example, the professor iterated on his code to demonstrate further uses of the interpolation technique (e.g., controlling the rate of animation with mouse movement). Still, Douglas' example was visually simple, and his subsequent iterations did not show salient visual improvements, which may have contributed to students' inattentiveness.

- PARTICLE SYSTEMS, WEEK 10 Eleven students attended the lecture. Before diving into the main theme of the session, the professor took ten minutes to summarize the previous week's, presenting code that simulated a ball bouncing in 3D space, which he had shared with students through Google folder; five students had opened the code and ran the sketch on their own computers.

  The professor then introduced the theme of particle systems by playing a video excerpt from the 1980s movie *Star Trek: The Wrath of Khan*, explaining the technique had been developed for its visual effects sequences. Douglas also presented the original research paper* about the technique. He then demonstrated how to code a 'space travel' animation in Processing. We saw seven students followed and reproduced the code on their own laptops. After completing the demonstration, the professor summarized the structure of his entire program, not asking whether students had been able to reproduce it. From the researcher's vantage point further back in the classroom, it could be seen two students had not been able to run their code and only managed to do it much later in the session.

  Douglas allowed students to take a short break about halfway through the session. Upon their return, he showed artwork in which Casey Reas used particle systems to create drawings; Douglas also explained how the technique is useful for simulating fire or water. In the remaining time, the professor continued to modify his demonstration program with additional refinements: he first assigned the 'star field' particles different velocities, colors, and sizes; then changed the code for their movement, so particles moved in one direction, simulating a 'jet thruster' effect; then refining that effect, so particles changed color and size as to appear to fade away. Five students had replicated the professor's code in their computers, yet others had not managed to do so by the end of the lecture.

- PROFESSOR'S BEHAVIOR Even though we only witnessed two lectures, Douglas presented consistent behaviors and stances which we believe are worth noting:

  FIRST, the professor sat down at his desk throughout his lectures, using his laptop, mirrored to the classroom projector, to demonstrate programming. Douglas often adjusted the Processing environment font size, for he struggled to maintain a balance between text readability and display enough code so students could keep up (even though the projection was large). The professor did not employ slide presentations and instead used his web browser to present content related to his lectures. Douglas spoke calmly and quietly; given the room's poor acoustics, sometimes it was difficult to understand his speech, even if students were silent and the door was closed.

* Reeves, William (1983). Particle Systems: A Technique for Modeling a Class of Fuzzy Objects, *Computer Graphics* 17 (3).

SECOND, we felt the professor's manner of coding presented hurdles to students trying to follow his demonstrations. Douglas typed fast but often made small mistakes which he would not correct immediately, preferring to edit a whole block of code for typos a few moments later. In addition, the professor did not comment his code—not during lectures and neither the code he shared with students. This was compounded by the professor's use of abbreviations for variable names. Though we understand good code is self-explanatory, that assumption relies on the programming proficiency of the reader. When a student questioned his abbreviated variable naming, Douglas tersely replied: *"many years of coding."*

THIRD, at times the professor showed some slight unfamiliarity with Processing, which he had admitted not to use regularly in his practice. We witnessed Douglas relying on the development environment's 'autocomplete' feature to look for the appropriate keywords or else looking for information in the reference website.

FOURTH, the professor rarely asked students any questions. In addition, he did not check whether students were keeping up with his demonstrations. Instead, Douglas preferred to summarize and reiterate how his code works when he successfully ran his program. When asked about this in informal conversation, Douglas told the researcher he felt his students were shy and unlikely to participate in class, adding that he stood by his approach.

FIFTH, we felt each lecture was approached as a self-contained tutorial that did not build from the previous weeks' lessons: e.g., not employing vectors in his lecture about particle systems, throughout which he explained details (e.g., initializing an array) which we felt should have been well understood at that point in the syllabus. By Douglas' own admission, he had not been able to follow his lesson plan, as he had postponed sessions due to his conflicting responsibilities. Still, we felt there was a great degree of improvisation in Douglas' lectures, with the professor relying on his deep knowledge of computer graphics algorithms.

✦ STUDENTS' BEHAVIOR Though we only witnessed two lectures, we witnessed a set of behaviors from students we find worth summarizing. We note some students' scheduling conflicts cast a shadow over the functioning of the curricular unit.

FIRST, students had no alternative but to work on their own laptops, as we noted. Some were tempted by distractions such as Facebook or messages whenever there was a lull during the lectures. Yet, we also witnessed students consistently working for other units and fully ignoring the lecture: of the five that stayed for the vector interpolation lesson, two were writing essays in Word. Another student was working with typesetting software during the lesson about particle systems.

SECOND, students attentive to the lectures rarely interrupted the professor and asked for clarification, even though they often struggled to replicate the professor's code in their own computers. As mentioned, we felt Douglas' brisk and irregular lecturing pace was challenging for students, who also had to contend with difficulties presented by the projection screen's limitations.

### 3.3.3 TUTORING

Eleven students were present during the first part of the week 8 session, which was dedicated to Douglas tutoring students working on their first individual exercise, a small program that generated a random pattern design, and which was due for submission the following week. During around 90 minutes, the professor paced around and oversaw the work of seven students, six of whom were female. Four of the discussions we witnessed between the professor and his students were quite brief: one student obtained a positive answer to whether he could use Processing's 3D graphics mode; two other students showed their exercises as they stood and were met with the professor's approval, though he advised one of the students to proceed with further aesthetic explorations; another student asked for an extension of the submission deadline, claiming problems with her computer. Douglas addressed the whole class and said he would grant another week.

Professor Douglas spent over twenty minutes sitting beside students in three separate instances, in which he used their laptops and wrote part of their code himself. In one such situation, the professor did so after verbally explaining to the student how to fix her positioning code; in another, the professor demonstrated how to use a loop to avoid using repeating statements. The third student expressed her desire to replicate the visual effect from a poster but did not know how to begin. Douglas spent a few minutes experimenting until he obtained a visually similar result, after which he challenged the student to improve the code as to add randomness in the shapes it produced. As during his lectures, Douglas did not write comments and employed a terse naming convention in his code, though he explained his code step by step once he had finished.

While the professor assisted individual students, we also looked at the behavior of their colleagues. As mentioned, some sat around a table they had put together close to the door and away from the rest of the class, so they could leave discreetly once they had talked to the professor. Though a couple of the students at that table had Processing on their laptops, they were mostly chatting with a colleague who was cutting prints with a pair of scissors, clearly working for another curricular unit. Still, we overheard these students discussing GRAPHIC: one showed her exercise to her colleagues and agreed she was beginning to enjoy the unit. Students sitting elsewhere in the classroom also felt comfortable to take the time to work for other units, though most also had Processing open with their exercise ready for demonstration, not just to the professor but also to their colleagues, with whom they were likely to discuss their work. We also saw two students wearing headphones and watching Processing video tutorials on YouTube.

⁂

We felt most students attending GRAPHIC had a genuine interest in learning computer programming. Despite the professor's inexperience and lack of lesson planning, we felt Douglas had the goodwill of the students, who were interested in his command of the subject matter. Nonetheless, most students were also quite tempted to dedicate their time to working for other curricular units. Though the GRAPHIC's status as an elective may have led some students to deprioritize it, we also felt the evaluation requirements were possibly too light and may have discouraged students from further dedication.

## 3.4 *Students' work*

Students were required to develop two individual programming exercises throughout the semester. As the professor did not hold formal presentation sessions, he asked his students to submit their source code to a Google Drive folder he managed, and which he subsequently shared with us at the end of the semester.

As mentioned, we witnessed students nearing completion of their first exercise during our first visit to the GRAPHIC classroom. The exercise, which consisted of writing a small program that generated a random pattern, had a few additional requirements, notably that students had to employ a grid system (though not necessarily orthogonal or explicit) to hold shapes that had to be more complex than straight lines. Douglas' instructions mentioned explicit evaluation items: 10% were assigned to code cleanliness, 80% to the implementation of the desired visual composition, and 10% to the aesthetic result. All students submitted the exercise; the researcher ran their code on his own computer to present a selection of the resulting patterns in Figure 7.3.3 (next page).

Students were asked to submit their second individual exercise by the end of the semester: an interactive generative animation, which was to be based on a historical computer artwork of their choice. The instructions also mentioned explicit evaluation criteria and again assigned 10% to code cleanliness and 10% to the aesthetic result, yet 40% were assigned to the implementation of the desired animation and 40% to the user interaction. Again, all fifteen students submitted the exercise; in Figure 7.3.4 (overleaf), we present a selection of still images captured after running their code.

With a couple of exceptions, we found there was little originality in most exercises, which tended to merely add a few changes to code the professor shared after his lectures. Though all met the second exercise's requirements, five students did not implement interactivity in any consequential way, as their programs merely paused the animation on a mouse button or keypress, easily implemented in Processing by appending two stand-alone lines of code.* Other students implemented more sophisticated controls; two students even implemented complete user interfaces that used both the keyboard and the mouse for controlling animation parameters.

Most students acquired the professor's habit of not commenting their code: twelve of the first exercise's submissions lacked any comments—their source code files even lacked identification at the beginning. The second exercise's submissions followed a similar pattern. Among the few students that commented their code, one stood out as proficient at programming (i.e., using object-oriented programming techniques not in GRAPHIC's syllabus in his first exercise), and therefore likely to have developed coding habits of their own.

We note students were also required to submit a written essay about computer graphics history or culture at the end of the semester, which accounted for 30% of their final grade; however, we were not given any access to students' theoretical work.

---

* `void mousePressed()  { noLoop(); }`
`void mouseReleased() { loop(); }`

– note all of these are Processing built-in functions.

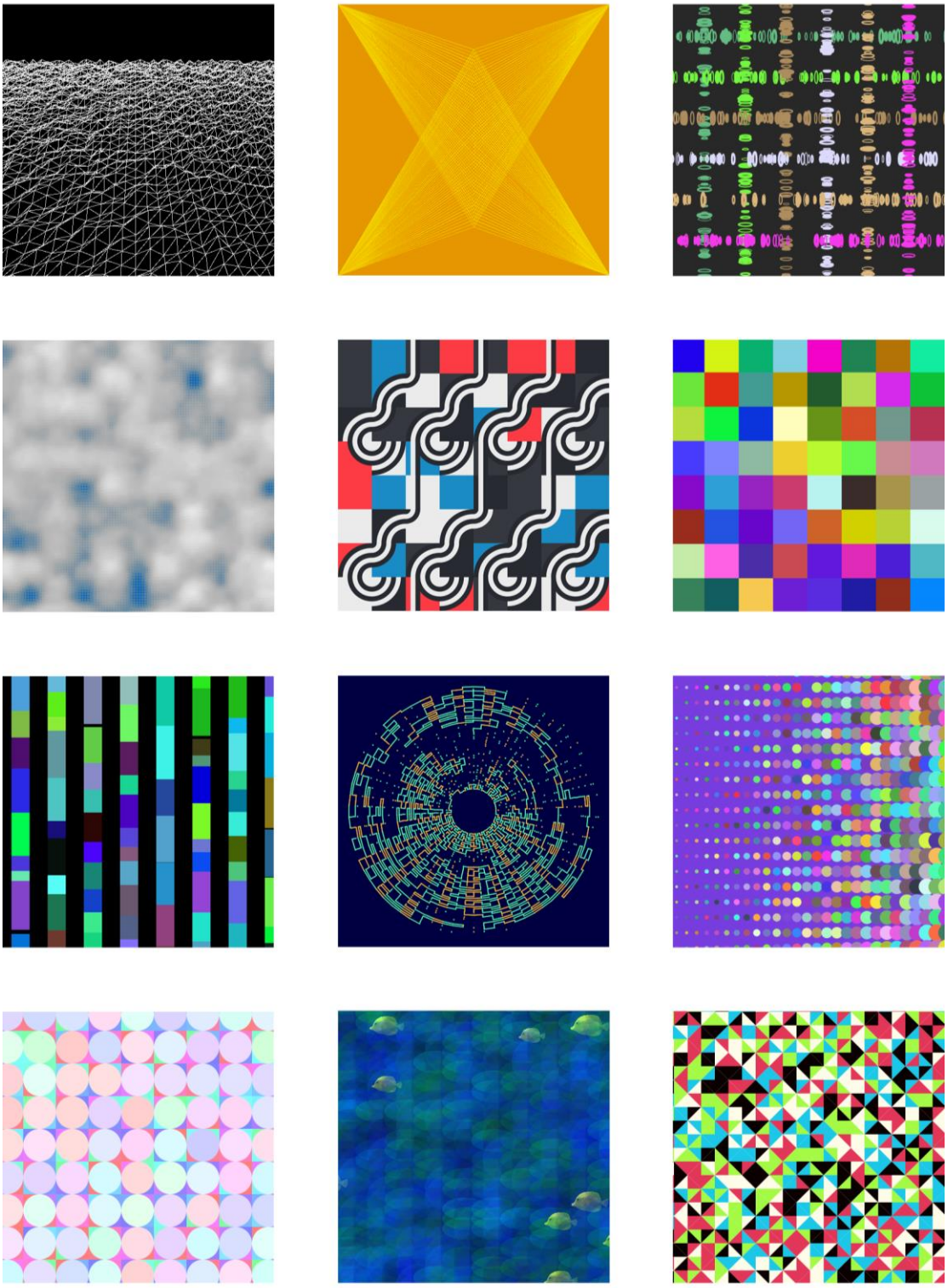


***Figure 7.3.3.*** *A selection of captures of the students' first exercises output, as run in the researcher's own Processing installation (some cropped to a square aspect).*

*Note the variety of approaches to the challenge of filling a grid layout with random elements.*

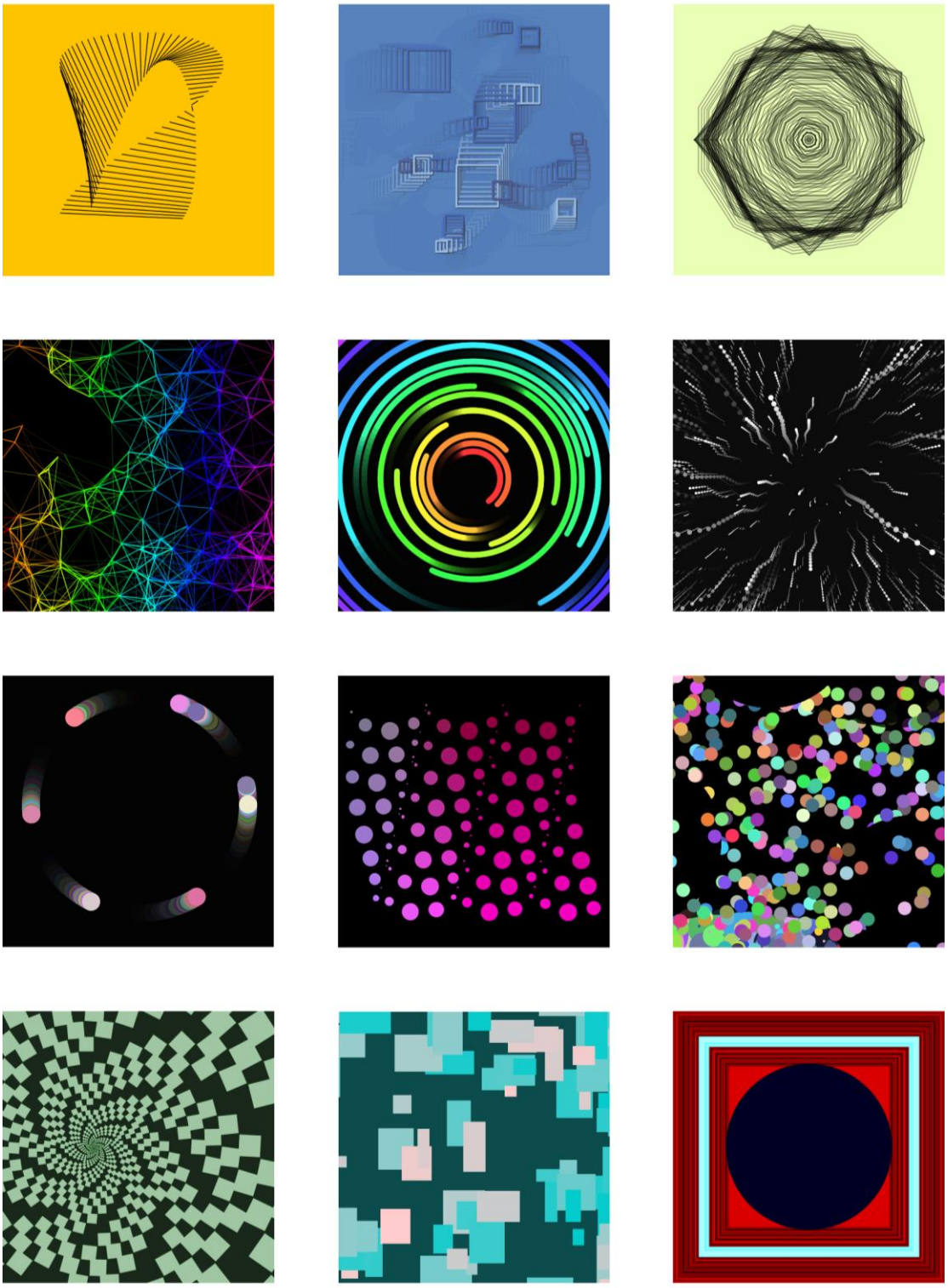


***Figure 7.3.4.*** *Stills captured from a selection of students' second exercises animated output, run in the researcher's own Processing installation (cropped to a square aspect).*

*We found considerable similarities between some submissions and the professor's code from his lectures—e.g., the work represented by the rightmost image in the second row, as well as another not presented here, closely match the 'star field' code demonstrated in the lecture about particle systems.*

## 3.5 *Interview with the professor*

Professor Douglas granted the researcher an interview via videoconference, which followed the procedure laid out in 5.5.1.4. The interview took place soon after the last session and lasted over ninety minutes.

✦ THE VALUE OF LEARNING TO CODE Professor Douglas' views regarding the relationship between computing and the arts were shaped by his experiences during his undergraduate studies in Computer Science, when he was involved in the 'demo scene,' which he defined as *"using computers for audiovisual production."* The professor told us (edited for clarity):

> All areas of knowledge are increasingly dependent on computational support. This also depends on people being able to bend these platforms according to their needs rather than on using general-purpose software. Therefore, all professionals related to knowledge production, information work, information analysis, need a computational foundation, especially computational thinking. It is not necessarily a matter of knowing how to program. More often, it is about being able to employ computational thinking to articulate processes.

Douglas believed the value of knowing computer programming depends on the specific practice: *"it makes sense for artists working outside of interactive or computer art to learn how to code [...] only if that will help them develop the conceptual thinking regarding their work."* Still, computational thinking skills can be useful for everyone, and Douglas gave the example of a Painting student that completed GRAPHIC:

> Though she might not use Processing ever again, at least she already has a set of mechanisms that will allow her to conceptualize her work [as well as] to formulate a discourse about those mechanisms.

The professor noted, *"design students have many more working processes that can benefit from computational aid"* and that understanding the foundations may help designers communicate with other professionals in the workplace.

✦ THE WORKING CONDITIONS The professor believed a seven-day gap between sessions was too great and that students' *"enthusiasm drops throughout the week and by the time of the following session they do not remember where they were."* He conceded it would be perhaps unrealistic to ask for additional hours, suggesting the alternate possibility of splitting the unit's schedule into shorter biweekly sessions.

Professor Douglas told us the classroom was too narrow and that he would have preferred if students were able to sit around his desk rather than giving a lecture from a stage. He also complained about the lack of a whiteboard, stating it was difficult for him to teach programming without being able to draw diagrams easily. The professor also thought projection was inadequate, as *"many [students] spent more time trying to reproduce the example than understanding"* his code. Douglas was looking for ways to write code on a real-time platform so that students could follow in their own computer screens. Ideally, he would set aside the projector for presenting only the visual output of his demonstration programs.

✦ INSIDE THE CLASSROOM Though there was a group of *"three or four students"* that had incompatible schedules and often left mid-session to attend another class, Professor Douglas explained there were several students who attended all sessions and were *"interested and participative in the sense of paying attention and following what was being done*." In that sense, he considered students participated in the class.

Yet, the professor felt his students were shy and unwilling to participate actively, *"neither asking questions nor exploring ideas."* Considering the elective nature of the unit, he *"had expected students with [an active] stance, which is the typical stance [he] found in the students in workshops."* Douglas was disappointed by students' passivity, which he related to the wider passive attitude to learning he found while teaching other curricular units; moreover, he claimed other professors at the faculty shared the same complaints. The professor told us that *"you either go straight into project-based teaching, as in those Design units in which students are immediately put to work on group projects, or else their attitude is to sit and wait for the knowledge."*

Asked about how he approaches his students' initial contact with programming, Douglas told us he starts with something simple that produces immediate results:

> First, I ask them to make drawings because it is immediate—you can easily put something on the screen, to make it move, animate, etc. As they do that, I keep teaching how the language works, about function calls, arguments, all that. Once they know that, then I introduce some complexity [...] What I try to do is to build a less inclined ramp for them.

The professor doubted most of his students were able to learn autonomously. Douglas lamented that *"though students have an almost infinite amount of information available, they often stop at ready-made things,"* copying and pasting code from online sources. He explained there is a limit to what can be done that way and that such solutions—which he jokingly called 'Stack Overflow-oriented programming'[*]—were often misapplied. The professor also explained the *"nomenclature problem"* presents a limitation for students at the earliest stages, as they often do not know what to search.

Douglas believed that asking students to complete short exercises during sessions is *"a waste of time*," as students become inattentive. Though he admitted students often struggle at first, he prefers they follow his coding demonstrations on his lessons' topics. The professor explained some students had difficulties with their first exercises, yet they understood the foundations of programming later in the semester.

✦ THE STUDENTS Professor Douglas estimated most students (*"sixty, seventy percent"*) had their first contact with computer programming at GRAPHIC. Though students at the faculty can enroll in the unit at any year throughout their courses, he felt many were delaying the option until deciding on a minor degree. Yet, students often asked Douglas questions such as *"how can I use [programming] in my projects?"* or *"how can I articulate this with my ongoing work?"* Hence, the professor believed it would be beneficial for students interested in programming to complete GRAPHIC early in their studies so that it could be *"useful for their work plan and their work process."*

[*] A popular forum about coding questions and answers – stackoverflow.com

The professor felt differences in the starting attitudes of students depending on their seniority, as he believed third-year students had enrolled in the unit more *"out of curiosity."* Remarking on a disparity in attitude between his licentiate and master's students, Douglas told us *"younger students are much more open to the possibilities,"* whereas *"older students, especially if they are practicing designers with sedimented working processes, resist the introduction of new processes into their work."* He believed younger students were more likely to regard computer programming as useful, despite any anxiety about learning.

Douglas met students that presented great anxiety toward programming when he was leading mandatory units, yet that was not the case in GRAPHIC. Though he *"never came across a student that was completely frustrated at the end [of the semester],"* the professor admitted some students carried such a heightened perception of the difficulty of learning to code that they gave up early. Douglas also told us some students prefer to perform punishingly repetitive tasks rather than learn how to code. As to illustrate, he told us of a student at the Design master's course who wrote hundreds of web pages by hand, copying contents from a database, instead of learning how to code dynamic webpage generation. The professor found it *"very worrying when [anxiety] leads students down a path that will bring them many difficulties in the future."*

✦ EVALUATION AND OUTCOMES The professor thought the unit went *"surprisingly well,"* as he had expected students to show greater difficulties with the syllabus. All students understood *"the basics of how Processing works [and] how they can use multiple language structures to plan a computer program."* Yet, Douglas lamented some students *"still cannot transcribe their ideas to the screen."* These students, he added, *"understood the structure of programs and how their code related to the visual output but were yet to take the leap into becoming self-sufficient."* In contrast, other students *"got to a level at which they can express themselves visually and think about interactivity."*

Still, Douglas told us he was considering changes to his evaluation methodology to address students' loss of contact with GRAPHIC in the days following each session. The professor entertained the possibility that practical evaluation could consist of a portfolio of short weekly exercises. However, he was wary of the likelihood of students not submitting their weekly coursework and of continuous evaluation possibly limiting students' expressive potential.

✦ OTHER REMARKS Though Douglas found it hard to conciliate his freelance work, his research, and his teaching, he felt these activities were mutually beneficial:

> My teaching practice helps me communicate with clients and with people within the context of research projects. My freelance practice helps me 'keeping it real' when working with students and on research projects. And research helps me advance rather than keep doing the same things, and to bring what I learn to the educational practice and to the commercial practice.

### 3.6 *Successes and limitations faced in this case study*

As GRAPHIC had been offered to students for the first time at the time of our study, it was already apparent that there would be some future changes in the profile of students attending the unit, as the more would be enrolled in the new course's curricula, the more would be interested in obtaining a minor degree. As he manifested his intention during our interview, it is likely the professor will adjust the syllabus and his teaching and evaluation methods. Therefore, this case study may be criticized for its time-specificity. Still, we believe the case study was valuable as it presents, in comparison to the PRIMER study in §2, a contrasting approach to introductory computer programming taking place within the same faculty. We thus summarize the successes and the limitations we faced:

***Table 7.3.1.*** *Successes and limitations faced in the GRAPHIC case study.*

| | *Planned* | *Unplanned* |
|---|---|---|
| *Successes* | Able to observe two sessions.<br>Granted the interview with the professor. | Opportunity to analyze students' code submissions after the professor shared their exercises. |
| *Limitations & Disappointments* | Significant delays in scheduling observations, complicated by the academic calendar.<br>Failure to obtain a meaningful participation in students' survey. | Though we were given access to students' code, we were not given access to their written essays. |

Though we desired to attend sessions from the start of the semester and had obtained the necessary authorizations from the institution well in advance, there were great delays in agreeing on a schedule with the professor; we were about to cancel the case study by the time we were invited to proceed. This difficulty was compounded by a complex second-semester calendar in which there were sessions suppressed due to national holidays and in which we faced scheduling conflicts with other cases' observations. In addition, only six students completed the first application of our acceptance survey; hence we opted disregard the survey.

# 4 SKETCH: BASICS OF INTERACTIVE DESIGN

The case which we shall refer to by the codeword SKETCH concerns a curricular unit offering a practical introduction to computer programming to second-year students attending a Design licentiate offered at a faculty which we shall refer to as LIME. We find it notable that, in the absence of curricular changes, people responsible for the course and the unit had gradually adapted the latter's plan, moving its focus away from teaching audiovisual software applications and toward teaching students how to code.

We will present LIME and the relevant Design course in the first subsection. In §4.2, we will turn our focus to SKETCH and describe its unit plan and syllabus, its student make-up, its session schedule, and its professor. In subsequent subsections, we will present our findings: in §4.3, we will describe our observations inside the classroom, made during six different visits to the faculty; in §4.4, we will describe SKETCH's online resources, as the responsible professor made broad use of other online platforms besides the unit's learning management system; in §4.5, we will report our observation of final project presentation and debate sessions. In §4.6, we will present findings from students' quantitative technology acceptance and use surveys, applied at the start and at the end of the semester. In §4.7, we will report on our interview with the professor leading the unit. We will conclude the section by addressing the successes and limitations we faced while conducting this case study.

## 4.1 *The course and its host institution*

LIME is a relatively small faculty integrated within a large university. Tens of thousands study at the university, yet LIME is attended by less than one thousand students, over 600 of which attend licentiate courses. Like RED, LIME's facilities are also located at a location away from the parent university's main campi.

✦ THE COURSE The course offering the curricular unit under study grants a licentiate classified as in Design under the CNAEF (see **4**.1.3). Hence, we shall henceforth refer to the course as 'Design,' though that is not its real designation. In the mid-aughts, LIME promoted a curricular reform to meet the Bologna Process requirements, which made an elective unit about introductory programming available to third-year students; yet that offering was removed two years later after adjustments to the curricular plan, which still holds to this day. Therefore, the present curriculum does not include any unit explicitly concerning computer programming, apart from an elective unit about advanced web design topics, which was in the meantime made available to third-year students. Thus, the initial contact with computer programming offered to students attending the licentiate does not emanate from the explicit curriculum but rather from the initiative of successive course coordinators and professors who have proposed to teach the subject within the existing curricular confines.

The curricular reform that implemented the Bologna Process introduced a mandatory second-year unit we might refer to as a 'multimedia laboratory.' LIME's

website includes a publicly accessible curricular unit plan (CUP) archive,[*] which enabled us to track the changes to the curricular unit's the years. Despite overlapping other units' syllabi, the CUPs of that 'multimedia laboratory' unit dating from the first few years of the new curriculum's implementation portray a concern with teaching video and audio editing techniques. However, despite maintaining the same designation and place in the curriculum, the syllabus and the objectives of the unit were deeply changed from 2011 onwards. The plan for that year's instance established the unit, which we henceforth will refer to by the codeword SKETCH, as a space for experimentation in generating visuals and sounds using electronics; subsequent instances' CUPs explicitly mention teaching to code. Electronics were phased out, and unit plans from 2015 to the time of our study portray SKETCH as a straightforward introduction to programming.

✦ STUDENTS According to official government data, there were over 200 students (70% of whom were female) attending the licentiate during the 2017/2018 academic year (DGEEC, 2019). Little over 150 new students were admitted during the three-year period from 2016/2017 to 2018/2019. Those arriving after completing secondary schooling had considerably high grades: the average candidacy grade for students admitted to the course in 2018/19 was almost 18 (86th percentile), while the last student admitted under the general admission contest had a candidacy grade over 17.

According to the same data set, 12% of Design students dropped out of the course before completing their first year during the 2015/16 and 2016/17 academic years. In the same period, a little under 100 students graduated with an average final grade of 15.

## 4.2 *The SKETCH curricular unit*

The curricular unit we refer to as SKETCH was a mandatory part of the second semester of the second year of the Design licentiate course plan. As mentioned, the curricular unit plans were deeply revised since the unit's initial offering related to audiovisual editing. At the time of our studies, these plans portrayed a unit offering a theoretical-practical introduction to computer programming. In the 2018/2019 academic year, sessions were weekly and three hours long.

✦ OBJECTIVES AND SYLLABUS According to the curricular unit plan (CUP), the unit's aimed at teaching *"basic computer programming foundations toward audiovisual creation"* and *"a historical understanding of interactive art and computers as a creative tool."* On completion of SKETCH, students were expected to be competent to *"create audiovisual works through the exploration of Processing code,"* as well as to *"employ algorithmic strategies to generate different visual outputs from the same source code,"* and to target these outputs at different media, such as *"print, video and animation, and interactive applications."* The syllabus was articulated to the unit's objectives and was thus summarized:

> [The unit] covers the essential concepts of programming in the first two individual modules (e.g., functions, variables, control structures, randomness,

[*] Reference omitted as to comply with the participants' request for anonymity.

> objects, libraries, interactions with sound, and communications with other software and interfaces). [...] The third module consists of a final global group exercise, where all the concepts will be put into action. Students will be encouraged to acquire the necessary team organization skills and additional programming techniques.
>
> — *from the SKETCH curricular unit plan for the 2018/19 academic year*

Students were required to use Processing (see 3.3.2.3) in their projects. As we described in 4.2, Processing was mentioned in over a fifth of the computer programming unit plans within the universe of Portuguese public arts and design undergraduate education. Processing use was also central in the units to which the PRIMER (§2) and GRAPHIC (§3) case studies report.

✦ TEACHING AND EVALUATION METHODS Though there was no explicit mention of teaching methods in the CUP, the evaluation methods section emphasized students' autonomous work. The periodic evaluation comprised a full 90% of the final grade, with the remaining resulting from class participation and small homework exercises. Students were required to submit two individual works during the semester, each weighing 30% of the periodic evaluation, as well as a group project at the end of the semester, which weighed 40%. As we will see in §4.3 and §4.5, students were required to meet precise instructions and were not free to choose the theme of their works. We note the CUP included an explicit admonishment that works submitted without prior discussion and supervision by the professor would be disregarded.

✦ BIBLIOGRAPHY The unit plan also included a section dedicated to SKETCH's bibliography, emphasizing two books by Processing's creators as required readings:

- Reas, Casey & Fry, Ben (2010). *Getting Started with Processing*. O'Reilly.
- Reas, Casey & Fry, Ben (2014). *Processing: a programming handbook, 2nd edition*. MIT.

Reas and Fry's books were frequently recommended by the units teaching Processing that we identified in our panorama of programming education for undergraduate arts and design (see 4.2). SKETCH's CUP also included a comprehensive complementary bibliography, which we believe is worth presenting:

- Bohnacker, Gross, et al. (2012). *Generative Design: visualize, program, & create with Processing*. Princeton Architectural Press.
- Greiman, April (1990). *Hybrid Imagery: the fusion of technology and graphic design*. Watson-Guptill.
- Klanten, Ehmann & Feireiss (2011). *A Touch of Code: interactive installations and experiences*. Die Gestalten Verlag.
- Lieser, Wolf (2009). *Arte Digital*. H. F. Ullmann.
- Maeda, John (1999). *Design by Numbers*. MIT Press.
- Maeda, John (2000). *Maeda@Media*. Thames & Hudson.
- Maeda, John (2004). *Creative Code: aesthetics and computation*. Thames & Hudson.
- Pearson, Matt (2011). *Generative Art*. Manning.
- Reas, Casey & McWilliams, Chandler (2010). *Form+Code in Design, Art, and Architecture*. Princeton Architectural Press.
- Shiffman, Daniel (2008). *Learning Processing*. Morgan Kaufman.

- Shiffman, Daniel (2012). *The Nature of Code*. Self-published.
- Wands, Bruce (2007). *Art of the Digital Age*. Thames & Hudson.
- Whitney, John (1980). *Digital Harmony: on the complementarity of music and visual art*. Byte/McGraw-Hill.


A single book amongst the required and suggested readings is available in Portuguese. Multiple suggestions in this complementary bibliography were alternative Processing handbooks, such as Bohnacker & colleague's, Pearson's, and Shiffman's. The latter's *Learning Processing* was a popular alternative to Reas and Fry's handbook, mentioned in ten other bibliographies, while the other two recommendations were unique and found in only one other CUP we analyzed. It is also worth noting the inclusion of *Generative Art* and Shiffman's *The Nature of Code* (the latter found in two additional bibliographies), which delve into advanced Processing techniques. The remaining suggestions tended toward distinctness and comprised panoramas of digital arts and design (Lieser's, Klanten & colleagues', Maeda's *Creative Code*, Reas & McWilliams', Wands') as well as a progressive selection of monographs by historical practitioners of computer art and digital design (John Whitney, April Greiman, John Maeda).

✦ THE PROFESSOR The unit was the sole responsibility of the professor we shall refer to as 'Edwin,' as to ensure his anonymity. He told us he is a Design graduate who had had no interest in computer programming right until his final licentiate project, when a professor made him appreciate the possibilities brought by code. Edwin described how he learned to code mostly by himself and by attending some workshops. He added he felt comfortable using Processing, JavaScript, and PHP, and less so using Python and C++, noting he only had a basic grasp of the latter. He admitted sometimes he feels limited by a lack of formal computer science training, adding that he codes with a *"reference website open at the same time because he has trouble remembering things."*

Professor Edwin explained he went to work at another university shortly after graduating, first as part of its non-academic staff, later being invited to teach, which he had been doing for thirteen years at the time of our interview. Edwin had returned to lecture exclusively at LIME two years before, shortly after completing his doctorate. At the time of our inquiry, he was teaching three other units at the Design course besides SKETCH, one of which also involved teaching students to code, though in JavaScript.

✦ STUDENTS, CLASSES, AND SESSIONS Professor Edwin gave us precise figures concerning the number of students enrolled in SKETCH. There were 61 local Design students enrolled, 40 of which were female and 21 were male. The students were almost evenly divided into two classes, which we shall refer to as classes I and II. As mentioned, each weekly face-to-face session was three hours long. Both classes had their sessions scheduled for the same weekday, one during the morning and another during the afternoon. In addition, there were also six Erasmus mobility students attending the unit—four Brazilian and two Israeli. As the latter two did not speak Portuguese, they were both assigned to class I, so Professor Edwin taught his morning lectures partially in English. The professor also had an office hours schedule slot exclusively for SKETCH students, as some repeating students had severe scheduling problems.

## 4.3 *Session observation*

We faced no difficulties in obtaining the necessary permissions to conduct the SKETCH case study and to arrange our observation of some of the sessions. We were only limited by some scheduling conflicts with the other studies taking place simultaneously during the second semester of the 2018/2019 academic year.* In total, we visited LIME six times to attend regular sessions of the curricular unit, as summarized in Table 7.4.1:

***Table 7.4.1.*** *Summary of the observed sessions of the SKETCH curricular unit.*

| *Class* | *Week* | *Students present* | *Session summary* |
|---|---|---|---|
| I | 2 | 30 (roll call) | The value of learning to code. Introduction to Processing. Drawing exercises. |
| II | 2 | 28 (roll call) | |
| I | 7 | 17 (roll call) | Introduction to 'active mode' animation. Custom functions. Coordinate transformations. |
| II | 9 | 23 (roll call) | Using external audio libraries. Introduction to object-oriented programming. |
| I | 12 | 18 (estimated) | *Laboratory practice.* Supervised 15 projects (involving 30 different students). |
| II | 12 | 20 (estimated) | |

We note Professor Edwin organized a Processing Community Day† at LIME in place of the first week's sessions, which the researcher attended. The event hosted masterclasses and talks by local computer art practitioners and researchers. These were attended, according to our observation, by a substantial number of students. Though omitted from the table, we also visited the faculty during the 14th (and last) week of the semester to witness students' formal presentations of their final projects. Our observation of these final presentations will be reported in detail in subsection §4.5.

### 4.3.1 CLASSROOM CONDITIONS

Both morning and afternoon sessions took place in the same large rectangular studio, which we estimated as fifteen meters long and eight meters wide. Six rows of desks were placed perpendicularly to the length of the room, and each row held about eight seats and faced the front of the classroom. There, a small desk was available to the professor, yet it was placed to the side and at a ninety-degree angle to the students' desks. The front wall also had a large whiteboard and a small projection canvas. In Figures 7.4.1 and 7.4.2, we present renderings based on our sketches and measurements of the studio.

As demonstrated in Figure 7.4.1, the first three rows of desks were each equipped with four desktop computers. Students seldom used these computers in the sessions we witnessed, yet the large 24-inch monitors took a significant amount of space and limited the professor's ability to make visual contact with the students.

* Given the delays we faced with some of the other case studies, in the case of a scheduling conflict we opted to prioritize their sessions' observation.

† A global outreach effort by the Procession Foundation, inviting members to organize local community events – day.processing.org.

***Figure 7.4.1.*** *A view of the classroom environment from the professor's perspective.*

The reverse perspective presented in Figure 7.4.2 demonstrates how students may have had difficulties following the professor. The projection screen, placed to the side, was too small to grant sufficient readability to students sitting beyond the first two or three rows. The professor often used a large font size to try to address the readability issues, but that also complicated the presentation of code on the screen. Often students asked the professor to scroll up and down the source code listings he was presenting.

***Figure 7.4.2.*** *Students sitting farther than the third row had visibility problems.*

We also note the studio had floor-to-ceiling windows on two of its sides and lacked any acoustic treatment. Therefore, the classroom often felt noisier and more crowded than warranted by the number of students in attendance. It was possible that students sitting in the farthest rows had difficulties understanding the professor's speech.

### 4.3.2 THE LECTURES

As summarized in Table 7.4.1, the first four sessions we observed were related to three distinct lessons, and the latter two were dedicated to students' laboratory practice and project tutoring (to be presented in §4.3.3). We thus describe the lessons we attended:

- INTRODUCTION TO PROGRAMMING AND PROCESSING, WEEK 2 We visited the faculty for the entire day and attended both the first formal morning (class I) and afternoon (class II) sessions, both attended by all students. Professor Edwin had taught a preceding unit about design software to most students; therefore, he skipped the introductions and dove straight into the lectures. We note that at this point in the semester, Erasmus mobility students were exploring available classes; the professor spoke in English during both sessions, often repeating in Portuguese what he had just said.

  After presenting the lesson plan and asking students to check SKETCH's Moodle regularly, the professor dove straight into comparing learning to code to learning Mathematics or Portuguese. Edwin presented slides with 'code examples' —cooking recipes, Sol Lewitt's artworks, Nicholas Feltron's data visualizations, or Stefan Sagmeister's dynamic identities—and defined algorithms as sequential sets of procedures, again mentioning the parallelism with recipes. He then presented Processing and asked whether all students had it installed in their laptops, as he requested via email. Edwin then ran the integrated development (IDE) software and explained how to access Processing's documentation through the Help menu. He demonstrated how to print 'hello world' to the console and how to draw a circle, taking the opportunity to discuss the language's coordinate system. He explained how to add comments to the code and told students their grades would reflect how well their code was commented. We saw some students were tinkering with the code, drawing ellipses of different shapes and sizes.

  Edwin then explained how to set the programs' window size and background color, how to set colors before drawing a shape, and discussed the execution order of statements (e.g., setting the color before drawing), emphasizing *"the computer is dumb."* He challenged students to take a few minutes to explore drawing shapes, telling them to read the documentation prior to posing questions. He added, *"now you know how to draw with code, the rest is just sweat."*

  The professor announced he shared the instructions for the first individual exercise on Moodle, in which students were to work individually and given four weeks to submit the code for a portrait of a historical figure related to computers. Edwin also gave students homework for the following week—to draw a robot with code. Some in the afternoon session protested, and the professor likened the drawing exercise to *"performing scales when you are learning the piano."* At this stage, though few minutes remained in the afternoon session, there was still one hour to go in the morning's, and students were invited to get started. A sizable number of the students felt lost about how to begin, and Edwin walked around the classroom to assist. One student told the professor her *"brain is blowing up,"* and several colleagues agreed they felt confused. Edwin insisted they should carefully read the reference documentation of all drawing functions.

ACTIVE MODE AND ANIMATION, WEEK 7 As the professor did not conduct a roll call, we estimate only half the students attended the morning session we witnessed. Edwin again spoke in English most of the time, to the benefit of the Israeli mobility students attending. Students had submitted their first individual exercise the previous week; Edwin told the class he was going to send emails asking some students for clarifications about their work.

The professor explained how to use Processing's 'active mode' to implement animation and interaction. He wrote code that drew a circle in the center of the screen, much like how he had started the first session, though using variables to define the coordinates; Edwin took the opportunity to explain variable scope. He then changed his code, so the coordinates were defined by Processing's internal variables referring to the current mouse position. Most students reproduced the code in the own computers, though we could see a couple distracted on Facebook and browsing unrelated websites. We saw students were pleased on running the program, which drew a circle that followed mouse movement; that was perhaps the first time they ran an interactive program that went beyond the possibilities of illustration software.

The professor moved to the topic of custom functions. He opened code from a preceding lesson that produced a drawing of a robot and demonstrated how to encapsulate that code, so the figure followed the mouse pointer. Edwin challenged students to contribute an explanation of function arguments, and to summarize he used electrical sockets as a metaphor. He then iterated his robot drawing code with additional arguments (e.g., size) to demonstrate. After pacing around the classroom to check whether anyone had difficulties, Edwin challenged students to modify the program so that a circle of robots followed the mouse. He drew some diagrams on the whiteboard about coordinate manipulation functions, which he hinted would be useful. Back at his desk, Edwin wrote the changes that met his own challenge, and as students reproduced his code, many adapted it to their own drawings for the first session's homework. The professor presented additional challenges and demonstrated their solutions: having a circle of slightly different robots, having two types of drawings following the mouse pointer.

After confirming students were successfully reproducing his examples, and with an hour remaining, Edwin asked students whether they wanted to continue to iterate on the latest example or would rather start a new program concerning motion graphics; yet he appeared unprepared as students chose the latter. Edwin drew a diagram of how to apply an 'elastic' effect to an image depending on mouse movement. Still, Edwin stumbled a few times during coding, visibly improvising and somewhat out of his comfort zone. After his first coding attempt did not work as intended, many students had stopped reproducing the code on their laptops. Professor Edwin opted to adjourn the demonstration, announcing it would be continued in the following session. He then reopened his robot sketch and showed additional improvements for the remainder of the session. He remarked students *"were talking to a guy who had once failed Math, and [coding] is not about mathematics, it's about the cool stuff they want to make."* One student was quick to offer her rebuttal: *"Yes, but we can't."*

■ LIBRARIES & OBJECT-ORIENTED PROGRAMMING, WEEK 9 We observed the afternoon session, which was attended by 23 students, according to the professor's roll call. Edwin lectured in Portuguese as there were no foreign-speaking students present. The professor had not prepared any slides this time. He told the class that while he was checking the profiles of his new followers on Instagram, he stumbled upon posts disparaging the curricular unit; there was nervous laughter from students. Edwin also warned students not to submit homework that plagiarized the code from his lessons.

The professor explained how to install external libraries in Processing and how to employ the Sound library to write a program that plays back an audio file. Students followed the demonstration, yet many struggled with the task of copying an audio file and placing it in their source code's folder. Edwin went around and helped. The classroom filled gradually with a cacophony as students were able to run the program, which made most noticeably pleased. Yet, one particular student was constantly late in his coding and was disruptive, interrupting with unrelated questions. Edwin quickly demonstrated the Video library, noting the similarities to Sound. He told *"there is a whole world to explore when you're at home,"* and noted students were required to employ audio or video in their group projects.

The professor then showed a program he wrote during the morning lecture, inspired by the operation of a player piano: small shapes moved from the left to the right of the screen, triggering different sounds when a shape intersected a vertical line. Edwin explained they would replicate the program, and in doing so, he would be introducing object-oriented programming. The professor explained the topic verbally and proceeded to code. With its growing complexity, Edwin struggled with maintaining a meaningful portion of code within the limits of the projector display; he tested a smaller font size, asking whether it was still readable. We saw students were helping each other filling out the gaps in their code.

Edwin allowed the class a break *"for mindfulness and understanding"* their code. He asked students to implement the movement of the shapes by themselves, yet few students rose to the challenge. Most took time to unwind and talk to each other. The professor thus implemented the rest himself, asking students to follow along. Edwin talked about the movie *Inception* to illustrate how objects can exist inside of other objects. While improving his code, the professor invited the class to suggest funny names for variables, yet some protested that they were not able to keep up with the changes. He discussed his changes with the students sitting in the front rows; we saw that students sitting farther back had dispersed and were either chatting or browsing the web. After running his completed program, Edwin demonstrated how modifying one value produced ten shapes moving across the screen instead of two, stating the usefulness of object-oriented programming lies in its scalability: *"it would be just as easy to have one hundred shapes."*

Edwin presented a summary of the lecture and admitted it had been the most challenging, noting that *"the syllabus is now over."* The remaining sessions, to resume almost a month later due to Easter, an ensuing national holiday, and the local academic week—would be dedicated entirely to supervising students' group project development.

✦ PROFESSOR'S BEHAVIOR Edwin presented some strategies and behaviors worth noting during the four lecture sessions we witnessed:

FIRST, the professor gave his lectures at a demanding, relentless pace. Edwin covered the entire syllabus in eight sessions so that the remainder would be dedicated to supervising the final project's development. He manifested his pace physically, as he would often alternate between sitting down at his desk to code and standing up to the front of the classroom to address the class, ask questions, or discuss his PowerPoint slides.* Each slide presented little content in large type, yet the decks were thorough, presenting definitions for concepts, code excerpts, visual examples, web addresses to related resources, as well as instructions for exercises. Edwin switched the projection display often: slides, source code, or the resulting programs. The projector's placement and resolution raised readability problems, and the professor often had trouble finding an adequate font size for presenting Processing code.

SECOND, Edwin had a dynamic and humorous manner of addressing his students. We did witness situations in which the professor took risks in his approach, as when he invited students to suggest funny names for the variables of his program. We also felt the jokes might have emboldened one student to affect insolent behavior in class, even though it was of little consequence at the time. Still, we believe the professor was mostly successful in capturing students' attention through his tone and language, given the demanding pace of his lessons.

THIRD, the professor always wrote comments in his code, and we felt the program listings he presented during his lectures were easy to understand. Though he met some slight problems while coding and needed to refer to Processing's online documentation, the professor was confident in presenting that as part of the task.

FOURTH, the professor often posed non-rhetorical questions to students during his lectures and was patient to wait for a correct answer. In jest, Edwin brought a bowl of candy to some of the sessions, which he would offer to students with a correct answer.

FIFTH, we felt that Edwin was able to provide students good explanations about some difficult subjects, employing a good set of metaphors. Yet, as seen when he was to demonstrate a motion graphic technique in his week 7 lecture, the professor struggled with improvisation, preferring to leave the explanation for another session. As we will see in the next subsection and in §4.7, the professor rehearsed his lectures on account of recording video lessons.

✦ STUDENTS' BEHAVIOR We witnessed a set of behaviors from students attending lectures which we believe are worth summarizing:

FIRST, though there were twelve computers available to students in the classroom, all but a couple chose to bring their own laptops to class. Few students were tempted by distractions such as Facebook or messages while the professor lectured.

SECOND, lectures almost ground to a halt when students had to deal with their operating systems—such as when they were asked to choose an audio file of their liking and place it inside the same folder holding their program's source code. It was noticeable that many struggled with basic computing tasks such as managing files and folders.

* We note the professor did not prepare slides for one of the sessions.

THIRD, during later lessons, some students struggled to understand the professor or to replicate code in their own computers, either due to the complexity of the topic or due to the aforementioned limitations presented by the classroom's acoustics and the projector screen. Students were often shy about asking the professor for clarifications and would rather prefer the help of a colleague.

FOURTH, students were noticeably tired halfway through the lessons. Though we witnessed few lectures to draw a definite comparison, we also felt students attending the morning sessions were less energetic and attentive than their colleagues in the afternoon class. On the other hand, the environment was noisier in the afternoon, and students were much more talkative, both to the professor but also between themselves.

### 4.3.3 PROJECT SUPERVISION

Sessions in the three weeks prior to the final project presentation were dedicated to supervising its development. Students had been given instructions that specified that their work had to relate to the theme of dynamic typography but also had to include elements of user interaction and audio or video manipulation. Students were to work in groups of up to three, though some opted to work individually.

During week 12, we spent a day at LIME and observed both morning and afternoon sessions. We witnessed 20 different discussions involving the professor and 32 individual students (23 female and 9 male). These conversations concerned 16 different projects, which were represented by an individual (4), a pair (6), or a group of three students (6).* In all but two short conversations in which individual students showed their ongoing work without posing or warranting further questions, the professor was able to dedicate over fifteen minutes to each discussion; yet we note that nearly half the students did not attend class that day. Edwin checked in on every student and group and had some time for renewed discussions (4) at the end of each session.

Professor Edwin showed concern that groups only showed working prototypes for about half the projects (8), as the final presentations were due in two weeks. We saw a few conversations in which students showed drawings or diagrams of their intentions (3), or in which they showed artworks they found inspiring (9); some groups presented an online tutorial which they intended to reproduce (4); in such situations, the professor called for additional work beyond the tutorial. Despite Edwin's instructions, students and groups found latitude within the project requirements. We witnessed discussions regarding animation techniques for kinetic typography (6), 3D graphics (4), generative visuals (4), simulation (3), audio visualization (2), and computer vision (2).

Students did not pose many direct questions to the professor. We only witnessed four situations in which Edwin was directly asked to help with some coding difficulties and two other situations in which he was asked for his opinion on whether it was possible to implement an idea. The professor drew diagrams on paper or the whiteboard (6) while giving his explanations, yet he also gave some verbal theoretical explanations (2). In one situation, he asked another student to explain a topic to their colleagues.

* These numbers do not line up exactly with the students attending as some were absent from their group's project discussion.

We found it worth noting Edwin used his own computer to demonstrate code (3), and though twice he also asked to use a students' computer to better read their code, we do note the professor did not write any code on students' laptops. The professor also discussed possible refinements to students' prototypes (6), recommended additional tools or libraries (4), discussed the project's aesthetics (3), or offered recommendations regarding teamwork (3) or work planning (1). We also witnessed situations in which the professor criticized students for plagiarizing their prototype from a tutorial (2), for their excessive ambitions (1), for their lack of initiative (1), or in which he complained that a group's code was hard to understand (1).

## 4.4 *Online learning management and educational resources*

Professor Edwin granted us guest access to SKETCH's Moodle learning management platform, which he employed to aid his teaching, in tandem with a public website and a YouTube channel, which had complementary roles. The professor restricted his use of online platforms to unidirectional communication, favoring email and office hours for discussions with students, as he explained in his interview (see §4.7).

- MOODLE Professor Edwin employed the learning management system provided by the university as a means to manage and share content related to the lessons and the activities taking place within the classroom, and which may therefore be characterized as restricted in access. The content was organized as follows:
  - HEADER SECTION, which the professor had customized. Edwin used the default announcements forum, in which he posted 33 topics with updates about sessions and the availability of new content online; however, students posted no replies. The professor also included a link to the unit plan, posted PDF documents with the unit's presentation and session calendar, and placed links to the weblog and the YouTube channel.
  - THREE MODULES related to the groups of lessons in which Edwin had subdivided the syllabus. As students' individual exercises or group projects were to result from each module's lessons, Edwin posted his work assignment instructions within the respective section. Each of these sections was subdivided into parts for each lesson. Within each subdivision, the professor shared links to relevant posts by others or articles in his weblog, as to specific *Coding Train* video tutorials by Daniel Shiffman.[*] In addition, Edwin posted resources such as PDFs of specific chapters from some of the books in SKETCH's bibliography. Crucially, the professor also shared the code he wrote during his lectures and each lesson's slide presentation, without additional notes; yet the professor posted lecture notes on his website.
- YOUTUBE CHANNEL As he explained in his interview, Professor Edwin prerecorded his lectures, both as a means for rehearsal and as to share them on an unlisted YouTube channel so students could later refer to them (noting he would only upload the videos

[*] shiffman.net

after giving the respective lectures). For easier viewing, the professor had divided his recordings into smaller videos related to specific topics, creating playlists to aggregate all videos related to a specific lecture. Hence, at the end of the semester, SKETCH's YouTube channel contained 34 videos with an aggregate duration that exceeded five hours. The average duration of each video was a little over 9 minutes, even though individual videos ranged from around 3 to nearly 18 minutes.

Bringing into focus the sessions we witnessed and described in the previous subsection, the videos related to the week 2 lecture introducing computer programming and Processing added up to a little over 29 minutes; the video playlist for the week 7 lecture about active mode and animation added to little over 40 minutes; however, the playlist for the week 9 lecture about external libraries and object-oriented programming added up to over 80 minutes, which testifies about the demanding pace we witnessed in that session. No other playlist exceeded one hour.

✦ THE WEBSITE Though the professor employed Moodle to share content related to classroom activities behind an authentication request, he favored his website for sharing public content to persist beyond the academic year. Edwin used WordPress and took advantage of the platform's ability to manage both a weblog and a set of static pages. The website was a work-in-progress, as some sections remained incomplete at the end of the semester; we understood Edwin would continue to add content in the future.* We took a snapshot of the website as it stood at the end of the semester using the HTTrack 3.49 software, as to describe its organization at the time:

- WEBLOG, which was the default homepage for the website. The professor added nine posts throughout the semester: two were updates about the website and the YouTube channel, three summarized the preceding lectures with links to relevant tutorials, and four included examples and resources that were relevant to students' ongoing work at the time; one post presented some of the previous year students' submissions.
- INTRODUCTION, containing a small text about the author and a link to the short documentary *The Art of Creative Coding*.† Professor Edwin also wrote a short text introducing Processing and giving instructions on how to install the software.
- AUTHORS & ARTISTS relevant to digital art and design practice. The professor organized links to the websites or Wikipedia entries of over one hundred people, categorized as conceptual artists (including Sol Lewitt, Bridget Riley, and Joseph Albers), digital pioneers (such as Vera Molnar, Frieder Nake, or Muriel Cooper), or computational artists and designers (such as John Maeda, Golan Levin, or Aaron Koblin). In addition, the section featured links to over forty Instagram accounts, the Creative Applications, OpenProcessing, and the DAM Digital Art Museum websites,‡ as well as to the OFFF and EYEO festivals.§

* At the time of writing, we indeed verified Professor Edwin continued to update his website.

† PBS Off Book (2013). *The Art of Creative Coding* [video]. YouTube. youtu.be/eBV14-3LT-g

‡ creativeapplications.net, openprocessing.org, dam.org

§ www.offf.barcelona, eyeofestival.com

- LESSON PAGES with annotated lecture notes. Some were quite thorough and included deep links to the Processing reference documentation throughout the text. Lecture notes also referenced relevant tutorials, often Daniel Shiffman's. Still, some of these pages remained as outlines or drafts at the end of the semester; we understood the professor would continue to complete them in future instances of the curricular unit.
- FUTURE EXPLORATION SUGGESTIONS, in which the professor organized links about other creative coding tools and programming languages, such as JavaScript, P5.js, Arduino, OpenFrameworks, Cinder, Pure Data, or Max.

## 4.5 *Students' projects*

Students presented 17 final projects, responding to the instructions that required them to articulate typography, interaction, and audio or video elements. Though the project's instructions specified groups of up to three elements, during development, the professor had invited students with similar projects to work together and allowed groups of four. Nine presentations took place during the morning session and the remaining during the afternoon. Students were granted fifteen minutes to present their work, which were followed by a few minutes of discussion. They were also asked to use the iMac computer available in the professor's desk and connected to the projector; therefore, groups had to ensure their projects would run on that computer. During the presentations, Professor Edwin sat in the front row alongside some students, taking notes on his laptop. He had placed a bowl of candy at his side.

We observed presentations comprising 3 individual works and 14 group projects—either developed by pairs (3), groups of three (5), or groups of four students (6). Overall, 47 students—36 female and 11 male—took part in creating the works we observed. One student presented one individual work in addition to his project with another colleague. We categorized projects according to formal characteristics, noting that most had mixed techniques, as required by the professor: works employed typographic animation (11), audio visualization (5), user interaction (5), three-dimensional graphics (2), generative visuals (2), physical simulation (1), and image processing (1). The student that took part in two projects presented a game as his individual work.

Most groups and individual authors began their presentations by showing design and artistic references that inspired their projects (12); many presented references that had been first shared by the professor, betraying a lack of research. Some also dedicated part of their presentation to introducing the external Processing libraries they used (2). One group presented a video about their project's possible practical uses, for which they designed outdoor advertising mock-ups. Other groups also presented possible uses for their work, including prints (3) and website mock-ups (2).

Some presenters projected confidence (6) and were likely to have rehearsed their presentations. In contrast, we felt students were nervous during two presentations. We witnessed single instances in which students admitted to difficulties programming or that their code required optimization. A single group concluded their presentation with a mention of future developments they would like to pursue.

In the ensuing discussion, Professor Edwin asked some presenters to explain their algorithms (3) or a particular function in their code (3). Though most had no problems answering, one group struggled to explain their algorithm. We witnessed 11 instances in which some aspect of students' work was clearly praised and 12 instances in which it was clearly criticized. Students received specific praise for their ability to code and solve problems (3), for their presentation skills (2), or for the attention to detail they put in their projects (1). Conversely, students received criticism for their presentation (7), for reusing most of the code in their projects (2), for *"having wasted"* an idea with greater potential (2), for their project's aesthetics (2), for showing little attention to detail (1), or for presenting a project which the professor considered simplistic (1). We also witnessed remarks not loaded with particular praise or criticism. These concerned the teamwork within group projects (6), the works' aesthetic merit (3), the works' programming (3), and the use of other tools besides Processing (1). Professor Edwin only offered remarks about future project possibilities in a single instance.

## 4.6 *Students' acceptance and use of computer programming*

Per our instructions, Professor Edwin asked students to complete our acceptance survey, described in **5**.4, both at the beginning and at the end of the semester, as described in **5**.5.1.5. Fifty-one local students* (83% of all enrolled) completed the first questionnaire during the second week of classes. The professor asked students to complete the questionnaire for a second time after their final presentations, yet only 17 students (28%) did so. We will report on these students' changing attitudes in §4.6.2.

⁂

Over the next pages, we will analyze responses to the first survey application ($n$=51). At the time, 14 participants (27.4%) claimed some familiarity with computer programming, 16 (31.4%) submitted they were learning how to code for the first time,† and 21 (41.2%) reported no knowledge of programming. Five participants reported they were repeating the curricular unit.

Of the participants reporting familiarity with programming, ten described it as 'basic,' and three described it as 'reasonable.' A single participant described himself as 'very proficient.' Asked to report all the applicable places and situations where they had learned to code, six mentioned autonomous online learning and four mentioned a prior higher education course; a single participant mentioned secondary school. In addition, though few reported which specific languages they were familiar with, three participants mentioned Processing and two mentioned JavaScript; Actionscript, C++, and Java were each mentioned once.

In Table 7.4.2, we present the first survey participants' gender, age group, and self-assessed daily computer use, grouped by their reported programming knowledge.

* Erasmus mobility students were waived from our surveys.

† Which may reflect autonomous exploration after attending the Processing Community Day.

***Table 7.4.2.*** *Students' characteristics and familiarity with programming (week 2 survey).*

| | | *No knowledge* (*n=21*) | | *Learning* (*n*=16) | | *Prior knowledge* (*n*=14) | | *Total* (*n*=51) | |
|---|---|---|---|---|---|---|---|---|---|
| | | *n* | *%* | *n* | *%* | *n* | *%* | *n* | *%* |
| *Gender* | Female | 13 | 61.9 | 12 | 75.0 | 9 | 64.3 | 34 | 66.7 |
| | Male | 8 | 38.1 | 4 | 25.0 | 5 | 35.7 | 17 | 33.3 |
| *Age group* | 18 – 20 | 18 | 85.7 | 10 | 62.5 | 7 | 50.0 | 35 | 68.6 |
| | 21 – 24 | 3 | 14.3 | 4 | 25.0 | 6 | 42.9 | 13 | 25.5 |
| | 25 – 34 | 0 | 0.0 | 2 | 12.5 | 1 | 7.1 | 3 | 5.9 |
| | 35 + | 0 | 0.0 | 0 | 0.0 | 0 | 0.0 | 0 | 0.0 |
| *Daily PC use* | < 1 hour | 0 | 0.0 | 1 | 6.3 | 0 | 0.0 | 1 | 2.0 |
| | 1 - 2 hours | 2 | 9.5 | 2 | 12.5 | 0 | 0.0 | 4 | 7.8 |
| | 2 - 4 hours | 9 | 42.9 | 4 | 25.0 | 8 | 57.1 | 21 | 41.2 |
| | 4 - 8 hours | 10 | 47.6 | 7 | 43.8 | 5 | 35.7 | 22 | 43.1 |
| | > 8 hours | 0 | 0.0 | 2 | 12.5 | 1 | 7.1 | 3 | 5.9 |

We conducted the appropriate analyses given the sample size and found a strong correlation between participants' age and a greater computer programming proficiency (Kendall's $\tau_b$=.49; $p$<.001). Nevertheless, we found no further relationships involving participants' individual characteristics and their reported programming familiarity.

### 4.6.1 STUDENTS' ACCEPTANCE AND BEHAVIOR

We computed means and standard deviations for the acceptance constructs according to the results of our reliability tests (see 5.5.3). To improve readability, we transposed the questionnaire's 7-point Likert scale to a new scale running from -3 (standing for full disagreement) to +3 (full agreement). We present participants' acceptance measures, grouped by their programming knowledge, in Table 7.4.3. We note that positive values reflect favorable opinions, except for *Anxiety*.

***Table 7.4.3.*** *Students' programming acceptance grouped by familiarity (week 2).*

| | *No knowledge* (*n=21*) | | *First time learning* (*n*=16) | | *Prior knowledge* (*n*=14) | |
|---|---|---|---|---|---|---|
| | Mean | St. dev. | Mean | St. dev. | Mean | St. dev. |
| *Performance Expectancy* | 1.10 | 1.36 | 0.15 | 1.59 | 0.28 | 1.65 |
| *Effort Expectancy* | -0.57 | 1.14 | -1.17 | 1.01 | -0.14 | 1.74 |
| *Attitude Toward Use* | 0.68 | 1.39 | 0.19 | 1.06 | 0.98 | 1.39 |
| *Subjective Norm* | -0.79 | 1.84 | -1.07 | 1.77 | 0.46 | 1.78 |
| *Self-Efficacy* | 0.89 | 1.16 | -0.28 | 0.93 | 0.60 | 1.60 |
| *Anxiety* (reverse scale) | 0.29 | 1.43 | -0.17 | 1.22 | -0.38 | 1.60 |
| *Behavioral Intention* | 0.98 | 1.26 | -0.58 | 1.51 | 0.21 | 1.61 |

Even though it can be ascribed to a variant interpretation of the survey measures (cf. Parameswaran, Kishore, & Li, 2015), we found participants learning to code for the first time harbored a set of views that stood out negatively. Despite the timing of this survey,

which may reflect the early initial impact of programming, these findings are aligned with both the preliminary and the other cases' surveys.

We also found a strong negative Pearson 2-tailed correlation between the age of participants with prior programming knowledge and their perceived voluntariness of programming ($n$=14; $r$=-.70; $p$=.012). Our tentative Kendall Tau analyses also yielded positive relationships between daily computer use and the *Self-Efficacy* of participants without any programming knowledge ($n$=21; $\tau_b$=.48; $p$=.009) and the *Performance Expectancy* of those who already knew computer programming ($n$=14; $\tau_b$=.65; $p$=.008).

✦ CORRELATIONS We looked for Pearson 2-tailed relationships between participants' acceptance of computer programming and their intention to use it in the future. Though we ought to emphasize the small number of participants within each group, we found significant correlations, which we present in Table 7.4.4:

***Table 7.4.4.*** *Significant Pearson correlations between students' acceptance and their Behavioral Intention (week 2). Lesser significances (p>.01) in italics.*

| | *No knowledge* ($n$=21) | | *Learning* ($n$=16) | | *Prior knowledge* ($n$=14) | |
|---|---|---|---|---|---|---|
| | *Correlation (r)* | *Significance (p)* | *Correlation (r)* | *Significance (p)* | *Correlation (r)* | *Significance (p)* |
| *Performance Expectancy* | *.45* | *.042* | .76 | .001 | .75 | .003 |
| *Effort Expectancy* | .60 | .004 | *.57* | *.021* | *Not significant* | |
| *Attitude Toward Use* | .65 | .001 | .72 | .002 | .70 | .008 |
| *Subjective Norm* | *.54* | *.012* | *.55* | *.035* | *Not significant* | |

We found no significant correlations between participants' *Self-Efficacy* or *Anxiety* and their *Behavioral Intention*. We looked for the relationships between students' intention and their reported programming use, as suggested by the UTAUT model (Venkatesh et al., 2003). We found no such significant correlations. It may be the case that the survey was applied prematurely, as it is likely that students were yet to start working for the unit during the second week of classes.

### 4.6.2 ACCEPTANCE AND BEHAVIOR AT THE END OF THE SEMESTER

Though we only obtained seventeen responses to the second application of our survey, we matched each participant's responses to their first survey's and calculated differences between the computed means* of each acceptance construct. We tested differences for normality before conducting the appropriate analyses. Through a Paired Samples t-test, we found suggestive evidence of a general increase in these participants' *Self-Efficacy* at the end of the semester ($t$=2.20; $p$=.044), from $M$=0.42; $SD$=1.36 to $M$=1.28; $SD$=1.11, but no further significant changes in acceptance. Wilcoxon Signed-Ranks tests also revealed that these participants reported both a higher frequency ($p$=.001) and a higher intensity ($p$=.004) of computer programming use at the end of the semester.

* Computed from the same items as in our analysis of the first survey application.

## 4.7 *Interview with the professor*

Professor Edwin granted the researcher a face-to-face interview, which followed the procedure laid out in 5.5.1.4. The interview took place two weeks after the last session and lasted over ninety minutes.

✦ THE VALUE OF LEARNING TO CODE Reiterating the start of his first lecture, Professor Edwin told us computer programming is a twenty-first century competency, as useful as Philosophy, Mathematics, or Portuguese. Edwin admitted it is *"difficult to explain to a normal person how [learning to code] is useful,"* and added, in jest, *"but it is the thing that gave me a job."* The professor believed his students' future work would be related to online media or platforms, wherein knowledge of computer programming is essential; he noted *"unicorn designers"* proficient at coding and visual communication were in great demand. He also believed programming and other IT competencies could have a positive impact on designers' *"working hours and quality of life."*

Noting he *"does not have the vision of an artist or of a performer,"* the professor believed programming would be useful to students harboring such ambitions as long as they commit to further exploration. In that respect, he saw the SKETCH curricular unit as a mere 'eye-opener' to the possibilities of computer programming.

✦ THE CURRICULAR UNIT PLAN The professor felt there was hardly enough time to meet the teaching objectives laid out in the curricular unit plan, though *"it would be impractical to have less ambitious objectives."* Lessons concluded with an introduction to object-oriented programming, and Edwin believed anything less would fall short of providing the minimal foundations to learn autonomously. The professor also explained each lesson introduced just a few new concepts, noting how it took six lessons to arrive at the point of teaching about multidimensional arrays.

Edwin believed that one semester of weekly three-hour sessions was not enough time for students to consolidate their learning. He suggested it would be important for a future curriculum to include a 'sequel' to SKETCH.

✦ THE WORKING CONDITIONS As most students brought their own laptops to class and Processing is free software, the professor saw no issues arising from a possible lack of computers in the classroom. Still, Edwin complained about the large classes: ideally, there should be three or four classes of no more than eighteen students. As it was, the professor felt the three-hour sessions were too short to supervise exercises and projects. He admitted becoming impatient during tutoring, noting that *"on average, we are talking about six minutes per student per lesson!"*

The professor also complained about the projection system and recognized the limitations inherent in using it to display code, though noting smaller classes would not be as subjected to readability problems. Edwin speculated about sharing his screen directly to students' computers using VNC* but admitted that could raise new problems due to their lack of IT proficiency.

* Virtual Network Computing, a protocol for remote operation and screen sharing.

✦ INSIDE THE CLASSROOM Professor Edwin tried to divide the three-hour sessions he dedicated to lectures into two parts. He felt students were tired after he introduced new content, thus he conceded the second part of such sessions for the completion of small exercises or discussing projects. Still, Edwin believed that in not dedicating the entirety of sessions to content exposition, he *"broke"* the rhythm of his teaching.

The professor told us he benefitted from 'rehearsing' his lessons. He sometimes advanced the next lecture during his office hours, but he also recorded his video lessons on the eve of his lectures in class. Edwin made sure his demonstrations in the classroom matched the video, which he would only publish after the lecture. He was looking forward to implementing a 'flipped classroom' methodology in the coming years as to dedicate more time to tutoring students.

Discussing his behavior while tutoring or supervising students' projects, Edwin stated, *"the question is whether you are coding with them or coding for them."* He observed some students did not iterate further upon his code, yet he nevertheless admitted his preference for seeing students presenting functional projects; he noted he took students' code originality into consideration while calculating their grades.

We had been curious by how Professor Edwin went out of his way to always use his own computer while tutoring. He admitted his main reasons were *"laziness and being comfortable,"* as he *"always struggles a bit when using students' environments."* He added:

> If [their question] is something simple, I quickly explain it. They reproduce it [in their own computers], or take a photo of the screen, and they are done. If it is something more complex, I get to keep the code, reuse the code. [...] I would comment, organize, and share it on Moodle.

At the beginning of the term, Edwin had no intention of sharing the code he wrote in class with the students: *"if they attended the lectures they'd write it themselves, and if not, they'd ask their friends."* The professor told us he changed ideas and was now thinking about *"building a repository and for this repository to grow year after year."* He was tired of repeating the same explanations while going around the room to check on students. Avoiding repetition was also a motivation for Edwin to pre-record his lessons on video, though another motivation was to address the limitations of the classroom projector:

> I spend half the session just standing there looking at them, right? Waiting for them to copy the code on the screen or scrolling up and down when they ask me... It does not make any sense, does it?

Professor Edwin described his step-by-step demonstrations as a 'defect': *"boring for the better students, while the worst think I'm going too fast."* Still, he preferred to wait for the slowest students and to give the faster additional challenges—e.g., *"can you make it draw a star?"* He saw coding demonstrations, both during lectures and besides students while tutoring, from *"the perspective that they learn by watching me use"* the computer.

We had not been invited to observe the office hours, but Professor Edwin was happy to discuss them. He was always visited by four students who were unable to attend sessions due to scheduling conflicts, and he gave them the necessary support for them to complete the unit. Others also visited, and Edwin explained good students were the most likely to take that additional opportunity to seek advice.

✦ EDUCATIONAL RESOURCES AND ONLINE VIDEO Students tended to ignore most of the textual educational resources Edwin shared on SKETCH's Moodle and website:

> Despite all the effort I had in building PDFs, preparing references, telling them which links to study before the next session [...] what they see is [Daniel] Schiffman's [YouTube channel]. They are a YouTube generation. Nobody reads anything. You know I am using multiple references—the O'Reilly pocket booklet, the handbook, the Web manual... and YouTube is all they see.

Though the professor believed video tutorials *"are not that efficient; sometimes you are there for ten minutes to see how to solve a problem when you could just read [the solution] in two minutes,"* he conceded he had *"decided to feed into it this year."* Edwin told us he even asked a colleague to help him record a video lesson with multiple cameras, yet he is unsure such production was necessary, given how easier and effective it is to record a lesson on his computer. Still, he was looking at improving the recording quality of his videos in the coming academic year. Edwin estimated, from visualization statistics, that about a third of students at SKETCH watched and paid attention to the video lessons.

✦ THE STUDENTS Professor Edwin saw no additional interest in learning to code since he gave his first Processing workshops fifteen years before. He told us that students' curiosity remains "*equally distributed across creative areas"* such as *"music, photography, illustration, or performance,"* programming being just one among them. Edwin also speculated whether students regard SKETCH as less important than other units taking place in the same term, as there was competition for students' attention from *"things such as illustration, which gives more immediate pleasure."*

Noting it is hard to pass judgment on his entire class, Professor Edwin praised his students' visual communication capabilities, adding *"there are students with intrinsic motivation [...] that understood they could use this knowledge for web and interactive work."* Yet, some of those students *"want to build complex things right at the start and then they hit a wall because support is hard to personalize"* due to SKETCH's time limitations. Noting he had a few exceptional students in that respect, the professor believed most lacked tenacity and quickly demotivated when faced with greater complexity: *"they are not stubborn and lack passion for exploring technology."*

We asked the professor about those situations in which we witnessed students struggling to manage the files in their own computers. Edwin confirmed and reported other similar situations, such as students attempting to run Processing straight from the ZIP file they had just downloaded without uncompressing the contents. The professor had realized many students reach higher education without sufficient IT training. We were tempted to ask the professor what he thought of the notion of 'digital natives,' which he emphatically discredited:

> They do not master the basics. I can tell you this because I teach an 'introduction to digital tools.' They do not know what a megabyte is, they do not know why an email would not arrive. It is completely transcendent. [...] They were supposed to have that preparation from basic schooling, but I do not feel they do. Simple things like dactylography, writing correctly on a computer... It is not that I see them [digital natives] as less prepared, but I think the problems remain.

Many students evince computer anxiety, Edwin told us, estimating that about a third were scared when they arrived for their first lesson, adding *"the word 'programming' is enough"* to trigger them. The professor told us he understood it takes time to lose that fear, as he also struggled when he first began learning to code. Hence, the first lessons are crucial, and students are often relieved once he presents works made in the previous year. Yet, Edwin noted such displays tempt some to imitate their senior colleagues' work.

The professor admitted he *"is not pedagogically equipped [...] to deal with students' mental blocks."* He tried to be playful as to put students at ease, hence *"all that goofiness with the chocolate and the candy."* The professor sometimes felt that students *"are there afraid of doing something wrong,"* and he struggles to motivate them in that case. Other students, *"usually two or three each year,"* are ambitious and present a different type of anxiety when facing the reality that their idea *"is going to take a long time to happen."*

Edwin saw no difference in attitude or anxiety between male and female students. He explained he had a student that *"manifested [her anxiety] physically,"* but so did a male student. Still, in his experience, female students tend to face their difficulties more silently. The professor also allowed those female students tend to be more dedicated but more modest than their male peers.

✦ PROJECTS AND EDUCATIONAL OUTCOMES As we saw in §4.5, students were given precise instructions for their work. Edwin told us he would have difficulty *"getting inside their projects"* when we asked about letting students propose their own ideas. He gave his first exercise—generative portraits—as an example of why he disagrees:

> I forced the *"let's make portraits about the history of computing"* theme. It would be a problem to let them choose. Ideally, I would ask for a communication design project: *"Use this specific program. This person has characteristics you should address."* I want to see how they adapt technically, visually, and metaphorically. I do not want to let them create everything by themselves. I have a problem with open projects. [...] They can present a blank screen and say it is a portrait of Malevich *"because he painted a white square on a white canvas."*

The professor allowed that perhaps a videogame project *"would be the ideal for the unit"* in regard to motivation; however, he considered that too would give students too much latitude in defining the project.

We also discussed how Edwin had specified intermediate submission milestones in his instructions for the final project. Nevertheless, he noted, *"eighty, ninety percent of students don't meet them, or even understand the purpose of organizing"* their project development in that way. Yet, the professor looked back on his experience as a student and conceded *"he would only gain that kind of discipline near to the end of the course."*

Professor Edwin told us students met the general objectives and he *"would grade the unit with a B,"* noting that taking second-year Design students *"from zero to object-oriented programming"* had been challenging. Moreover, Edwin had four students whom he considered as 'excellent' and whom he believes are interested in continuing to use programming in their own practice or their future studies. Yet, of the 61 local students, fifteen failed to submit the required work and were ungraded. The professor *"felt bitter"* whenever he received an email from a student confessing they had no work to show. He had to persuade a problematic student to submit the one exercise that was missing in

order to obtain a passing grade. Edwin explained he did not like students repeating the unit: *"if they are unmotivated and fail this year, next year it will be worse, and I don't want someone who will ruin the class' dynamic and will keep failing."* Only one student did not complete the unit due to failing grades after completing most work; Edwin explained students are likely to obtain a passing grade as long as they submit all required work.

The professor was nonetheless upset about receiving poor individual exercises in which students made very slight modifications, such as changing color values, to his own code, *"not even rewriting my comments in their own words."* A few students benefited from their group projects *"with some cool colleagues"* as to obtain a passing grade. Still, Edwin regarded most students as good, *"not technically evolved or very autonomous, but that can express their knowledge visually very well."* The professor told us he liked these students but often found their attitude frustrating and was concerned many would forget what they learned *"by halfway through the next semester."* Edwin believed the less motivated students would face difficulties in the curricular units related to web design; in contrast, those units *"will be like cutting through butter for ten or twenty percent."*

## 4.8 *Successes and limitations faced in this case study*

We were most pleased with our study of the SKETCH curricular unit, as we were able to conduct it as planned. We were able to obtain the necessary permissions in advance, and the responsible professor was supportive of our study and agreed to most of our requests. We thus summarize the successes and the limitations we faced:

***Table 7.4.5.*** *Successes and limitations faced in the SKETCH case study.*

| | *Planned* | *Unplanned* |
|---|---|---|
| *Successes* | Collected all types of data we sought and without delays. | Opportunity to attend students' project presentations.<br>Access to a plethora of unplanned online educational resources (e.g., video lectures). |
| *Limitations & Disappointments* | Suspected premature first application of the students' survey.<br>Bad participation percentage in the second students' survey.<br>Observation scheduling conflicts with other case studies. | - |

As shown, applications of our students' acceptance survey questionnaire were somewhat unsuccessful. Though the first survey application at the start of the semester obtained a high percentage of responses, we came to suspect the application was premature during our analysis. Furthermore, the low participation at the end of the semester had a negative impact on our ability to conduct longitudinal analysis.

# 5 WEAVE: PROGRAMMING FOR THE WEB

This case, which we shall henceforth refer to by the codeword WEAVE, concerns a curricular unit that introduces web programming to second-year students attending a licentiate in the field of Media Production ('AMP') at a faculty which we shall refer to as BLUE. Unlike the cases reported in the previous sections, we expected students enrolled in WEAVE to bring a prior familiarity with programming after having attended two units explicitly related to programming during their first year at the course.

We will present the licentiate and the host institution in the first subsection, and in §5.2, we will describe WEAVE's curricular unit plan, the unit's students, class schedule, and the responsible professor and his assistants. In §5.3, we will report our observations of lectures and workshop sessions. In §5.4, we will describe WEAVE's online educational resources. In §5.5, we will report our findings from the students' quantitative technology acceptance and use surveys. We also conducted interviews with the unit's professors, which we will present in §5.6. Finally, we will address the successes and limitations we faced while conducting this case study in §5.7.

## 5.1 *The course and its host institution*

BLUE is a large science and engineering faculty that is a part of a large university. This faculty alone is attended by over 6,000 students; of these, over a third are enrolled in licentiates, and over half are enrolled in integrated master's courses. The remaining students attend either one of the dozens of independent master's courses and doctoral programs offered at the faculty.

✦ THE COURSE The course offering the curricular unit under study grants a licentiate classified as in Audiovisual and Media Production under the CNAEF (see 4.1.3). Though unrelated to its real designation, we shall henceforth refer to the course by the initials 'AMP'. True to the nature of the host faculty, the course's three-year curricular plan emphasizes technological practices, in particular when compared to the similar courses offered by institutions with a Fine Arts tradition. It is revealing that the curriculum planners assigned over a third of the course's 180 ECTS credits to mandatory units in the Informatics area. Hence, this AMP plan of study includes a curricular unit introducing computer programming right in the first semester of the first year.

Recent curricular changes moved the units related to web design and development to a more central position in the curriculum, possibly reflecting the perceived demands of the job market. In this study, we focus on one such unit, which introduces web programming techniques and takes place in the first semester of the second year. As it is, the unit we refer to as WEAVE is preceded, in the first year of the course, by units about the foundations of computer science and about multimedia programming. In later semesters, the study plan includes advanced web and mobile programming units, as well as units about specific topics such as computer graphics or artificial intelligence.

✦ STUDENTS AND ACCESS According to official government data, there were over 200 students (56% of whom female) enrolled in the course in 2017/2018 (DGEEC, 2019). In the three-year period from 2016/17 to 2018/19, a little over 200 students were admitted to the course. Those newly arriving from secondary schooling brought good grades: the average candidacy grade for 2018/19 was nearly 17 (74th percentile), and the last student admitted under the general contest had applied with a candidacy grade exceeding 15.

According to the same official data, almost 12% of students who enrolled in 2015/2016 and 2016/2017 dropped out of the course before completing their first year. During the same two-year period, almost 100 students graduated with an average final grade of nearly 14.[*]

## 5.2 *The WEAVE curricular unit*

The unit we refer to by the codeword WEAVE is a part of the mandatory curriculum of the third semester of the AMP licentiate. According to the curricular unit plan (CUP) publicly available on BLUE's website,[†] students were advised to obtain a passing grade in the first-year programming units before enrolling; still, the course did not enforce strict precedence requirements. At the time of our inquiry, WEAVE had two weekly sessions: one two-hour theoretical lecture and one two-hour laboratory practice session.

✦ OBJECTIVES AND SYLLABUS According to the curricular unit plan (CUP), students were to acquire *"knowledge and competencies about the technologies required for building real-world applications for the internet"* in addition to several transversal competencies, such as autonomous learning, analysis, critical thinking, planning, teamwork, decision-making, and creative problem-solving. The CUP also included a section dedicated to a brief summary of WEAVE's syllabus:

> History of the Internet; Web, DNS, Browsers, and the anatomy of a Web Server; Client-side technologies (HTML5, CSS, JavaScript); Creation and maintenance of web interfaces with JavaScript; JavaScript libraries and components; Notions of Responsive Web Design; Server-side technologies (ex: PHP, CMSs).
>
> — *from the WEAVE curricular unit plan for the 2018/19 academic year*

As mentioned in our panorama of computer programming education within Portuguese public arts and design licentiates (see chapter 4), JavaScript (see 3.3.2.2 for more details) was the most common programming language taught in the units we analyzed. Fewer units covered PHP or any other server-side programming language, as those are not as relevant for creative practitioners working on a website's visual presentation. Though those are not programming languages, but rather standards for document structuring and formatting, we understand teaching HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets) ought to precede the introduction of JavaScript, as much of the work in that language consists of manipulating objects defined in HTML and CSS.

---

[*] We note the data also reveals that yearly admissions were increased by a third in 2016/17, which explains the relatively small number of graduations reported.

[†] Reference omitted as to comply with the participants' request for anonymity.

✦ TEACHING AND EVALUATION METHODS According to the unit plan, theoretical lectures were to present *"concepts, principles, and fundamental theories of internet technologies [as well as] practical exercises that arouse students' interest in these subjects and exemplify their application,"* while laboratory sessions were to let students *"acquire knowledge about tools and techniques to develop practical applications and, with the guidance of the teacher, develop a laboratory project."*

The evaluation methodology matched the unit's even division between theoretical lectures and practical sessions. Students were required to develop a project, working individually or in pairs to meet a set of milestones throughout the semester, which was to weigh 40% of the final grade. Another 40% was to result from students' outcomes in a theoretical exam at the end of the semester. The remaining 20% of the final grade would reflect students' participation in classes and their completion of small exercises during practical sessions. We note attendance was mandatory in at least half the theoretical lectures and the practical sessions.

✦ BIBLIOGRAPHY The curricular unit plan also included a section about WEAVE's recommended bibliography, which we present here:


- Abreu, Luís (2015). *HTML 5*. FCA. 9
- Abreu, Luís (2015). *JavaScript 6*. FCA. 7
- Berners-Lee, Tim (1999). *Weaving the Web: the past, present and future of the world wide web by its inventor*. Harper Collins.
- Lawson, Bruce & Sharp, Remy (2012). *Introducing HTML 5*. New Riders.
- Remoaldo, Pedro (2011). *CSS 3*. FCA. 5
- Segaller, Stephen (director) (1998). *Nerds 2.0.1: a brief history of the internet* [documentary film]. PBS Home Video.


It is worth noting the inclusion of three Portuguese language books covering most of the technical syllabus. Indeed, these were frequent recommendations found in the CUPs of units concerning web design and development within the universe analyzed in 4.2. Lawson and Sharp's HTML textbook, an alternative to Abreu's, was a more unique recommendation, only found in one other bibliography. Though outdated, we also find it interesting to note the unique inclusions of Tim Berners-Lee's account of the invention of the web and of a late 1990s documentary about the history of the internet. We should note there was a mismatch between WEAVE's public syllabus and bibliography, as there were no book recommendations concerning PHP or server-side programming.

✦ THE PROFESSORS The curricular unit was coordinated by the docent we shall refer to as 'Frank,' to comply with our commitment to participants' anonymity. Students were divided into separate classes for the laboratory practice sessions; Frank was assisted by the professors we shall refer to as 'Grace' and 'Hal' in leading said sessions. Though Frank only granted us permission to observe his own sessions, we still requested to interview Grace and Hal at the end of the semester. Hal, newly hired by BLUE, declined to participate, but Professors Frank and Grace accepted and shared details about their teaching experience and professional background during their interviews.

- PROFESSOR 'FRANK' told us he had almost fifteen years of teaching experience, having spent his first twelve years at a private university in another city before

moving to teach exclusively at BLUE. He has a doctorate in Computer Engineering and has done research on mobile and interactive systems as well as on the relationship between arts and technology. Frank explained he was solely dedicated to teaching and research, having no other professional activities. He was teaching three other curricular units over the 2018/19 academic year, though only WEAVE was focused on computer programming. Frank also told us he was proficient in writing Java, JavaScript, PHP, Pascal, C, C#, and C++ code.

- PROFESSOR 'GRACE' told us she had been teaching for over thirty years at BLUE's parent university, where she had completed a licentiate in Electrical Engineering. A Portuguese computing pioneer, Grace described a time when there was a single mainframe computer in the entire university. The professor obtained a doctorate in Computer Engineering and explained her present research interests concern educational simulations and e-Learning. She also told us she did not have any professional activities outside of academia. At the time, she was teaching three curricular units concerning computer programming besides aiding Professor Frank at WEAVE. Grace added she was proficient in several languages, naming Fortran, Pascal, C, C#, Python, PHP, Java, and JavaScript.
- PROFESSOR 'HAL,' who declined to be interviewed, had just started teaching at BLUE when we conducted our study. According to the faculty's public website, he had completed his Ph.D. in Computer Engineering a pair of years before. Before being hired as an assistant professor exclusively at BLUE, he had taught at two other higher education institutions.

✦ STUDENTS, CLASSES, AND SESSION SCHEDULE Though we were not provided with a detailed roster, Professor Frank told us there were 91 students enrolled in the curricular unit. For the laboratory practice sessions, the students were divided in four classes. Frank led two of the practical sessions and Grace and Hal each led one session. Three of the practical classes had their sessions taking place simultaneously and were immediately followed by the two-hour theoretical lecture attended by all students and led by Professor Frank. The other remaining practical session was also led by Frank and scheduled for a different weekday.

## 5.3 *Session observation*

During our initial contacts, the professor chairing WEAVE had advanced his lesson plan and explained the first half of the semester would be devoted to webpage layout and styling with HTML and CSS. As our study concerned students' contact with computer programming, we scheduled our first visit for the ninth week of the semester, in which the professor was due to give the first lecture about programming with JavaScript. Yet, that session overlapped with the first opportunity we had for observation in the PRIMER study, which had been under a significant delay (as described in §2.8) and which took place at a considerable distance. Therefore, we postponed our first visit to BLUE for one week. We revisited the faculty three weeks later, in which we observed another practical session and a lecture dedicated to PHP programming techniques.

### 5.3.1 CLASSROOM CONDITIONS

Unlike in the other cases, in which class activities were circumscribed to a single room or studio, WEAVE's sessions took place at either an auditorium or a computer room, depending on whether the sessions were theoretical or practical.

The laboratory practice sessions we attended took place in a small square room we estimated to measure no more than 20 m². As shown in the rendering, presented in Figure 7.5.1, which we based on our sketches and measurements, the room had twelve similar tables, each long enough to seat two people side-by-side. Six tables were placed along the side walls—three on each—while the remaining tables were put together in the center of the room. The side of the room facing the entrance had floor-to-ceiling windowpanes, which required curtains and shades, while the other side (not pictured) had a whiteboard next to the doorway. As shown in the figure, there was little room for movement, despite the few students attending sessions.

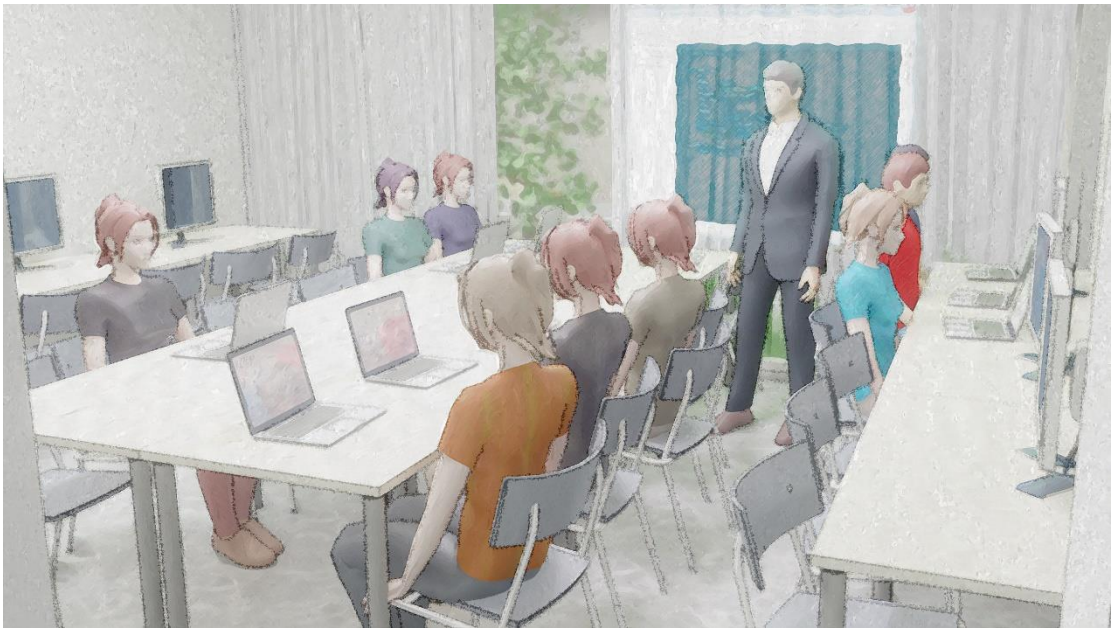


***Figure 7.5.1.*** *A view of the computer room from its doorway. Not pictured, a large whiteboard was placed alongside the wall next the entrance.*

In the two practical sessions we observed, the professor set his station at the center table, where he connected his own laptop to a portable projector pointed at a closed shade. This arrangement led several students to work with their backs turned to the professor. There were six desktop computers placed on the side tables, which looked dated and had bulky monitors which occupied some space. These computers sat unused during the sessions we attended, as students used their own laptops.

We were unable to ascertain whether the adjoining rooms where Professors Grace and Hal simultaneously led their classes' practical sessions had similar conditions; yet, given the building's layout, we are led to believe so.

⁂

The theoretical lectures with all students took place at an auditorium we estimated to have a capacity of 120 people. Six wide rows, each with around 20 seats and a continuous countertop, faced a stage we estimated at 12 meters wide and 2 meters deep. We present renderings of the auditorium in Figures 7.5.2 and 7.5.3, made according to our sketches and measurements.

***Figure 7.5.2.*** *A view of the auditorium from its entrance.*

As shown in Figure 7.5.2, the stage was equipped with a large desk placed to the side, an overhead projector, and a projection screen, which hung from the middle of the back wall. The screen was small in relation to the dimensions of the auditorium. We believe students sitting in the wings or in the farthest rows may have faced difficulties reading the text and the code presented.

***Figure 7.5.3.*** *A view of the auditorium from the stage.*

Figure 7.5.3 makes it clear the professor had an unobstructed view of the attendance. The continuous countertops in front of each row of seats gave students ample space to use their laptops or writing materials. The wood surfaces and paneling may have also helped with the auditorium's acoustics; sitting at a vantage point in the farthest row, the researcher felt the professor's speech was clear.

### 5.3.2 THE LECTURES

We witnessed two lectures, on the tenth and the thirteenth (and penultimate) week of classes. As mentioned, the topic of computer programming was not introduced until late in the semester, and we planned our observations accordingly.

Professor Frank had a calm and steady lecture presence. He alternated between standing up—to address students and explain his presentation slides—and sitting down at the desk—to write prepared code examples for students to reproduce. The professor mirrored his computer screen to the projector, and throughout the lessons he switched the display between his coding demonstrations and his presentation slides, written in English and which mostly consisted of small definitions and code excerpts. Despite his slides, we note Frank spoke in Portuguese. The professor also adjusted the font size of his tools—such as his web browser or the PHPStorm development software[*]—to ensure his demonstrations' readability. We thus narrate the lessons we witnessed:

- JAVASCRIPT EVENTS & CONTROL FLOW, WEEK 10 The professor told us 69 students had signed the attendance sheet for the lecture. Throughout the session, Frank alternated between his slides and the Codepen online code editor, which allows a side-by-side display of JavaScript and HTML sources and their resulting output. After introducing JavaScript programming the previous week, the professor dedicated this lesson to multiple topics: how to generate timed events, control flow statements (e.g., `if` and `else`, `for`, `while`), closures[†] (which Frank admitted not to master thus only demonstrated their use), and how to properly embed JavaScript code an HTML document as to ensure the script has full access to the webpage's elements.

  From the researcher's vantage point, he could see about a third of the students in attendance were using their laptops during the session. Some followed the professor's coding demonstrations, though a few others were distracted—one student was even playing a computer game, with the sound turned off. A couple more students were taking notes on paper. Nonetheless, most of the class was passively watching the lecture. We note Professor Frank proceeded briskly and did not allow much time for students to intervene and pose questions.

  Students appeared to be itching for the lecture to end; unprompted by the professor, a rumbling noise took over the auditorium as most students gathered their things and began to leave as the time neared one o'clock.

---

[*] Despite the name, the software also fully supports advanced HTML, CSS, and JavaScript editing features (e.g., syntax highlighting, automatic completion) – jetbrains.com

[†] Within the context of JavaScript, closures are also referred to as *anonymous functions*.

■ PHP RECIPES, WEEK 13 The last lecture of the semester (as a written test would take its place in the following week) was mostly dedicated to demonstrating common PHP development 'recipes' that students would find useful, continuing from the preceding week's lesson in which Frank had introduced the language's foundations. The professor told us 40 students had signed the attendance sheet. Frank again alternated between presenting slides and code during the lesson; he split the display, placing his text editor software on one side and the web browser with the resulting webpage on the other. As to underpin his demonstrations, the professor ran the XAMPP web server software* on his computer.

Professor Frank demonstrated several short PHP programs throughout his lecture: how to iterate through an array to output a list of keys and associated values, how to employ a condition to present different content depending on query strings,† how to write a custom function to return a given number of words from a text string, how to send an email with a webpage form's contents, how to read a Comma-Separated Values (CSV) file and render its contents as a table, how to implement a file upload form, and how to create a session to enable persistent user authentication.

As it can be inferred, Frank proceeded quickly through the aforementioned coding demonstrations. Though the auditorium was filled with the sound of keyboards, few students were, in fact, reproducing the professor's code in their laptops; most were distracted—one student was again silently playing a computer game, another was watching a movie—or writing an essay for another curricular unit. The professor jokingly admonished students that he would not ask whether the class had any questions.

✦ PROFESSOR'S BEHAVIOR Though we only witnessed two of his lectures, Frank showed consistent behaviors and stances which we believe are worth noting:

FIRST, the professor sat down at his desk through most of the lectures. He spoke calmly, and the auditorium's acoustics allowed his speech to reach the last seats clearly. Still, Frank's even tone failed to capture many students' attention. Moreover, he rarely sought that attention, apart from the situations in which students talked or made noise; then, the professor would stop talking for a few moments until the room was silent.

SECOND, Professor Frank nevertheless presented his lectures with a brisk pace, not pausing to check whether students were successfully reproducing his code. Frank made remarks about similarities between his present topic and the other programming languages students were presumed to know, yet he did not take the opportunity to pose questions. We believe we saw little effort to challenge students' passivity: though the professor often asked whether there were any questions before moving on to a new topic, we felt it as a rhetorical habit, as Frank advanced immediately.

---

* In the next subdivision we will see how students' issues in installing and running web server software in their own computers dominated the week's practical sessions.

† To the uninitiated, this relates to the part of a web address after an interrogation mark, where variables may be passed to the web server (e.g., example.com/blog.php?*page*=7).

THIRD, we found the professor's use of a split-screen display—presenting his code on one side and the resulting output on the other—to be noteworthy. Though limited by the screen size, the technique may improve readability and understanding.

FOURTH, we saw that Professor Frank mastered the subject matter. He wrote his demonstrations' code without mistakes, made use of good naming conventions, and embedded comments to improve readability.

FIFTH, we found the professor's demonstrations rather brief. Frank rarely iterated on previous examples, presenting instead 'recipe' after 'recipe'. We understand this may result from the little planned lecture time dedicated to teaching JavaScript (six hours) and PHP (four hours). Still, we had the impression some examples were too simplified and would benefit from greater attention, especially those concerning PHP scripting, which had been simplified to the point of embodying a disregard of data security (e.g., one example stored a password in plain text). Though the required correctness would complicate the lessons with topics of input sanitation and basic cryptographic functions, perhaps these would present good opportunities to engage the students.

✦ STUDENTS' BEHAVIOR With the caveat that we only witnessed two lectures, the first attended by 69 students and the second attended by 40 students, we witnessed a set of noteworthy behaviors during our observations:

FIRST, students were often late to arrive at the auditorium. We note the lectures were scheduled for immediately after three of the classes' practical sessions, and though the auditorium was in physical proximity to the computer rooms, students were likely to take a break, which we found understandable as the professor did not allow additional intermissions in neither the practical sessions nor the lectures we observed. Hence, students kept arriving for the first fifteen to thirty minutes of the lectures.

SECOND, only about half of the students attending the lectures we witnessed had turned on their laptops or were seen to take notes. Though a large portion was seen reproducing the professor's demonstrations, several were taking the time to work for other curricular units. A sizable portion had their minds elsewhere, checking Facebook or using chat applications. We also happened to witness the same student playing a computer game in both lectures, and another student was seen wearing headphones and watching a movie.

THIRD, many students passively watched the lectures, neither taking notes nor using their computers. Throughout both lectures, we witnessed a persistent light rumble of conversation from the back rows of the auditorium.

⁂

The professor had an abstract approach to computer programming, presenting several stand-alone examples rather than iterative demonstrations related to web designers' practice. An intention of avoiding the pitfalls of displaying complex code examples may have underpinned the professor's approach, alongside his own background as a computer engineer and BLUE's tradition as an applied sciences faculty. Still, many students paid little attention; we believed they were only attending the lectures in order to meet the curricular unit's 50% attendance requirement.

### 5.3.3 PRACTICAL SESSIONS

We witnessed two of WEAVE's practical sessions, both led by Professor Frank. These sessions were dedicated to the completion of small practical exercises or to students' ongoing project: a small website to be developed individually or working in pairs throughout the semester. The professor set milestones, requiring students to submit their work at each step: first, the website's structure and content as HTML documents; second, adding layout and styling through the use of CSS; third, implementing dynamic page behaviors in JavaScript. For the fourth and final submission, students were to submit the address to their live website, which was to include a simple server-side PHP script (e.g., a contact form), alongside their source code.

Our first visit to the computer room occurred during the tenth week. The session was attended by 12 students (10 of which were female), which were half of the 24 students assigned to the class, according to the professor. As Frank had first introduced JavaScript in the previous week's lecture, he asked students to spend the practical session completing a set of four short programming exercises he shared on the unit's website. The session took place after students submitted the second aforementioned milestone, and therefore the professor dedicated a significant part of the session to the discussion of students' ongoing projects. Frank sat down before his laptop and went through the submissions, calling out the respective students to sit beside him so he could present his questions and offer his remarks. He thus discussed seven projects involving ten of the students present. The professor gave notes regarding aesthetics (4), functionality (2), and possible references and tutorials (1). He also admonished a pair of students for not submitting their work. Halfway through the session, Professor Grace entered the room and conferred with Frank about her class, which she had been conducting similarly.

Professor Frank paced around the room during the remaining thirty minutes of the session, offering his assistance to students working on the JavaScript exercises. He helped four students with their coding difficulties, giving verbal explanations. In two situations, the professor asked the students to read some of their code aloud and to explain its meaning; in one such situation, the student had several difficulties and Frank did not hide his exasperation. One other student had already completed her exercises and asked for the professor's assistance in uploading her website to the university's servers; Frank helped her with the File Transfer Protocol (FTP) client software to do so.

⁂

We visited BLUE again three weeks later and witnessed the same class's penultimate practical session, attended by 13 students (12 of which were female). Though Professor Frank had introduced PHP in the previous week's theoretical lecture, he began the practical session by instructing students to download and install the XAMPP web server package* so they could test their projects' server-side scripts on their own computers. The professor added that Mac users might experience some difficulties in configuring the web server and told students he would be willing to assist. Frank asked students to

* A distribution that includes the PHP interpreter, the Apache server, and the MariaDB SQL database, and can be obtained from www.apachefriends.org.

complete the small programming exercises had he shared on WEAVE's website once they install the software. However, difficulties with the software's installation turned out to take the entirety of the session for most, and Frank was fully occupied in assisting it. As in the previous session we witnessed, Professor Grace also came into the room to confer with Frank, and it was understood many students in her class were having the same software installation difficulties.

While many students and the professor were caught up in web server installation and configuration difficulties, a few students were able to proceed to the PHP coding exercises. The researcher pace around the room and saw five students' who spent some time reading and reproducing online tutorials. One student explained to her colleague how to write a 'hello world' script in PHP; the same student was also seen helping two other colleagues configuring XAMPP in their Mac laptops. Nonetheless, installation problems were proving intractable for students with a more recent operating system, and five students were seen distracted—browsing the news or Facebook—while waiting for the professor's assistance. Frank sat down at his computer and searched online for solutions, turning off the projector as it had been mirroring his screen. The professor asked to use the computer of one of the students having difficulties so he could test one of the solutions he read about, and eventually announced the class he would email instructions on how to get the webserver running during the afternoon. At that point, it was time to leave for the theoretical lecture.

⁂

While completing their exercises, some students made use of the professor's resources, browsing the tutorials on the unit's website or revisiting previous lectures' slides. Others preferred to look for external resources; one such student was seen in both sessions wearing headphones and watching JavaScript lessons on Codecademy. Another student, exhibiting a clear familiarity with web programming, took on the role of an informal assistant and also paced around the room to help her colleagues, especially during the PHP session. After that session, we felt that dedicating such a significant part of the laboratory practice to software installation was not optimal and may have frustrated some of the students at such a crucial point in the semester.

Though we only watched four hours of laboratory practice, we felt the professor favored some students and did not hide his exasperation at others' lack of progress or understanding of the subject matter. We also witnessed absenteeism had been high in the face of a mandatory attendance requirement. Though we were not provided any data concerning the unit's outcomes, we suspected WEAVE would maintain its failure rate: there was a significant mismatch between students enrolled in the unit (91) and students arriving at the AMP course (69 students on average over the previous three years*).

* Of which we note the not-insignificant first-year drop-out rate. Hence, the number of students enrolling in WEAVE for the first time was expected to be closer to 60.

## 5.4 *Online educational resources*

Professor Frank made use of a single public website, which he gave us the address for, to support WEAVE's lessons and activities. The professor built the site in the Codepen platform, taking advantage of the platform's ability to manage webpages and embed code editors supporting HTML, CSS, and JavaScript. The website had been written in English, perhaps to the benefit of international students (though none attended the practical sessions we observed). Using the HTTrack 3.49 software, we captured a snapshot of the website at the end of the term as to describe its organization at the time. The website's homepage consisted of a simple table, in which each row related to each week of classes. Links to content were presented in three columns:

- LECTURE SLIDES, which were hosted on Google Drive and not downloadable. The decks were shared as presented, without notes. As mentioned, most slides either consisted of small code excerpts or definitions and were not easy to understand outside of the context of the lecture, without the accompanying explanation.
- LECTURE NOTES, which were thorough and embedded dynamic code examples throughout the text, as facilitated by Codepen. Some of the first lecture notes also included instructions on how to complete exercises using this online platform, or in the alternative, how to install WebStorm or PHPStorm, which were the professor's recommended development tools. We note Professor Frank had only completed the notes related to his first few lectures, and several pages—which included those concerning the JavaScript and PHP lessons—remained incomplete at the end of the semester. At the time, we understood the professor would continue to improve his website in the future.
- PRACTICAL EXERCISES, for which Frank shared thorough instructions breaking each exercise into five or six small tasks. Many of these tasks were presented as to employ the Codepen embedded code editor, which enabled students to solve them directly on the page, as long as they registered as users of the platform first.

BLUE's parent university also had its own proprietary learning management platform. As we will see in §5.6, Professor Frank criticized the platform and did not use it in support of WEAVE, except as a means for students to host their websites and as an alternative for uploading their practical exercises.

## 5.5 *Students' acceptance and use of computer programming*

Per our instructions, Professor Frank forwarded our survey questionnaire to students' institutional email addresses in two distinct moments at the beginning and at the end of the semester, as described in **5**.5.1.5. Fifty-one students (56% of all enrolled) heeded the request and completed the first survey during the third week of classes. However, only twelve of those participants completed a second survey application during the last week of classes. Though the small participation prevents robust longitudinal analysis, we will report on these specific students' changing attitudes in §5.5.2.

⁂

Over the next pages, we will focus on the responses to our first survey at the beginning of the semester (*n*=51). Though 29 participants (56.9%) reported a prior familiarity with computer programming, we were surprised that 21 participants (41.1%) submitted that they were learning to program for the first time, as these AMP students were expected to have attended two computer programming units during their first year at the course. One single participant claimed to have no knowledge of computer programming and we chose to discard their responses from our analyses. We are unable to explain this level of reported lack of prior familiarity with programming at that stage in their studies. In addition, four participants also reported they were repeating the unit after previously failing to obtain a passing grade.

Of participants claiming prior familiarity with programming, thirteen described their knowledge as 'basic,' another thirteen described it as 'reasonable,' and three participants claimed to be highly experienced. We also asked these students to list the additional places and situations where they had learned or were presently learning to code. Nine participants mentioned another higher education course (which we find noteworthy), secondary school was mentioned five times, and online learning sites were mentioned twice; moreover, paid lessons and unspecified self-education were each mentioned once. Though not many participants answered our question regarding their familiarity with specific languages, Processing was mentioned 14 times, Java and PHP were each mentioned six times, JavaScript was mentioned by five participants; Arduino C, C++, Pascal, and Visual Basic were each mentioned once.

We present validated participants' (*n* = 50) gender, age group, and daily computer use, grouped by their programming knowledge, in Table 7.5.1:

***Table 7.5.1.*** *Students' characteristics and familiarity with programming (week 3 survey).*

| | | *Learning* (*n*=21) | | *Prior knowledge* (*n*=29) | | *Total* (*n*=50) | |
|---|---|---|---|---|---|---|---|
| | | *n* | *%* | *n* | *%* | *n* | *%* |
| *Gender* | Female | 14 | 66.7 | 15 | 51.7 | 29 | 58.0 |
| | Male | 7 | 33.3 | 14 | 48.3 | 21 | 42.0 |
| *Age group* | 18 – 20 | 10 | 47.6 | 13 | 44.8 | 23 | 46.0 |
| | 21 – 24 | 8 | 38.1 | 11 | 37.9 | 19 | 38.0 |
| | 25 – 34 | 2 | 9.5 | 3 | 10.3 | 5 | 10.0 |
| | 35 + | 1 | 4.8 | 2 | 6.9 | 3 | 6.0 |
| *Daily PC use* | < 1 hour | 0 | 0.0 | 0 | 0.0 | 0 | 0.0 |
| | 1 - 2 hours | 1 | 4.8 | 3 | 10.3 | 4 | 8.0 |
| | 2 - 4 hours | 3 | 14.3 | 5 | 17.2 | 8 | 16.0 |
| | 4 - 8 hours | 12 | 57.1 | 12 | 41.4 | 24 | 48.0 |
| | > 8 hours | 3 | 14.3 | 9 | 31.0 | 12 | 24.0 |

We conducted the appropriate analyses to tentatively look for relationships between participants' personal characteristics and their programming knowledge or daily computer use, or between these two latter items. We found no significant relationships.

### 5.5.1 STUDENTS' ACCEPTANCE AND BEHAVIOR

We computed means and standard deviations for the acceptance constructs according to the results of our reliability tests (see 5.5.3). These are presented in Table 7.5.2, grouped by participants' programming knowledge. For easier readability, we transposed the original 7-point Likert scales to run from -3 (standing for full disagreement) to +3 (standing for full agreement). We note positive values, therefore, reflect a more favorable opinion from participants, except for *Anxiety*.

***Table 7.5.2.*** *Students' programming acceptance grouped by familiarity (week 3).*

| | *First time learning* (*n*=21) | | *Prior knowledge* (*n*=29) | |
|---|---|---|---|---|
| | Mean | St. dev. | Mean | St. dev. |
| *Performance Expectancy* | 0.74 | 1.67 | 1.59 | 1.16 |
| *Effort Expectancy* | -0.69 | 1.16 | 0.10 | 1.24 |
| *Attitude Toward Use* | 0.24 | 1.69 | 1.14 | 1.37 |
| *Subjective Norm* | -0.07 | 2.00 | 0.52 | 1.84 |
| *Facilitating Conditions* | 0.51 | 1.57 | 0.86 | 1.19 |
| *Self-Efficacy* | 1.14 | 1.33 | 0.80 | 1.04 |
| *Anxiety* (reverse scale) | 0.05 | 1.54 | -0.49 | 1.35 |
| *Behavioral Intention* | 0.75 | 1.70 | 1.47 | 1.42 |

We applied the appropriate analyses to look for associations between participants' individual characteristics and their acceptance and use measurements. Among those learning to program for the first time (*n*=21), we found Pearson 2-tailed correlations suggesting older participants were likely to harbor worse expectancies of *Performance* ($r$=-.47; $p$=.031) and *Effort* ($r$=-.53; $p$=.014) and to be more critical of the *Facilitating Conditions* ($r$=-.61; $p$=.003); in addition, older first-time learners were likely to perceive programming within the curricular unit as less voluntary ($r$=-.51; $p$=.020). No such relationships were found among the responses of participants that claimed a prior familiarity with programming.

Among participants already familiar with programming (*n*=29), we found Kendall Tau correlations between their perceived voluntariness of use and their *Performance Expectancy* ($\tau_b$=.34; $p$=.025), *Effort Expectancy* ($\tau_b$=.54; $p$<.001), *Attitude* ($\tau_b$=.38; $p$=.014), *Subjective Norm* ($\tau_b$=.34; $p$=.030), *Self-Efficacy* ($\tau_b$=.39; $p$=.010), and *Intention* ($\tau_b$=.44; $p$=.004). However, among the participants learning to code for the first time (*n*=21), we found no such correlations with perceived voluntariness, with the exception of better regard of the *Subjective Norm* ($\tau_b$=.46; $p$=.007) and a likely *Behavioral Intention* to use computer programming in the near future ($\tau_b$=.39; $p$=.026).

✦ CORRELATIONS We also looked for correlations between participants' acceptance of computer programming and their *Behavioral Intention* of coding in the near future. Through a Pearson 2-tailed analysis, we found relationships suggested by the UTAUT model, which we present in Table 7.5.3.

***Table 7.5.3.*** *Significant Pearson correlations between students' acceptance and their Behavioral Intention (week 3). Lesser significances (p>.01) in italics.*

| | *First time learning* (*n*=21) | | *Prior knowledge* (*n*=29) | |
|---|---|---|---|---|
| | *Correlation (r)* | *Sig. (p)* | *Correlation (r)* | *Sig. (p)* |
| *Performance Expectancy* | *.46* | *.038* | .83 | < .001 |
| *Effort Expectancy* | | *Not significant* | .66 | < .001 |
| *Attitude Toward Use* | *.46* | *.036* | .64 | < .001 |
| *Subjective Norm* | .67 | .001 | .58 | .001 |
| *Facilitating Conditions* | *.53* | *.013* | .68 | < .001 |
| *Self-Efficacy* | *.47* | *.033* | | *Not significant* |

We note that we found no significant associations between students' *Anxiety* and their *Behavioral Intention*. We also looked for correlations between students' intention and reported computer programming use, as suggested by the UTAUT model. Though no relationships were found between intention and usage frequency among participants learning to code for the first time (*n*=21), as these students are presumably yet to form a habit they can assess, we found a likely correlation between those participants' *Behavioral Intention* and their reported intensity of programming use ($\tau_b$=.42; *p*=.017).

Participants already familiar with programming reported a *Behavioral Intention* that had persuasive correlations with their reported programming use frequency (*n*=29; $\tau_b$=.39; *p*=.009) and intensity ($\tau_b$=.58; *p*<.001).

### 5.5.2 ACCEPTANCE AND BEHAVIOR AT THE END OF THE SEMESTER

As mentioned, we were disappointed that only twelve of the first survey participants completed the second survey at the end of the semester. We matched each participant's set of responses to their first survey and looked for differences between the computed means of each acceptance construct, which we tested for normality before conducting the appropriate analyses.

Hence, a Paired Samples t-test revealed a likely overall increase in the participants' *Performance Expectancy* at the end of the semester, from *M*=0.91; *SD*=1.67 to *M*=1.33; *SD*=1.59 (*t*=2.64; *p*=.023). We found no other significant changes in these students' acceptance or intention to program.

## 5.6 *Interviews with the professors*

Though we considered face-to-face interviews desirable, Professors Frank and Grace only accepted to be interviewed via email, whereas Professor Hal declined to grant us an interview. We followed the procedure described in 5.5.1.4. The professors took time to reply after additional reminders, and their answers were rather structured and concise.

✦ THE VALUE OF LEARNING TO CODE Professor Frank believed programming is an important, though not fundamental, component of digital literacy, whereas Professor Grace told us learning to code provides students with *"logical and critical thinking skills they may apply in other areas and to solve transversal problems."* Nonetheless, Frank told us he regards computer programming as an essential skill for designers:

> Even if one does not expect designers in general to code (and if this is debatable), it is clear to me a large majority will need to communicate with programmers. Knowing some programming will help them have a dialogue with the coders that will probably be a part of their teams in the future.

Frank did not believe students had a growing interest in computer programming in recent years, though there had been progress in the accessibility of programming languages and tools, mentioning Processing as one such example. Professor Grace was more positive, as she believed students were increasingly aware that programming skills will grant better job opportunities "*even outside of 'technological' domains."*

✦ TEACHING CONDITIONS Professor Frank told us the working conditions had been adequate, though he wished the computer rooms assigned to the practical sessions had been more spacious. Both Frank and Professor Grace criticized the few hours of contact assigned to WEAVE in the curricular plan. Professor Frank also explained BLUE's web services and learning management system was hard to use, complaining that students had wasted precious time configuring their web space and submitting their work.

✦ THE STUDENTS Professor Grace did not doubt her students' sensitiveness to the importance of learning computer programming. Professor Frank was not as positive, explaining that some *"do not overcome their 'fear' and keep [commitment] to a minimum."* However, Professor Grace was unsure about students being increasingly autonomous in their learning, whereas Professor Frank was positive in that regard, noting the greater availability of online resources and tutorials.

We asked whether the professors noticed any differences in students' attitudes that may relate to their age or gender. Noting he had not paid much attention to it, Frank told us he felt his better students were *"slightly likelier to have been male."* Professor Grace did not comment on her students; still, she admitted gender differences were noticeable in the past, yet those differences *"are fading globally—there are no longer jobs for men and jobs for women."* Grace also felt older students are *"more motivated for work and study than younger students because they understand life differently,"* and such a difference is noticeable in their attitude to learning to code.

✦ CLASSROOM PRACTICE Professor Grace recognized that *"programming is not easy nor quick to learn for most people."* She believed students are not always aware of how *"one needs to make many mistakes,"* and told us her main strategy to address difficulties is to pass on that reality to students:

> Then I coach them along that way, trying to help them overcome their difficulties without judging or issuing negative comments. To demonstrate ideas, I often employ visual diagrams and examples from their reality.

In Professor Frank's experience, students rarely express anxiety or a negative attitude toward programming. However, when that happens, *"students stop attending classes, and it becomes impossible to help them."* Frank told us his strategies for helping the students that persevere and keep going to class:

> My strategies are to offer students positive reinforcement on any small success in solving problems and to insist on *"divide and conquer"*—showing them how to break down a problem into smaller parts and leading them to focus on each individual part.

Professor Frank also addressed students' absenteeism, admitting "*there is a stigma about theoretical sessions."* We were led to believe the low attendance in the practical sessions we witnessed may have been exceptional.

✦ EDUCATIONAL OUTCOMES Professor Frank believed the semester went well and that his students achieved the unit's minimum objectives, and that they had met the website project's milestones and submitted their exercises on time. Professor Grace reported a somewhat different experience with her class: whereas, on the one hand, she complained about students' tardiness in submitting their work, on the other hand, Grace revealed greater enthusiasm about her class, which she generally described as *"highly committed and autonomous."*

In contrast, Professor Frank believed only a small set of his students demonstrated an ability to learn and work autonomously, *"in the sense of finding good solutions to problems not addressed in class."* He explained many students struggled with JavaScript, despite having acquired computer programming foundations in the first year. He also criticized students for presenting solutions that were "*based on web searches and applied without mastery."* Nevertheless, Frank commended students' creativity, adding most had been interested in building their website.

## 5.7 *Successes and limitations faced in this case study*

Though we obtained the necessary permissions before the start of the academic year, and the responsible professor forwarded us his lesson calendar so we could better organize our observations, we still faced limitations while conducting this case study. Some were inherent to WEAVE's lesson plan, which brought the subject of computer programming only during the last five weeks of the term and thus limited our window for observation. Noting this particular inquiry met no unplanned events and that we were able to gather most data we had planned, we faced some limitations, which we summarize:

- Within the limited time window for observing lectures and practical sessions related to programming, we faced scheduling conflicts with another of our case studies. Hence, we were unable to observe one of the sessions we would have preferred, given our analysis of the unit's lesson plan.
- We were denied the observation of practical sessions led by the assistants. Hence, we were prevented from comparing different professors' pedagogical strategies.

- Though in reasonable number, a disappointing percentage of students (56%) completed our first survey application. We then faced bad participation at the end of the semester, which impacted our ability to conduct longitudinal analysis.
- We also faced difficulties in interviewing the professors. The responsible professor and one of his assistants only consented to an email interview and took time to reply to our questions after repeated reminders. The other assistant did not grant us an interview.

This case, which we shall refer to by the codeword TOUCH, concerns a curricular unit taking place in the final semester of a licentiate in the area of Media Production ('AMP') offered at a school which we shall refer to as ROSE. Following from a unit in the preceding semester that introduces students to computer programming, TOUCH is aimed at teaching skills and techniques to support students' development of their final course projects, led at another curricular unit taking place simultaneously. Given its supporting role, TOUCH presents a vessel to hold a syllabus comprising the techniques the unit's professor and the course coordinator deem most relevant at the time. In the instance investigated, students were taught to program 3D graphics and to build and code custom physical interfaces for real-time interaction.

In the following subsection, we will present ROSE and the licentiate offering the curricular unit. In §6.2, we will turn our focus to TOUCH and will present its unit plan, the professor, and the students. In the subsequent subsections, we will present our findings: in §6.3, we will describe the observations we made inside the classroom; in §6.4, we will report on our visit to a public exhibit of students' ongoing projects and on the final presentation session. In §6.5, we will report on students' technology acceptance and use survey. We conducted an interview with the professor, which we will report in §6.6. We will conclude this section in §6.7 by addressing the successes and limitations we faced while conducting this case study.

## 6.1 *The course and its host institution*

ROSE is an arts and design school attended by over 1,500 students in a purpose-built campus away from the rest of the parent polytechnic institute's facilities. Three-quarters of these students attend one of the multiple licentiate courses on offer in the fields of fine arts, performing arts, audiovisual production, multimedia production, and design. The school also offers technical vocational and independent master's courses in the same areas.

✦ THE COURSE The course offering the curricular unit under study grants a licentiate classified as in Audiovisual and Media Production under the CNAEF (see 4.1.3). Though unrelated to its real designation, we shall henceforth refer to the course by the initials 'AMP'. It features a three-year curriculum which is divided, from the second year onward, into Design and Multimedia branches. Students at either branch are also given two options regarding the final semester of the course: they can either develop a final project at the school or complete an internship.

At the time of our research, the course featured no curricular units explicitly named in relation to computer programming, yet we found the subject as part of the syllabi of units taking place in the first and the second semester of the third year of the Multimedia branch curriculum. Said units, of which TOUCH was the curricular unit taking place in the second semester, were explicitly meant to introduce and develop programming techniques in support of students' final project development, led in parallel 'Project' units.

✦ STUDENTS AND ACCESS According to official data, there were over 300 students (62% of whom were female) enrolled in the course in the 2017/18 academic year (DGEEC, 2019). During the three-year period from 2016/17 to 2018/19, almost 300 new students had enrolled in the course; we note, however, that the data does not discriminate between the Design and the Multimedia branches. Students newly arriving from secondary schooling brought reasonable grades: the average candidacy grade for those admitted at the start of the 2018/19 academic year had been a 15 (59th percentile), while the last student admitted under the general contest had a candidacy grade over 14.

According to the same data, 11% of students newly enrolled in 2015/16 and 2016/17 dropped out before completing their first year, while nearly 170 students graduated during the same two-year period, with an average final grade nearing 15. We note that we were unable to find values only relating to students that completed the Multimedia branch of the course.

## 6.2 *The TOUCH curricular unit*

The curricular unit, which we refer to by the code TOUCH, is a mandatory part of the curriculum of the second semester of the third year for students at the Multimedia branch of the AMP course opting to complete their licentiate through the development of a final project. Though TOUCH has its autonomous curricular unit plan (CUP) and separate evaluation requirements, it was understood as having an ancillary role to the 'Project' unit taking place simultaneously, which required students to develop a final project which mimics marketplace conditions. 'Project' has a bigger curricular weight (twelve ECTS credits to TOUCH's six) and requires students to dedicate more time, both in face-to-face sessions (seven hours per week) and in autonomous work. At the time of our study, TOUCH was scheduled as a weekly four-hour session.

This dynamic between two distinct curricular units, in which one provided the skills to support another leading students' project development, continued from the first semester. The first-semester unit that preceded TOUCH had the same responsible docent, 'Ivan,' who at the time introduced computer programming to his students using both Processing and P5.js.

✦ OBJECTIVES AND SYLLABUS Because of the aforementioned interdependence with the 'Project' unit, TOUCH's learning objectives strive to meet the needs and close gaps in the knowledge required for students' successful development of their final projects.

According to the CUP available on the school's website* at the time of our study, TOUCH's objectives were summarized as to *"teach 3D graphics programming and physical computing for the design of digital artifacts integrating conceptual and interface development competencies."* TOUCH's unit plan also referenced a set of transversal skills to be acquired, such as research, planning, communication, and critical thinking, as well as *"developing competencies for autonomous and authorial work."* To these ends, the unit plan included a detailed syllabus, from which we reproduce the main topics:

* Reference omitted as to comply with the participants' request for anonymity.

> 3D-WebGL digital artifacts in P5.js. Tridimensional shapes, spatial placement and rotation, lighting, materials and textures, 3D file management. Physical computing, the Arduino hardware, and its programming environment. Tangible interfaces. Electronics. Sensors, actuators, resistors, using the 'breadboard'. Comparison between the P5.js and Arduino programming syntax. Digital and analog inputs and outputs. Clock functions. Serial communication.
>
> — *from the TOUCH unit plan for the 2018/19 academic year, translated and abridged*

Students were to learn and develop a small project integrating P5.js and Arduino: the first (see 3.3.2.3 for more details).  is a free and open-source JavaScript library aimed at educators and visual artists which allows users to write and run code that is formally like Processing's inside a webpage. P5.js also offers a free online editor at editor.p5js.org, allowing users to write, store, and share their programs (or 'sketches'). As for Arduino (see 3.3.2.6), it refers to a range of mutually compatible hardware microcontrollers that can be easily reprogrammed and used in electronics projects or as interfaces between a computer and components such as sensors. A free online Arduino code editor like P5.js' is also available at arduino.cc.

✦ TEACHING AND EVALUATION METHODS The CUP included a section dedicated to the teaching methodology, which noted students ought to be encouraged to *"explore new contents and concepts and incentivize autonomous problem solving and development of digital artifacts."* Theoretical-practical sessions were meant to present the contents of the syllabus through lecture demonstrations, digital artifact presentations, and practical exercises, as well as to support and supervise students' work.

The section concerning the evaluation methodology reiterated the expectation of students' autonomy, though assigning importance to the continuous supervision of their work. Attendance, participation, and work in the classroom weighed 30% of students' final grades, and their group project outcomes contributed to the remaining 70% of their final grade. Such projects were expected to be developed iteratively, as students were required to submit and present their work at certain milestones.

✦ BIBLIOGRAPHY The unit plan also included TOUCH's recommended bibliography:

- Banzi, Massimo & Shiloh, Michael (2014). *Getting Started with Arduino, 3rd ed.* Maker.
- Igoe, Tom (2011). *Making Things Talk: using sensors, networks, and Arduino*. Maker.
- Noble, Joshua (2009). *Programming Interactivity*. O'Reilly.
- Shiffman, Daniel (2015). *Learning Processing, 2nd edition*. Morgan Kaufman.

It is worth noting all recommendations are English language books. The bibliography is not very distinctive. Compared to the bibliographies found in the curricular unit plans we analyzed in 4.2, only Banzi's introduction to Arduino was unique. Noble's handbook, covering Processing, Arduino, and the C programming language, was mentioned in nine other CUPs, and Shiffman's Processing manual was mentioned in ten others. Though Processing was not part of TOUCH's syllabus, the book may still be useful as some of the techniques and code listings can be easily adapted to P5.js; nevertheless, it is worth noting the absence of books dedicated to P5.js.

✦ THE PROFESSOR A single docent had the sole responsibility for TOUCH. To comply with the studies' anonymity requirements, we shall refer to him by the pseudonym 'Ivan'. He explained he had his first contact with computer programming while at secondary school, at which point he discovered an interest in the visual aesthetics of his programs. Ivan told us he was a ROSE graduate, who while studying Design, became fascinated with real-time graphics and discovered Processing, which he had been using ever since. In the years since, Ivan also explored and became proficient with JavaScript and P5.js, OpenFrameworks, and C for Arduino.

Professor Ivan had been teaching at higher education institutions for eight years, after working as a vocational training instructor for a similar number of years. He explained that at the time, he was teaching both at ROSE and at a private university in another city. He had a full teaching schedule, being responsible for a total of eight curricular units during that academic year—four at each institution—, including units whose students were split into additional classes. He added all his teaching assignments were related to computer programming.

The professor also explained that he was working toward obtaining his doctorate, and that he had been finding research hard to articulate with his teaching and personal life: in his words, he *"had been sleeping a lot less."*

✦ THE STUDENTS Though over eighty students enroll at the AMP course each year, by the time students reach their third year, they are divided between the Design and the Multimedia branches. By their final semester, students are further divided between those choosing between an internship and those choosing to stay at school to develop a final project. TOUCH is a mandatory unit only for the latter group; therefore, only twenty students—eleven female and nine male—were enrolled in the curricular unit.

✦ LEARNING MANAGEMENT Ivan told us he preferred to communicate with students via email and did not use a learning management system (e.g., Moodle) to support his teaching. Still, the professor gave students the web addresses for his profiles at the P5.js and Arduino online editors, through which he shared examples and some of the code he demonstrated in class.

## 6.3 *Session observation*

Though we contacted the docent responsible for TOUCH months in advance, we faced delays in obtaining the necessary authorizations from the school administration. Moreover, we then faced delays in obtaining the docent's agreement to a visit schedule. Therefore, by the time of our first visit during the sixth week of the semester, we had missed the first weeks' content exposition, and sessions had settled in a workshop format to support students' development of their final projects. We faced an unfavorable calendar and were unable to schedule additional visits to regular sessions. As we will report in §6.4, we were only able to visit ROSE again to attend a public exhibition of students' ongoing work and the final session dedicated to their projects' presentation.

### 6.3.1 CLASSROOM CONDITIONS

Sessions took place in a studio which we estimated as 10 meters long and about 5 meters wide. Based on our rough measurements and sketches, we made the renderings which we present in Figures 7.6.1 and 7.6.2. Two workbenches were placed along the long sides of the room and held several dated desktop PCs, which sat unused during our observations. Eight other tables and drawing boards of differing sizes and heights were placed in the middle in a 'U' configuration. Available seats exceeded the needs of the class. As shown in Figure 7.6.1, most students preferred to sit at the center tables, where they could face each other and had more space for placing their laptops. Yet, others preferred to sit on their own, with their backs turned on the rest of the classroom, as shown in Figure 7.6.2.

***Figure 7.6.1.*** *A view of the classroom environment, as seen from its doorway.*

***Figure 7.6.2.*** *Some students sat with their backs to the rest of the classroom.*

There was a desk available for the professor, equipped with another desktop computer and monitor, facing the students' work area. The room was equipped with a projector, and its projection screen hung from the short wall behind the professor's desk; when unrolled, the screen covered a whiteboard also bolted to this. Still, the room was also equipped with a small, wheeled whiteboard, which was propped against a corner.

Even though the projection size was large, text and code at regular font sizes were hard to read from across the room. As seen in Figure 7.6.1, there were large windows and skylights across one of the sides of the room. As some of the shades were damaged, the projection also lacked good enough contrast during the early part of the afternoon. Nevertheless, we also felt the studio's acoustics was adequate.

### 6.3.2 PROJECT SUPERVISION SESSION

Students had been required to build one group project throughout the semester, for which they had received precise instructions: in groups of up to three, students were to use P5.js to create an interactive animation articulating dynamic typography with three-dimensional graphics. Groups were to develop their projects iteratively, meeting three milestones. In the first, they were to implement a 'resting' non-interactive animation. For the second, they were to implement mouse or keyboard interactions, allowing users to change animation parameters; projects at this stage were to be presented at a public exhibition promoted by the school. For the final milestone, to be formally presented in the last session, groups were to build and program a custom Arduino-based interaction device and to implement the necessary interface in their animation's code.

⁂

During the sixth week, we observed a session fully dedicated to the supervision of students' ongoing project development. Seventeen students attended, divided into eight groups. Professor Ivan began by presenting code that concluded an explanation pending from the previous session, noting he had shared it on his P5.js profile page. Shortly after, the professor began walking around the classroom and talking to every group, who were expected to submit their first project milestone in the following week. We witnessed 15 separate discussions between Ivan and the students concerning the eight projects under development. The professor was able to dedicate over twenty minutes to each group during his first tour of the studio. After taking a break, during which some students left, the professor was able to check once more on the groups still present.

Six groups showed prototypes. The professor was afraid most were disregarding his instructions as four such prototypes lacked a 'resting' animation, the first milestone's objective. Moreover, two of these groups were 'ahead of themselves' experimenting with keyboard and mouse interactions, even though the second milestone was still more than one month away. Yet, some groups were facing further difficulties and yet to begin. One group produced a small video animation prototyping their idea, and another group showed their drawings; both asked Ivan whether they ideas were possible to implement and how to start. Another pair vehemently disagreed on their project idea, and Ivan had to mediate and suggest a compromise.

We also witnessed situations in which students posed direct questions about how to implement a specific effect or animation (3), asked for help with coding difficulties they were having (4), or with importing 3D models for use in their programs (2). One particular group, comprised of three female students, expressed their lack of motivation; while the professor used one of their laptops to correct the group's code, we understood the group lacked programming foundations (e.g., awareness of closing P5.js statements with a semicolon), as all three struggled to reply to Ivan's basic questions until a nearby colleague mouthed the correct answers.

Indeed, we witnessed five situations in which the professor asked to use a students' computer. In one instance, the student resisted and asked Ivan just to guide him; in two other such situations, the professor spent some time writing code and implementing part of the group's idea. Professor Ivan also made use of other means to answer students' questions, such as drawing diagrams on paper or on the whiteboard (3); during one of the latter situations, he asked for the whole class' attention as he thought his explanation useful for all. Ivan also gave theoretical verbal explanations (3), in one instance using his hands and plastic bottle caps to aid his discussion of distance measurement in 3D space. In another situation, he asked a student to open the relevant P5.js documentation webpage so they could read the solution together. Ivan also discussed the aesthetics of students' prototypes and ideas (6), advised refinements to groups already experimenting with interaction (4), and advised groups to lower their project's complexity (2).

One group had asked for an extended submission deadline. Though the professor did not grant it immediately, Ivan addressed all students at the end of the session to let them know he would grant them an additional week to submit the first milestone, as he felt many groups were not ready.

⁂

Students displayed good autonomy throughout the session. Most worked diligently on their projects while the professor was busy attending to others. Any time we checked, more than half the students in the studio were using the P5.js online editor, and a couple more were using illustration or 3D modeling software for tasks related to their project. Students were free to take a break and go outside, as the professor did not set a time for an intermission; nevertheless, such breaks tended to be brief. Moreover, any distractions we witnessed (e.g., mobile phone, watching unrelated videos on YouTube) also tended to be momentary breaks during the four-hour session.

Still, we saw a considerable number of students had difficulties programming. Ivan used the computers of some students, possibly those expressing less self-confidence (it is telling that one student, more confident, resisted it); in two instances, he wrote a substantial part of the solution. Though that may solve the group's immediate problems, it may negatively impact their learning. It is also possible the precise project instructions had an adverse effect on some students' motivation. On the one hand, instructions provide a shared space for problem-solving; on the other, they might go against students' wishes for their final semester, a time when many are thinking about putting together a personal portfolio. Hence, several groups departed from the instructions and explored interaction ahead of time, as the subject possibly held greater interest.

## 6.4 *Students' project exhibition and presentation*

As mentioned, TOUCH students were to iterate on their group projects according to the milestones set by the professor. After developing a default 'resting state' animation, they were to add mouse or keyboard interaction as to present their work at a public exhibition promoted by ROSE during the school's academic week. Later, students were to build a custom Arduino controller and modify their animation program in order to interface with it; complete projects were to be presented in TOUCH's final session.

### 6.4.1 PUBLIC EXHIBITION

A month before the end of the semester, ROSE organized a public showcase of students' work during the polytechnic institute's Academic Week. Students from all the school's courses had been asked by their respective professors and course coordinators to submit their practical work. Within some curricular units, such as TOUCH, participation was an evaluation requirement. The researcher visited the exhibition, which took multiple public halls and corridors in ROSE's main building, as well as to attend, in a private capacity, a few of the accompanying talks by digital art and design practitioners.

We were able to see the projects under development at TOUCH as they stood after students submitted the second milestone. We met Professor Ivan in the exhibition hall to gather his views. The school had only allocated a single computer to TOUCH's exhibit, which shared the hall with posters for the film and the product branding projects of students from other courses. Ivan explained he had asked for additional means, but lacking those, he wrote a small script that rotated through students' eight submissions every thirty seconds.

The professor complained most groups had not understood or care to comply with the requirements; only two of the works presented featured a 'resting' default animation (as he had required earlier), though all implemented mouse or keyboard interaction. The professor believed some of the groups had implemented interactions (e.g., using onscreen buttons) that he envisioned to be difficult to adapt to a custom controller for the final presentation. Ivan expressed frustration at the lack of progress several projects had seen since we had visited his classroom a month prior.

As he led only two sessions during the intervening weeks, due to the semester's unfavorable calendar, the professor speculated that some groups were not working on their project outside of the classroom. Ivan explained he had given students a lecture introducing the subject of tangible computing and explaining how to set up and upload programs to an Arduino controller, yet he added that the next session would only take place two weeks hence due to a holiday, and thus there were only two sessions remaining before the final presentations were due.

### 6.4.2 PROJECT PRESENTATION

We were invited to visit ROSE once again for the last session of the semester, in which we had the opportunity to witness students' final presentation of their projects. As groups were to demonstrate their custom Arduino-based controllers interacting with

their P5.js animations, there was an amount of setting-up between each presentation, as students carried all their hardware to connect their own computers to the classroom's projector. As the school did not loan the Arduino kits (though we understood one group had purchased their own), most students spent the first forty-five minutes of the session reconnecting and testing their electronics. Students that were ready and waiting were taking Polaroid photos and making party preparations to celebrate their graduation. Most groups showed autonomy during their preparations, and the professor found time for conversation with the researcher. Ivan told us he had scheduled one additional three-hour session that took place a couple of days before in order to help students complete their projects. He also explained he required groups to submit their projects' source code alongside images documenting their electronics, their finalized input devices, and captures of their running programs. Ivan did not issue presentation requirements: when one student asked him whether his group should have prepared a slide deck, Ivan replied that students near graduation should know how to present their work best.

Presentations followed in the order that groups volunteered; Professor Ivan took a seat at one of the central tables. We witnessed seven presentations by pairs (3) or groups of three students (4). Though there were nine female and nine male elements involved, all groups but one pair were segregated by gender. No groups employed slides or showed additional materials; all opted to demonstrate their programs and interaction devices with no further ado. Though two groups had merely built a control device consisting of a push-button (employed to replace the mouse button functionality) and were criticized for it, the others built more complex devices to control aspects of their animations: one housed a joystick in the surface of a wooden box, another housed a photoresistor in a painted cardboard box (sensitizing the size of the shapes on the screen to proximity to light), and three other groups placed rotating knobs in boxes of varied designs. During their presentations, five groups walked through their code to address the modifications they implemented since the school's exhibition. Three groups in particular projected confidence and were likely to have rehearsed their presentations, though only a single student spoke in one such presentation.

We understood the class was familiar with all projects; about half the students were distracted during their colleague's presentations. Except for one situation in which he asked a group to explain part of their code, which the students did with confidence, Ivan did not ask questions. Instead, the professor presented his comments at the end of each presentation. Groups were praised for the interaction processes they devised (3), for their ability to solve programming problems (2), for their attention to detail (1), for their project's completeness (1), or for the motivation demonstrated during development (1). Ivan also admitted to one group that they overcame his reservations about their work. Conversely, groups were criticized for presenting a project that had seen little progress since the exhibition (3) or for devising a simplistic interaction (2).

While waiting for the last group to set up their presentation, the professor spoke to a student who had arrived in the meantime and who told Ivan she would not present her project's final milestone and would rather take the appeal exam.* The student blamed her group partner for giving up and deciding to repeat the curricular unit the

* Failing students could request an appeal exam as long as their grade was at least 7.5.

following year; the professor noted her partner had made that decision early and asked the student to take responsibility for her own lack of dedication, noting she was always late to class and did not seek to join another group.

At the end of the session, Professor Ivan asked the students to pay attention to their email inboxes, in case he needed to request anything to grade their projects. He added that, should anyone need to take the exam, it would consist of a Processing and Arduino practical exercise to be completed in three hours. Ivan then asked the students about their plans and recommended ROSE's master's degrees. Ivan then dismissed the class, wishing students a good summer holiday.

⁂

We were disappointed with the quality of most projects. We had believed the four-hour sessions would provide ample time for development, despite the unfavorable calendar. Moreover, TOUCH was part of the final semester of the AMP licentiate, and we had expected students to bring a prior familiarity with programming from the preceding semester. In our view, though some projects had a certain complexity, most lacked a strong idea and attention to aesthetics. As the professor alluded after the exhibition, it may have been the case that most students did not work on projects outside of the sessions. That being true, one may speculate that the precise project requirements, at this late stage in the course, may have divorced students from TOUCH and led them to commit more of their attention to the main project led at another curricular unit.

## 6.5 *Students' acceptance and use of computer programming*

Professor Ivan asked students to complete our questionnaire, described in **5**.4, during our first visit in the sixth week of the semester. Given the delay and that the students were expected to have a prior familiarity with programming, we opted to conduct a single application of the survey, as explained in **5**.5.3. Seventeen students participated (nine female and eight male). Five were aged 20 or younger, eleven were aged between 21 and 24, and a single participant was aged between 25 and 34. One participant reported up to one hour of daily computer use, another one reported between one and two hours, four participants reported between two and four hours, eight reported between four and eight hours, and the remaining three participants reported over eight hours of daily PC use. Two participants reported they had failed TOUCH previously and were repeating the curricular unit.

We were surprised that only ten participants reported a prior familiarity with computer programming, as all students had previously been enrolled in a first-semester curricular unit, also led by Ivan, where they were taught an introduction to computer programming, specifically using Processing. It is possible the units' reliance on group projects let some students do without paying much attention to programming. Indeed, four participants reported having no knowledge of programming, and the remaining three claimed they were learning the subject for the first time. Of the ten who reported familiarity, eight participants described their knowledge of programming as 'basic'; one described his knowledge as 'reasonable,' and another claimed he was highly experienced.

Besides their current licentiate, two participants mentioned having learned to code at a vocational school, one mentioned learning at a different higher education course, and another mentioned online courses. Few participants answered our question about their familiarity with specific programming languages: three mentioned JavaScript, and two mentioned P5.js; it is interesting to note no participants mentioned Processing.

✦ ACCEPTANCE AND BEHAVIOR We computed means and standard deviations for the constructs according to the results of our tentative reliability tests (see 5.5.3). Given the few participants and their varied computer programming familiarities, there is little that may safely be concluded. We compared the responses of those who claimed prior knowledge of computer programming against the responses of like participants in our preliminary survey (see 6.2). As shown in Table 7.6.1, we found a worse acceptance of programming among such TOUCH students in comparison to like participants (of equal knowledge and seniority) in the preliminary survey. As before, we transposed the 7-point Likert scales to run from -3 (standing for full disagreement) to +3 (full agreement) as to ease readability. Positive values reflect favorable opinions, except for *Anxiety*.

***Table 7.6.1.*** *Programming acceptance among TOUCH students familiar with the subject (week 6) and preliminary survey participants attending the third or a later year.*

| | TOUCH ($n$=10) | | *3rd+ year students* ($n$=70) | |
|---|---|---|---|---|
| | Mean | St. dev. | Mean | St. dev. |
| *Performance Expectancy* | -0.25 | 1.51 | 0.97 | 1.47 |
| *Effort Expectancy* | -1.24 | 1.04 | -0.07 | 1.41 |
| *Attitude Toward Use* | -0.55 | 1.60 | 0.78 | 1.69 |
| *Subjective Norm* | -1.35 | 1.43 | 0.09 | 1.70 |
| *Self-Efficacy* | 0.13 | 1.01 | 0.63 | 1.24 |
| *Anxiety* (reverse scale) | -0.27 | 1.26 | -0.46 | 1.66 |
| *Behavioral Intention* | -2.03 | 0.90 | -0.04 | 2.16 |

The table tells us TOUCH students were generally more pessimistic about programming than other students in their situation, but starkly so in their *Behavioral Intention* to use computer programming in the future—in their case, after concluding their course.

⁂

We tentatively looked for relationships among the distinct acceptance, intention, and use measurements of students familiar with programming ($n$=10). We found a persuasive Kendall Tau correlation between their perceived voluntariness of programming in the curricular unit and a better *Attitude* ($\tau_b$=.68; $p$=.010). Yet, we could only find suggestive Pearson correlations between, on the one hand, these participants' *Behavioral Intention* and, on the other, *Effort Expectancy* ($r$=.67; $p$=.036) and *Attitude* ($r$=.69; $p$=.027). We found no significant correlations between participants' intention and their frequency or intensity of computer programming use—a contradiction of the core assumptions in the UTAUT model we may nonetheless ascribe to the small number of participants.

## 6.6 *Interview with the professor*

At the end of the project presentation session, the researcher requested to schedule an interview with Professor Ivan, who said he would grant the interview but would prefer to do it immediately, as he had about forty-five minutes available. Hence, the interview was carried out at that time, following the procedure laid out in 5.5.1.4.

✦ LEARNING TO CODE AND DIGITAL LITERACY Professor Ivan believed computer programming is an important skill for design practitioners and that the subject would increasingly figure within Design courses' curricula. He noted the growing demand for generative and visualization design, as well as for customizing design software applications—citing Python scripting within Blender for procedural 3D modeling. The professor explained he taught workshops in which students *"built small software applications for doing things for which there was nothing else in the market."* He added:

> Students understand, *"with this, I can set myself apart from all the others who just work with Adobe or some other brand's products."* And from that moment, they also dedicate themselves more because they think, *"I can create my own visual language and make my own tools rather than being limited."*

The professor noted that his students' work gained increasing visibility over the last few years due to social media and their public exhibition. Ivan told us he is often contacted by outside students asking for his advice in learning to code.

Ivan told us programming has a place within a broader digital literacy. Though not everyone has to know how to code, he told us, all would benefit from *"understanding the thinking underlying a condition, a repetition, a variable"* as to *"unlock the potential of existing tools,"* citing the examples of Excel and email filters.

✦ THE WORKING CONDITIONS Professor Ivan told us the working conditions had been good. A few years before, he explained, students would have to purchase their own Arduino kits, yet in recent years the school had enough kits for the students to use. Ivan added that the ROSE was in the process of acquiring Virtual Reality equipment to be employed in the following academic year's projects.

The professor complained about the academic calendar for the second semester, explaining it had been 'bad luck' that his session plan had been compromised by the successive breaks for holidays and school events. Though he had been able to schedule one remedial session, he believed students' motivation had been affected by the loss of regular contact with the curricular unit.

✦ THE STUDENTS Professor Ivan believed his students' interest in learning to code has been stable over the years. He believed his students at TOUCH and at the preceding unit were aware of the benefits of learning to code (edited for clarity):

> They are aware that the job market requires those skills. However, I suspect not all are willing to invest time to overcome that barrier [of mastering computer programming], and those try to get into the market in some other way. But I have no doubt whatsoever they are aware of the potential [...].

The professor also believed students were increasingly autonomous, emphasizing the growing availability and importance of online resources. He specifically noted Daniel Shiffman's videos: *"even if you don't tell students, sooner or later, you'll see someone in class watching a Shiffman tutorial on YouTube."*

We also asked the professor whether he noticed students' anxiety toward learning to code. Ivan noted he read our survey questionnaire and wished to state that he never heard a student claim they were anxious about computer programming because "*they were afraid of making a damaging mistake*." Nonetheless, he agreed students' anxiety was a real phenomenon, though it improves throughout the semester:

> I always feel their fear at the beginning of the semester. They always say things like, *"Hey, professor, programming is very difficult!"* In both schools I teach.

When asked whether he noticed differences in students' attitudes related to their age or gender, the professor told us, *"older students have always paid better attention and dedication in class."* Ivan explained he could make no such inference regarding students' gender: *"sometimes male students show more interest, but for instance, the best students this semester were two women."*

✦ STUDENTS' PROJECTS AND EDUCATIONAL OUTCOMES The professor told us that leading students to overcome their initial difficulties was a reason for asking for their participation in a public exhibition, as *"students get enthusiastic when their projects have visibility."* Ivan noted almost all students presented their projects at the end and that by that time, he had long ceased to receive complaints about how programming is difficult.

The professor compared his students' dedication to programming unfavorably to their dedication to video work, noting he and his colleagues *"noticed students love video"* and commit to it. He believed many students are unwilling to meet the challenges of understanding computational thinking or of overcoming their own habituation to graphical user interfaces. While *"all projects are good"* in curricular units concerned with video, Ivan told us, students' programming work is more heterogenous: *"two or three students make excellent projects, and the rest are merely reasonable."*

Students demonstrated a good degree of autonomy in their work, Ivan told us. He told us he had expected more emails with questions throughout the semester. Ivan also noted that whenever he asked students *"questions about their code, you can see they didn't plagiarize, it's their actual work."*

Professor Ivan believed students met the overall objectives of the curricular unit, though conceding the negative impact of the calendar, as well as that his students could have presented greater dedication. Still, he believed the better students had shown a good ability to integrate computer programming with their design and aesthetic objectives. Ivan also ascribed some of the weaker projects to students' initial lack of motivation toward overcoming the challenges of programming.

## 6.7 *Successes and limitations faced in this case study*

As mentioned, we faced some delays at the beginning of our study. The school's services were slow to grant permission after asking for additional assurances of compliance with data protection regulations, after which we had some difficulties in finding a convenient date for our first visit in agreement with the curricular unit's docent, as we had some scheduling constraints due to conducting three other case studies at the same time. We thus summarize this case study's successes and limitations:

***Table 7.6.2.*** *Successes and limitations faced in the TOUCH case study.*

| | *Planned* | *Unplanned* |
|---|---|---|
| *Successes* | Collected most types of data sought.<br>Good participation percentage in students' survey. | Opportunity to attend a public exhibition of students' work.<br>Attended final project presentations. |
| *Limitations & Disappointments* | Significant early delays and scheduling conflicts; missed expositive lectures.<br>Somewhat limited number of students for quantitative survey; eschewed the second application due to delay.<br>The professor granted limited time for his interview. | - |

Due to the delays mentioned, we were only able to visit the school by the sixth week; we missed the unit's early expositive lectures and were only able to attend one practical studio session due to scheduling conflicts. This also affected the timing of our students' survey application. Still, given the small class size and the expected familiarity with computer programming, we saw no harm in eschewing the second survey application.

# 7 FLOW: VISUAL PROGRAMMING FOR MUSICAL COMPOSITION

The case which we shall refer to by the codeword FLOW concerns a rather specific situation, both in comparison to the other cases and to the larger universe of computer programming within arts and design licentiate courses offered by the Portuguese public system. FLOW introduces computer programming to second-year students studying musical composition at a performing arts school, which we shall refer to as AMBER. The case study is notable as it focuses on an exceedingly small course that admits no more than six students per year. We also find it notable that these students are taught to code using Max, a visual programming language—an entirely different paradigm in comparison with the text-based languages commonly taught.

We will present AMBER, its Music licentiate, and its atypical student admission process in the first subsection. In §7.2, we will turn our focus to FLOW, and will present its curricular unit plan, its students, and its professor. In §7.3, we will describe our observations from inside the classroom. Then, in §7.4, we will turn our attention to the unit's supporting website and the shared educational resources. In §7.5, we will present our findings from students' technology acceptance and use surveys conducted at the start and at the end of the semester. In addition, we conducted an interview with the unit's professor, which we will report in §7.6. We will conclude this section in §7.7 by addressing the successes and limitations we faced while conducting this case study.

## 7.1 *The course and its host institution*

AMBER is a small performing arts school that is a part of a large polytechnic institute. According to the latest information publicly available on the school's website, [*] related to 2016/2017, the school hosted several hundred students and employed around 150 academic staff. Over three-quarters of the students were enrolled at the school's licentiates, while the remaining attended independent master's courses. As in the other licentiates dedicated to Music offered at public higher education institutions, there were several variants of the course. Government records list a single licentiate (DGEEC, 2019), yet each variant features a wholly different curriculum. Therefore, for the purpose of this study, we will treat the Composition variant of the Music licentiate as a distinct course.

✦ THE COURSE The course is a three-year licentiate with a study plan solely comprised of semester-long units. After the implementation of the Bologna Process, the curricular plan of the Composition variant featured units explicitly dedicated to programming, which were offered from the first year of the course. This curriculum also included units concerning specialty subjects such as sound synthesis, which can leverage and strengthen students' knowledge of computer programming. Small revisions of the Composition curriculum during the past decade increased the relative weight of these technological units, which at the time of writing comprise about a tenth of the course's total ECTS credits.

[*] Reference omitted as to comply with participants' anonymity. We note that much data about AMBER and its courses was not found on official records (DGEEC, 2018, 2019).

We should note the curriculum had one distinctive characteristic: whereas most courses require their students to complete five or, at most, six curricular units per semester, the Composition study plan required students to attend and complete eight or nine units in each term. In addition, the school's academic calendar indicated longer than average terms, with 18-week periods of classes in each semester.*

✦ STUDENTS AND ACCESS Unlike in other arts and design courses to which these case studies relate, access to higher education Music courses does not merely require a secondary school diploma. Applicants are also required to pass a musical competency exam and to be approved in an audition before a jury at the school to where they are applying. In the case of the Composition variant of the Music course, the audition process involves an interview and an analysis of the applicants' portfolio. Students admitted to the Music courses often came from conservatory offering complete secondary schooling, and the remaining would have had attended a music academy while completing their basic and secondary education at a 'regular' school.

Some information about AMBER's courses was missing from official datasets (DGEEC 2019, 2018), and we are unable to present detailed numbers about the students attending the Music Composition licentiate. According to the latest report available on the school's website, there were over four hundred students attending all variants of the Music licentiate in 2016/2017. Still, according to the 2018/2019 application regulations published on the same website, Composition students are a small minority, as there were only six places available for students wishing to enroll in that variant.

## 7.2 *The FLOW curricular unit*

The curricular unit, which we refer to by the codeword FLOW, is a mandatory part of the second semester of the first year of the curriculum of the Composition variant of the Music curriculum. As mentioned, FLOW offers students an introduction to computer programming that is focused on the needs of music composers. The unit was classified as theoretical-practical, and weekly face-to-face sessions lasted two hours.

✦ OBJECTIVES AND SYLLABUS According to the publicly available curricular unit plan (CUP) at AMBER's website,† students attending FLOW were expected to *"acquire essential knowledge about algorithms in the context of musical programming"* and thus proceed to *"develop Max/MSP programming skills,"* to *"develop skills for creating simple interactive musical systems,"* and finally to *"use the Max/MSP programming environment to create a musical piece."* Though the unit plan did not include a syllabus section, the professor responsible for the curricular unit built a website (which we will analyze in §7.4) to support his lectures, where he posted his syllabus for the curricular unit. There were three major topics in this syllabus, which we reproduce:

* Academic calendars with 15 weeks of classes per semester are the most common among Portuguese higher education institutions.

† Reference omitted as to comply with the participants' request for anonymity.

- Fundamental computing principles for musical programming—including discussions on the purpose of computers, algorithmics and programming principles, and programming languages.
- Musical programming with Max/MSP—including an explanation of visual object-oriented programming and the Max environment, and subtopics dedicated to MIDI, audio manipulation, and sound synthesis.
- Composition using Max/MSP programming—including the audition of historical examples and the practical application of programming to electroacoustic composition.

— *from the FLOW professor's website, translated and abridged*

The unit was focused on teaching programming with Max, which students would be required to use in composing their projects. In our panorama of computer programming within Portuguese public arts and design licentiates (see 4), Max was mentioned in nine other CUPs from six other courses; not all related to Music. As presented in more detail in 3.3.2.4, Max is a visual programming language, which requires its users to place and connect the inputs and outputs of visual objects rather than type text statements; hence, Max source code is often called a *patch*. It is worth reminding Max is commercial, closed-source software, though a free evaluation version is available.

✦ TEACHING AND EVALUATION METHOD The professor's support website also mentioned the evaluation methodology, which had been left out of the CUP. The evaluation methodology was distinctive: the professor was to be responsible for 70% of each student's final grade, according to his appreciation of the student's attendance, participation, and work presented in sessions. The remaining 30% of each student's final grade was to be decided by a jury of other educators, following a public performance of the musical piece the student was required to write throughout the semester.

✦ BIBLIOGRAPHY The unit plan included a list with FLOW's recommended readings, which we reproduce here:

- Cipriani, Alessandro & Giri, Maurizio (2010). *Electronic Music and Sound Design: theory and practice with Max/MSP*. Contemponet.
- Pope, Stephen (ed.) (1991). *The Well-Tempered Object: musical applications of object-oriented software technology*. MIT Press.
- Roads, Curtis (1996). *The Computer Music Tutorial*. MIT Press.
- Winkler, Todd (1998). *Composing Interactive Music: techniques and ideas using Max*. MIT Press.

Though three of the books had been published over twenty years before, even though they concern practical use of computers, that may attest to the long-standing presence of Max in the computer music ecosystem. We also note there were no books written in Portuguese among these recommendations. The bibliography was not very distinctive: in our analysis of relevant CUPs within public arts and design undergraduate curricula presented in 4.2, Cipriani & Giri's, Pope's, and Winkler's books were mentioned in the bibliographies of two other units of AMBER's Music course, one of which was also the responsibility of the same docent as FLOW. Roads' possibly mislabeled 'tutorial'—for it is a detailed panorama of techniques—was found among the recommendations of two curricular units taught outside of AMBER.

The docent responsible for the curricular unit also included an independent set of recommended readings in his support website, though relating to all of his teaching assignments at AMBER without discriminating individual curricular units.

✦ THE PROFESSOR The responsible professor offered us some key details about his own professional background. As to ensure our compliance with the studies' anonymity requirements, we shall refer to the professor chairing FLOW by the pseudonym 'Joel.' Professor Joel is an accomplished composer, and both his artistic practice and academic research focus on the intersection between music and technology. Joel told us he had been teaching for nearly twenty years and that, at the time of our study, he had been working in higher education for the last thirteen years. The professor also explained that he had been working at other institutions for the last several years and that we met him in his first year at AMBER, where he now taught exclusively. Joel also explained he was assigned to teach six curricular units during that year, all related to electroacoustic music; yet only FLOW and another unit taught to second-year students were explicitly concerned with computer programming (though he added students might use computer programming in conceiving their electroacoustic compositions for the other units). In addition to those six curricular units, Joel remarked he also gave some of the individual composition lessons all students are required to attend throughout the course.

Professor Joel told us that because of his father's occupation, there were computers in the house as he grew up, and he had taken programming as a hobby at an early age. He explained that when he was studying Music, he and his colleagues were introduced to Max, and he felt his early computing hobby had given him a greater and easier understanding of the tool. Though mentioning he *"hates Lisp,"* Joel told us he also uses Processing, C, C++, OpenFrameworks, Java, and Python.

✦ THE STUDENTS There were only six students attending FLOW, including an Erasmus mobility student from Slovakia. Including the mobility student, only two were female. All local students attending the unit completed our surveys, which we will report in detail in the following subsection. Hence, we can advance one of the male students was over 25 and admitted to being attending FLOW for a second time, while all others were under 20 and newly enrolled in the Music licentiate.

## 7.3 *Session observation*

We had no difficulties obtaining the necessary permissions to conduct the FLOW case study and to arrange our visits to attend some of the sessions. The weekday in which the sessions took place was free from scheduling conflicts with the other studies we carried out simultaneously throughout the second semester of the 2018/2019 academic year. In addition, the academic terms at AMBER were longer than at the other institutions. Still, there were periods with two or three-week breaks between consecutive sessions; three planned sessions were suppressed due to holidays and a field trip to attend a colloquium at another institution. We visited the school six times in order to observe sessions of the curricular unit, as summarized in Table 7.7.1:

***Table 7.7.1.*** *Summary of the observed sessions of the FLOW curricular unit.*

| *Week* | *Attendance* | *Session summary* |
|---|---|---|
| 1 | 5 students | Historical introduction to visual object-oriented programming;<br>Basic operation of the Max environment. |
| 6 | 6 students<br>(inc. Erasmus) | Generative music using stochastic processes. |
| 10 | 6 students<br>(inc. Erasmus) | *Project idea discussion.* |
| 13 | 6 students<br>(inc. Erasmus) | *Topics suggested by students:*<br>User interface elements; Automated recording and looping playback. |
| 18 | 4 students | *Project discussion:*<br>Last minute questions before students' final submission. |

According to the lesson plan, during the first nine weeks, the professor gave students a general introduction to programming in Max. After that, students were to discuss their ideas for their composition project, which was initially intended for performance at a live recital at the end of the semester. From that discussion, the professor focused on programming topics suited to meet the needs of the students' ongoing projects.

### 7.3.1 CLASSROOM CONDITIONS

Sessions took place in a small room, which we estimate not to exceed 15m² in area. Two rows of tables, with four seats each, faced a small desk available for the professor, with a whiteboard on the wall behind it. The room was also equipped with a set of surround speakers, an amplifier, and a digital piano close to the professor's desk. We present a rendering based on our rough sketches and measurements in Figure 7.7.1:

***Figure 7.7.1.*** *A general view of the classroom. Note the close distance between the professor and the students. The professor had a small digital piano (lower right corner) and a television set (upper left) at his disposal.*

The room was possibly assigned to other uses, as the furniture and equipment pertaining to the class was placed in the front half of the room, near the doorway. In the back of the room, there was another desk with a computer, a rack of audio equipment, and two bookcases. Instead of a projector, the room had a flat-screen television set attached to an articulated arm bolted to a side wall. The professor used this screen to mirror his own laptop's during his Max programming demonstrations. We felt the screen was small and lacked the necessary clarity to display all but the simplest Max patches. In addition, the room may have had insufficient insulation, as outside noises were often heard—indeed, the professor complained about the acoustics during his interview (see §7.6).

No computers were available to students, who had to bring their own devices. We note Max is commercial software. Though students could use it outside of class in the school's computer room, the professor asked students to purchase an academic license, as saving 'patches' (i.e., programs) is deactivated in the free trial version.

### 7.3.2 THE LECTURES

As summarized in Table 7.7.1, three of the sessions we attended concerned programming lessons, while the other two (to be reported in §7.3.3) were dedicated to the discussion of students' ongoing projects. We thus summarize the lectures we witnessed:

- INTRODUCTION TO MAX, WEEK 1 The lecture was attended by all five local students enrolled in the unit. Professor Joel knew the class from a first-semester curricular unit (unrelated to programming), so he dove straight into explaining the lesson plan and how students were required to devise an individual musical composition using Max, intended for performance at a public recital at the end of the semester. Joel explained he was going to invite professional composers to take part in the jury and to have some of their music performed at the recital.*

  Professor Joel began his lesson with a theoretical and historical introduction to Max, which he described as a visual environment that is also an object-oriented programming language. The professor likened Max programming to operating a modular analog synthesizer, with which the students would be familiar, and drew a comparison to imperative programming, which he likened to the operation of a cash register at a supermarket. The professor also discussed Max's pricing and the availability of student licenses and of the limited trial. Joel taught at a relaxed pace and in conversation with students, who were attentive and posed questions.

  Sitting down before his computer, Joel opened Max and began a simple example program that added two numbers, from which he proceeded to demonstrate basic examples about triggering events, the execution order of patches, setting objects' initial parameters, and explaining how the inner code of built-in objects can be inspected. Throughout these demonstrations, we could witness that two students had Max installed on their own laptops and were reproducing the professor's examples. Another student was using their phone to record audio of the lecture (we were unaware whether that had been authorized).

* The recital was later cancelled, unfortunately. Students were asked to deliver recordings of their finished compositions instead, which we were not given the opportunity to listen.

Professor Joel proceeded to demonstrate the use of Max to generate MIDI* notes. He showed how to connect Max's output to a piano synthesizer software on his laptop so the result could be heard, testifying about the language's usefulness for musicians and composers. For the remainder of the session, the professor iterated on his example, generating random musical notes with a time delay, and then setting every fourth note to play louder than the others. As the patch grew in complexity, we saw that students began struggling to follow the example as the screen lacked enough visual clarity. The professor tried to address this by drawing diagrams on the whiteboard.

■ GENERATIVE MUSIC, WEEK 6 In addition to the five local students, the Slovak Erasmus attended, so Professor Joel addressed the class in English most of the time. He began his lesson by taking a few minutes to devise a Max program and then asking students whether they could explain what the program does. We felt students were not as engaged with the lecture this time, thus Joel proceeded: he explained the program was a stochastic note generator, which he demonstrated by connecting his laptop to the digital piano and striking a few keys, after which the program took over and started to generate similar-sounding notes. Joel asked to suggest an improvement; one of the students gave a suggestion. Still, from our vantage point, we could see no students were reproducing the professor's patch, even though four had Max on their laptops' screens. One student was not paying attention and was apparently working for another curricular unit, while another checked Facebook occasionally.

Joel continued to iterate on his program. At some point, while discussing computers' timing precision, he remarked on *"the stupidity of computers."* He also asked the class to name a composition of their liking, and after looking online for a MIDI file of the composition a student suggested, he demonstrated how to load the file into his program and ran it so the class could listen to how the first few notes of that theme were followed by similar-sounding generative music. The professor then dedicated a few minutes to talk about the work of the composer Karlheinz Essl,† who wrote generative music software, and showed video excerpts of Essl's live concerts. Joel also took this break from Max to make announcements concerning festivals and contests that may interest the class. The professor then showed one of Essl's software applications and challenged students to explain how a similar program would be devised in Max. As nobody volunteered an answer, Joel thus demonstrated it himself, spending twenty minutes devising a Max patch while posing rhetorical questions that called for little involvement from students. After successfully demonstrating his code interacting with the notes he played in the classroom's piano, the professor asked students for a small exercise as homework before dismissing the class.

---

* Musical Instrument Digital Interface, a set of communication standards for communicating musical notes and their respective parameters (e.g., pitch, velocity, stereo panning) between musical equipment and software.

† www.essl.at

Q&A, WEEK 13 This lecture was attended by the entire class, including the Erasmus student. Joel dedicated this lesson to aspects of Max programming that would be pertinent to students' ongoing work, and thus made himself available to demonstrate solutions to students' questions. Though local students posed their questions in Portuguese, Joel used English during his explanations of how to code Max patches that responded to particular students' problems. From our vantage point, we saw most students were uninterested when the professor's explanation did not concern their own question. As during the other sessions, programs often became too complex, and the classroom's television set did not offer any visual clarity, as exemplified in the following figure:

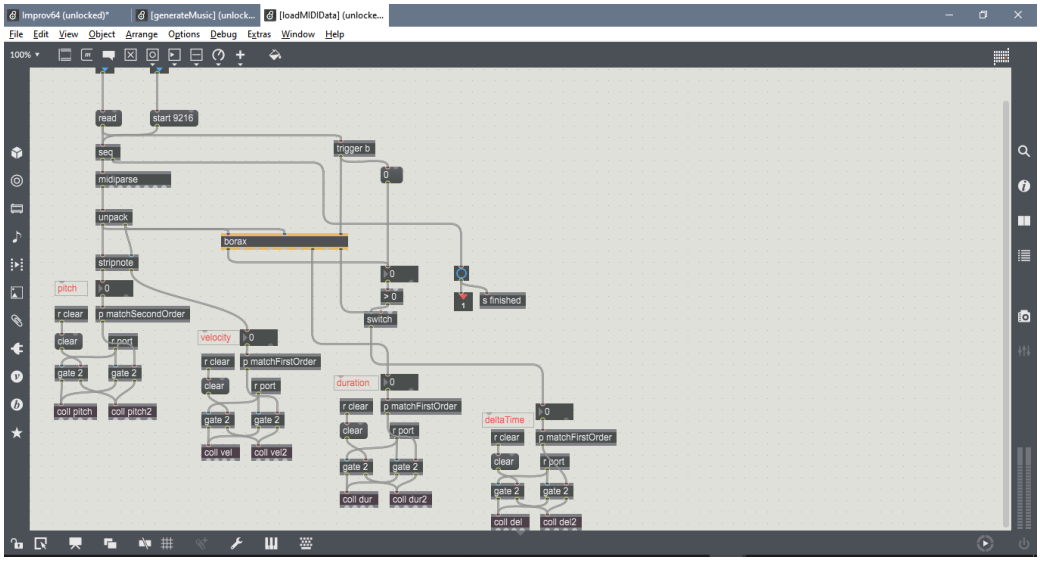



***Figure 7.7.2.*** *A view of H.M. Simmonds' Improv64 source code (github.com/HMSimmonds), similar to the professor's loop recorder example, captured in the researcher's computer at the same pixel resolution of the classroom's television set. The workspace is nearly unreadable at 100% zoom level, even in print.*

The lack of clarity often forced the professor to get up and point at the screen to clarify what he was doing. As Joel tended to iterate on his demonstrations, the class disengaged further: we saw one student writing an essay, possibly for another unit, another browsing the web, and two others whispering in conversation. Upon concluding each patch, Joel would offer to email a copy of the program to the student who had made the original question.

As to recapture the class's attention after answering questions, Joel showed students videos of performances by Dan Tepfer, a pianist who employs generative algorithms to assist in playing a robotic piano. The professor told students they ought to be able to create similar work; afterward, he asked the class how they would devise a patch that reproduced one of Tepfer's works. At this stage, Professor Joel assembled the program in Max while following the instructions of students, most of whom participated in the exercise. Joel ended his lecture with a conversation on the topic of machine learning, emphasizing how students ought to *"either do better or different"* in the face of the general availability of automated musical composition algorithms.

✦ PROFESSOR'S BEHAVIOR Professor Joel presented behaviors and strategies worth noting during the three lectures we witnessed:

FIRST, the professor sat down at his desk most of the time. Given the small class size, there was proximity with the students, which enabled a conversational tone. Joel often posed questions to individual students, not just about the topic at hand but also about their preferences. For the most part, Joel easily captured the class's attention.

SECOND, Joel eschewed the use of PowerPoint slides, favoring a spontaneous-seeming approach in which he transparently made web searches to present content related to his lecture, switching back and forth from Max. We felt the professor was quite experienced with his approach, as he talked calmly throughout such presentations.

THIRD, albeit favored by the small class, we note the professor's command of the lecture rhythm. Given that each session was only two hours long, Joel did not concede intermissions. Instead, every half-hour to forty minutes, the professor paused and offered digressions away from programming—e.g., presenting videos related to a session's topic. This gave the students time to rest and refocus before the professor resumed his coding demonstrations.

FOURTH, as the professor demonstrated his proficiency in Max, he was sometimes oblivious to the jumps in the complexity of his programming examples, proceeding at a pace dictated by his enthusiasm, which students sometimes failed to follow, moreover given the lack of clarity of the classroom's screen. Sometimes students asked clarification —at which point Joel got up and either pointed at the screen or drew diagrams in the whiteboard –, yet they were also as likely to disengage from the lesson.

FIFTH, Joel took care to motivate students and prove the salience of his lectures to their artistic practice. In multiple instances, often during the intervals, he employed to pace his lectures, the professor showed and demystified the work of professional composers, telling students had already experimented with similar techniques. He also asked students to consider and discuss the implications of what they were learning.

✦ STUDENTS' BEHAVIOR During our observations, we also noted a set of behaviors from the students attending the lectures which are worth summarizing:

FIRST, students happily conversed with Professor Joel during the first lecture, spoken in Portuguese, but were more reticent to talk with the professor during the other two sessions. Though we believe students understood these English-language lectures reasonably well, most tended to reply to the professor's direct questions in Portuguese.

SECOND, students were more likely to quit reproducing Joel's demonstrations and to display inattentive behaviors as the lectures advanced. We saw that students responded positively to the professor's bids to manage and refocus their attention, which suggests that students' attention capacity decreased with time, counter to the professor's programming demonstrations' increasing complexity.

THIRD, as a possible corollary to the aforementioned behaviors, though some pertinent questions were still asked, students were quieter and less likely to ask the professor for clarifications during the later lectures.

⁂

Though these students were highly motivated and well-equipped to understand the foundations of Max visual programming, we believe the inadequacies of demonstrating the use of a tool through a mirrored display of the professor's computer screen revealed themselves even earlier than in the case of text-based languages. Hence, students were sometimes unable to muster the required level of attention. Still, we ought to note the professor was aware of these limitations and placed constant emphasis on the students' need to commit to homework and autonomous study.

### 7.3.3 PROJECT SUPERVISION

We had visited AMBER during week 10 for a session dedicated to discussing students' project proposals. All students attended, and Professor Joel spoke mostly in English for the benefit of the Slovak student. As in other sessions, most local students preferred to talk in Portuguese. The professor thanked some students for sending materials in advance (e.g., sheet music) and emphasized how students were free to diverge from their initial proposals as they worked during the rest of the semester.

One student was looking to place generative music sequences within a written composition. The Erasmus student also proposed a generative music piece; yet, based on excerpts of preexisting music. Another student presented sheet music he had written for saxophone and synthesizer, noting he was interested in using Max to control the performance's spatialization. Two students were interested in exploring the same set of techniques, using Max to enable a performer to manipulate the rhythmic syncopation of looping sequences. The final student was looking to invite countertenors, who were spending the semester at AMBER through the Erasmus program, to participate in his performance; he was adamant in employing Max for the real-time manipulation of the voice audio, even though signal processing was not part of the syllabus.

Most colleagues paid attention during these exchanges. Though the professor did not delve into the finer details of implementation in Max at this stage, he brought depth into the discussion of musical ideas and techniques, which we were often unqualified to understand. Joel gave examples of relevant contemporary composers, and in two cases, referred to students the contacts of people he considered qualified to help them, such as another professor at the course or a doctoral student he knew. Joel remarked to the first student looking into generative music that one *"should not use technology for the sake of using technology"* and asked the Slovak student to pay attention to the copyright status of the musical excerpts she intended to use. The professor also had a long discussion with the student who was adamant about using audio manipulation, though the technique would not be addressed until a curricular unit in the following year. Joel suggested a compromise of having an automated piano respond to the performer's voice, which he considered easier to implement.

We believe the session demonstrated the virtue of ample tutoring time, as we felt students gained a much clearer idea of how to proceed. After thanking students for the discussion, Professor Joel asked students to identify their main technical challenge by the following session, adding that he would require them to submit a small program solving that challenge a couple of weeks after. The professor also dispensed some general advice (edited for clarity):

> If you develop an idea and try to program it afterward, you can find that hard to do. However, if you try to prototype your main problem, though you may not solve the problem, you will play around with it.

Just before dismissing the class, the professor addressed the specificity of programming for musical composition, recognizing that *"some problems are not entirely formalizable."*

⁂

We made one last visit to the FLOW classroom during the last week of the semester. At this stage, we had expected to witness a public recital of students' work; however, the professor had decided to cancel the recital and postpone the students' evaluation deadline for another two weeks. Students were to submit materials related to their composition (e.g., sheet music and Max patches) along with an audio recording of their work, which was to be judged by Joel and another professor at AMBER. We thus witnessed a final session dedicated to project supervision, attended by four students.

One of the students explained her work was nearly complete. The professor asked the student to connect her laptop to the classroom's sound system and play her music. While playing, Joel asked the student to mute tracks in order to understand which parts were generated by Max and which were audio tracks. The professor agreed the work was nearly complete, and discussed the musical aesthetics and suggested some refinements. Two other students were having technical difficulties with Max. One of these students posed a direct question, which Joel answered by quickly devising a Max patch on his computer, while the other student explained he had some problems with his program. After Joel invited this student to connect his computer to the classroom's speakers to demonstrate the issue, he used the students' laptop and attempted to solve the problem. Unsuccessful, the professor went back to his desk and began a new patch in his own computer. After several minutes, Joel managed to run a small program that created the desired effect, which he offered to share with the student.

Professor Joel spent almost half the session helping one particular student, who pleaded for a way to obtain a passing grade, frustrated he had *"lost the thread"* of the unit halfway through and that *"the train did not stop."* Joel emphasized the unit was of paramount importance as it precedes other programming units. Still, he motivated the student, noting he had completed his homework exercises and that the final project would not require programming any more complex; he asked the student to focus on his composition work. Seemingly relieved, the student asked for help with a technical issue, which Joel took some time to solve using the students' laptop.

Throughout these exchanges, other students were seen working, and their audio could be overheard (one used his laptop's speakers while another wore open-back headphones). Someone was practicing the flute in an adjoining room. At some point, the student talking to the professor asked his colleagues to lower their volume.

At the end of the session, Professor Joel gave his final remarks to the class. He admitted the conditions had not been ideal and that he expected the electroacoustic and programming units to move to a new classroom in the coming year. Joel also complained about how these curricular units are seen as secondary and advised students to begin treating programming as more relevant to their portfolios.

## 7.4 *Online learning educational resources*

Professor Joel shared with us the address of a static website he had put together to support his teaching. We captured a snapshot of the website as it stood at the end of the semester with the HTTrack 3.49 software, as to describe its organization. We note the site was not only dedicated to FLOW but to all of Joel's assignments at AMBER. Still, the website was rather simple and organized as follows:

- CURRICULAR UNIT PAGES, in which Joel put all content related to each curricular unit he taught. In the case of FLOW, a single page was divided into three sections: first, a summary of the curricular unit plan—objectives, syllabus, and evaluation methodology; second, an outline of the session calendar; third, a list of download links for the code examples (14 in total) the professor had produced in class.
- MUSIC / READINGS PAGES, in which the professor listed bibliographic and music recommendations related to all curricular units assigned to him. Joel included links with all references, ranging from full-text PDFs or integral audio recordings available online to excerpts or websites related to a book or musical piece.

We understood the professor favored email to communicate with his students and did not make use of any other learning management system.

## 7.5 *Students' acceptance and use of computer programming*

Despite the small number of students enrolled in Music Composition, such as that only six attended FLOW classes (including a non-Portuguese speaking mobility student), we still asked the professor to request students to complete a questionnaire identical to the instrument we employed in the other case studies (see **5**.4).

All five local students completed the first survey application. Four were aged 20 or younger—the remaining student, aged between 25 and 34, was also the only attending FLOW for a second time. There was a single female student among the participants. Concerning their daily use of computers, one participant reported between one and two hours, another reported between two and four hours, two participants reported between four and eight hours, and the remaining participant reported over eight hours of daily computer use. It is interesting to note that even though the unit offers an introduction to computer programming, only one participant wrote at the beginning of the semester that she would be learning to code for the first time. Three others described their knowledge as 'basic'; the remaining student described it as 'reasonable.' Two students mentioned having learned to code during secondary school; autonomous learning and online courses were mentioned twice. Asked about their familiarity with programming languages, Max was mentioned twice, and audio-specific languages Supercollider and SonicPi were each mentioned once, as were C#, Java, and Visual Basic.

As mentioned in **5**.5.3.1, the number of students in this case study prevented quantitative analyses; thus, we explored whether there would be qualitative shifts in students' acceptance and use of programming across the semester. Moreover, one of the participants that completed the first survey did not respond to the second questionnaire

at the end of the semester. Still, we matched the four sets of second survey responses to the respective participant's first survey. Though the four participants' opinions changed heterogeneously between the start and the end of the semester, we identified items in which participants unanimously demonstrated either a negative or a positive change in acceptance, or else there was a unanimous lack of difference.

Regarding whether *"interaction through a programing language would be clear and understandable"* (*Effort Expect. #1*), all four participants either maintained or evolved favorably in their opinion. Likewise, all four either maintained or held a more favorable opinion on whether they could complete a project *"if [they] had a lot of time to complete it"* (*Self-Efficacy #3*). The four participants also tended to either maintain or lower their agreement with all three items measuring their *Behavioral Intention*, which we found a possible reflex of the coming summer holiday, as the students were expected to attend a curricular unit about programming in the coming academic year.

## 7.6 *Interview with the professor*

Professor Joel granted the researcher a face-to-face interview, which followed the procedure laid out in 5.5.1.4. The interview took place two weeks after the last session and lasted around forty-five minutes.

✦ ARTS EDUCATION AND LEARNING TO CODE The professor told us *"the world does not know very well what artists are for; they think artists are people that make ornaments,"* and made a strong remark about *"the educational system's absurd normalizing impetus"* during the last several years, which *"annihilates students' individuality"* and greatly complicates the work of arts educators. He explained (edited for clarity):

> At a medicine or engineering school, you want to put a stamp on people: *"this person was Paul when he came here, now he is a computer engineer."* You want to shape people to correspond to a certain facticity. But in an arts school, you want the young music student to leave as a composer different from everyone else because you want to stimulate that singularity.

Professor Joel added, *"students nowadays look much more alike,"* and leveled the same criticism at higher education institutions' hiring and training of academic staff:

> Institutions do not want excellent professors with their individualities. They want a model professor repeated five thousand times. [...] To avoid having bad professors, they want average professors. But when you had bad professors, you could have excellent professors. And it is the same thing as with students.

The professor told us discussing how programming interrelates with digital literacy and digital competencies, though important, was not interesting to him. Joel would rather discuss how he sees programming as an artistic technique:

> It is clear to me [...] that it does not make sense for an artist to use a technical resource they do not master. For instance, I may hire someone who plays the violin better than me—which is not difficult—to play my composition for violin.

> But it does not make sense to write for violin if I lack a deep understanding of what is possible to do with a violin.

Professor Joel believed many of his colleagues disagree with his view: *"people think a visual artist can be ignorant about computers and about programming and hire someone to code what he wants,"* yet he found that illegitimate. Joel explained that *"if you want to work with digital art, then programming is THE discipline that teaches you how to master whatever media you want to employ"* (his emphasis). Still, he noted that digital art does not necessarily imply the use of digital media: "*I can make digital art just using thread and fabric, and [...] if I play Beethoven's Moonlight Sonata in a digital piano that does not make it digital art."*

✦ THE CURRICULUM AND THE UNIT PLAN Professor Joel explained that the Composition course aims to qualify students to use Max in their work, after a knowledge of "*what is possible to do with [programming], even if not all become proficient."* He told us that this latter objective is complete by the end of the first year.

Joel explained he had no say over curricular unit plans for the 2018/19 academic year, as he had just recently been hired by the school. He told us he intended to revise FLOW's plan according to his personal objective of emphasizing that *"programming can be useful to solve problems beyond those directly related to electroacoustic composition, [...] even to aid the composition of piano pieces."*

The curriculum includes a core Composition unit in each semester throughout the course, in which students are given individual lessons and are required to compose musical pieces to be performed in public and reviewed by a jury. Professor Joel told us units about electroacoustic music and programming, such as FLOW, have similar work requirements (i.e., writing a musical composition), yet students "*don't have the same kind of tutoring, and the units don't have the same kind of formal evaluation."* He believed *"this leads students to regard electroacoustic as a poor sibling, and that is a problem."* Joel explained he was intent on solving that problem in the coming years:

> The curricular unit plans and the thinking that devised them are still artifacts from a time in which there were very few students using electroacoustic means in their individual composition. [They] were the exception. I would say they are the rule nowadays.

Joel told us that *"it makes more sense for these lessons to focus deeper on [programming] technique rather than on composition."* He hoped the new curricular unit plans would be motivating, as students would *"feel the school had more realistic expectations from them."*

✦ THE WORKING CONDITIONS According to the professor, the classroom was uncomfortable, and its acoustics were inadequate. He *"never had good conditions to teach,"* adding that schools, for the most part, overlook the spatial conditions. We also asked the professor to comment on the use of Max, given that it is commercial software. He explained he favors open-source software and had pondered whether he should teach

Pure Data instead.* He nevertheless chose Max, as it is an industry standard *"made by composers for composers*" and *"is absolutely perfect for those students*." He noted that the school owns Max software licenses for the workstations in the computer lab and that student licenses are affordable, *"the price of a book or two records."*

✦ THE STUDENTS Professor Joel believed the small class size was one of the course's strengths, as he got to know his students well. Allowing students had been participative, Joel believed they did not see FLOW as important as their individual Composition unit and thus were not as motivated. The professor added he did not believe students were sensitive to the value of learning to code. He also had the impression none of the students at FLOW had any prior familiarity with programming when they began the semester, contradicting their answers to our survey.

In recent years, the professor has seen qualitative changes in students' interest in learning to code. Though he did not believe there was an increased curiosity about the subject, he stated that learning to code *"is less polarizing nowadays; there are no longer people thinking that computers will destroy art, though you also don't see the same passion for making different things."* Professor Joel told us he believed music education is highly conservative, and more so than in his own time as a student. He explained that before the late aughts, students at music conservatories were also required to complete regular basic and secondary schooling, whereas *"nowadays we funnel music education,"* and specialization may start as early as in the fifth grade. He added:

> In higher education we get—more so nowadays—students whose paradigm is to become an oboist at an orchestra. Of course, in Composition we get the eccentric ones. Even so, [...] I would say the battle with students not sensitive to the matter [of the value of programming], is even more difficult than ten years ago.

The professor explained that conservatories rarely offer IT training, though students may attend workshops on audio editing and musical notation software. He believed more people are curious about digital art and technology in the fields of design and fine arts. Still, Joel also saw virtues to the traditionalism of music education:

> These are students that understand the importance of skill. That is something you might struggle to work with visual arts students, which is the importance of working eight hours every day for three years to achieve a good drawing skill. Music students understand that without needing any explanation.

Professor Joel told us his position that *"computers are the opposite of intelligence"* helps students overcome their difficulties with programming:

> I only need to explain to a student that programming is hard [...] because they are too intelligent. They need to learn to be dumber to be able to program! From the moment you pass on this message, their anxiety goes away.

---

* The professor explained Pure Data was also created by Miller Puckette, Max's original creator, and also talked at length about PD's strengths and weaknesses *vis-à-vis* Max, noting how it he had taught PD previously while teaching in Multimedia courses, given its greater interoperability with web and mobile applications.

For the last several years, Professor Joel had been working exclusively at master's and doctoral programs. Thus, he admitted he did not know how to assess the strengths and weaknesses of undergraduates. He believed first-year students were immature, though he expected it. He told us, *"a student called me 'teach,' which I don't think ever happened before. There was a secondary school ambiance, of students talking to each other in class, which I was not used to."* Maturity plays a role in students' attitude toward programming, but that maturity *"goes both ways."* The professor told us he is helping his children learn to code, *"and there is an enthusiasm that goes away when you turn sixteen [...] you regain the maturity later when you feel life is passing by faster and you need to get on board."* Joel noted these differences in attitude apply generally.

We asked Joel whether he noticed any differences in attitude toward programming depending on students' gender. Though there may exist differences imposed by society, the professor us, gender differences *"are fading, and rightly so."* He explained he was not concerned by a need to address any gender differential between students. The professor thought it senseless as *"all students are different, and it is important each student is unique within that difference."*

✦ EDUCATIONAL OUTCOMES Professor Joel was critical that *"we nowadays conflate objectives and evaluation, and often our evaluation criteria become the objectives:"*

> Students are used to meeting [objectives] from a young age. And in that sense, they do meet [objectives], which is great if all you want is to produce people that meet objectives. If you want to educate individuals, artists, that is terrible!

Students met FLOW's objectives, which Joel noted *"were not very ambitious"* as they concerned the basics. The professor told us, *"more than half the students do okay with Max, to the point of being able to help their colleagues."* However, he did not think he had been successful in persuading students of the value of learning to code.

The professor explained he was going to change the evaluation methodology for the coming academic year, possibly requiring three specific programming exercises. Joel reiterated he lacked conditions to supervise a composition project properly. Moreover, he had failed to persuade the school to let him conduct the formal jury evaluation he had intended, which led students to take their work for FLOW less seriously.

✦ ADDITIONAL REMARKS Professor Joel offered some remarks regarding our own research, about which he was very curious. He believed *"programming education models exist to teach computer programmers, and don't work to educate artists to make use of programming."* Given how artists most often do not start from a problem that needs a solution but instead experiment and *"find solutions to problems that only manifest themselves"* simultaneously, *"teaching artists to code needs a radically different model."* Joel told us he holds on to a twenty-year-old book* about Max *"because it is among the few with an interesting approach to programming"* for artists, and also praised Daniel Shiffman's books and videos as offering a novel approach. The professor noted, *"except for those two, I don't know any good models of programming for arts education."*

---

* The professor noted he was referring to *Composing Interactive Music: techniques and ideas using Max* by Todd Winkler. MIT Press, 1998.

Professor Joel added that he, and his colleagues in arts education, found their teaching strategy, *"but it is never formalized or discussed."* In addition, *"most professors teaching programming in art schools never even had formal programming lessons themselves,"* thus raising the question of *"how are you going to teach something you learned on your own?"* He suggested that a formal meeting about computing education for the arts would be valuable.

## 7.7 *Successes and limitations faced in this case study*

Though the small class brought some inherent limitations to the case study, we were pleased to be able to conduct it as planned. We obtained the necessary permissions in advance, and in addition, the responsible professor was supportive and agreed to most of our requests. We thus summarize the successes and the limitations we faced:

***Table 7.7.2.*** *Successes and limitations faced in the FLOW case study.*

| | *Planned* | *Unplanned* |
|---|---|---|
| *Successes* | Collected all types of data we sought and without delays. | - |
| *Limitations & Disappointments* | Limited number of students for quantitative survey.<br>Researcher's unfamiliarity with the musical field. | The public recital of students' work was cancelled. |

Though the researcher was quite comfortable observing other case studies' sessions due to his familiarity with visual arts and design topics, his unfamiliarity with musical theory and techniques presented difficulties in following the discussions during sessions. In addition, though we had made little plans concerning the analysis of students' work, we were looking forward to attending their public recital at the end of the semester, once that had been laid out as one of the unit's objectives during the first session we attended. We were disappointed that the recital was canceled and that, save for the brief excerpts played back during the final session, we were unable to listen to the practical outcomes of the curricular unit.

# 8 INTER-CASE COMPARISON

Following the individual case analyses reported in the previous six sections, we will turn our attention to the parallelisms and contrast such reporting evinced. Though we will follow a similar principle to organize this section—from the general to the particular, or from the collective to the individual –, we will also invoke professors' statements during their interviews at every step of the way: commentators in our documentary, if we are permitted to present it as such.

In the following subsection, we will therefore compare the characteristics of the curricular units', starting from the educational positions which inform them. In §8.2, we will look at the facilitating conditions for learning to code—temporal and spatial, hardware, software, and online. Then, in §8.3, we will compare how professors led their classes—their strategies to engage students and their specific behaviors in theoretical lectures and practical sessions. In §8.4, we will see how students reacted and behaved in different classrooms, and in §8.5, we will compare their acceptance and use of computer programming, as reported to our surveys. In §8.6, then, we will describe how students' work differed across cases and how professors regarded their learning outcomes. Finally, in §8.7, we will briefly address the limitations we faced in carrying out our studies.

## 8.1 *Units' positioning and characteristics*

As laid out in Table 7.1.1, each curricular unit that constituted each of our cases had unique characteristics that set them apart from each other; hence our cases conformed to the *maximal variation sampling* we regarded as inevitable given the heterogeneity of approaches to including computer programming within arts and design curricula (see **5**.5.1). Moreover, though that heterogeneity may be partially ascribed to institutional policies and the parent courses' positioning, the professors leading the units also have an undeniable latitude. Hence, we found it worthwhile to dedicate the following subdivisions to comparing professors' statements about how they see the salience of learning to code within an education in arts and design (in §8.1.1), and how they envision the relationship between the curricular unit they led and the wider course (in §8.1.2). We will then dedicate §8.1.3 to a comparison of the units' objectives and syllabi and §8.1.4 to a comparison of their evaluation methodologies.

### 8.1.1 THE VALUE OF PROGRAMMING

Professors Bill (PRIMER) and Edwin (SKETCH) believed it would be important for all to have contact with programming at some point during schooling, as to have the opportunity to discover a vocation. Though not in disagreement, Professor Alan (PRIMER) believed that too much a focus on digital competencies distracts from the more urgent need of addressing students' gap in logical thinking skills, to which they are often oblivious (see §2.7). Professors Douglas (GRAPHIC) and Grace (WEAVE) also emphasized the general importance of developing computational thinking skills, while Ivan (TOUCH) noted such skills permit more effective use of technology.

Admitting that programming is a specialized artistic technique, Professors Carl (PRIMER), Frank (WEAVE), and Joel (FLOW) joined Edwin in noting how fundamental it is for digital arts practitioners to be proficient at programing. Professor Carl decried the uncritical use of ready-made software by artists, whereas Professor Joel went further and told us it would be illegitimate for an artist exploring the potential digital media to hire a computer programmer to execute their ideas.

Professors Douglas, Edwin, Frank, Grace, and Ivan also discussed programming as a marketable professional skill for designers, with Edwin and Ivan noting the growing demand of the job market. Moreover, Douglas and Edwin also believed in the potential of programming and computational thinking skills to ease professionals' quality of life, both through automating processes and a greater capacity to solve problems.

### 8.1.2 RELATIONSHIP WITH THE COURSES

Each unit had a particular relationship to their parent course's curriculum. PRIMER (§2), GRAPHIC (§3), SKETCH (§4), and FLOW (§7) first introduced computer programming to their respective students. Prior familiarity was expected from those attending WEAVE (§5) and TOUCH (§6), as their courses' curricula included earlier units about computer programming. All units were mandatory except for GRAPHIC, though PRIMER and SKETCH were the sole mandatory contacts with code in their courses' curricula. In contrast, students attending WEAVE and FLOW would have a continual commitment to the subject throughout the rest of their courses. That was not the case in TOUCH as it took place in the final semester of the course.

Addressing the recent changes to the parent course's curriculum that had moved PRIMER from the third to the first year, Professor Alan believed it was important to provide the initial contact with programming at an early stage so students could choose to explore it throughout the rest of their studies (see §2.7). Indeed, we observed a lower commitment from students at TOUCH, which may relate to its particular course only introducing the subject in its third year. Still, Professor Carl, who assisted Alan, believed a single contact with code was insufficient and that it would have been desirable for the course to include mandatory subsequential units about programming, an opinion also shared by Professor Edwin while discussing SKETCH's parent course curriculum.

According to Edwin, students regard the multiple specialty units included in the curricular plan as matters of preference and tend to favor more immediately pleasurable subjects over learning to code (see §4.7). Professor Ivan agreed with that view, noting his own students' preference for videography over programming (§6.6). Professor Joel also told us institutional practices lead students to disregard the units dedicated to computer programming and electroacoustic music, even though these are assigned a significant number of credits in FLOW's parent course (§7.6).

Discussing the wider field beyond his specific institution, Professor Joel criticized the normalizing impetus of contemporary arts education, which he believed did nothing to nurturing students' individuality. Moreover, Professor Alan criticized how curricula were often shaped by a contradiction between meeting the demands of the job market and ensuring students acquire a wide enough range of skills to meet the same market's

volatility. Still, Alan positioned himself in favor of a wider curriculum and expressed the need for greater dialogue between arts education and science.

Professor Alan and Joel also agreed on the need to recenter arts education on the development of individual skill. Alan, whom we have witnessed as enthusiastic about the potential of Virtual Reality technologies, noted how education ought to find ways to address the paucity of digital interfaces as to bring back artistic virtuosity into the studio.

### 8.1.3 OBJECTIVES AND CONTENTS

The curricular units providing an initial contact with computer programming all shared the common objective of raising their students' awareness of the aesthetic and artistic possibilities granted by code. Indeed, Professor Joel told us it had been FLOW's foremost objective, though he recognized the syllabus and evaluation methodology would need adjustments in order to better persuade students of that potential (see §7.6).

Professor Alan defended PRIMER's reliance on project-based learning and autonomous study, not only because of the contingencies posed by large classes but also due to a need to *"theorize by making and making by theorizing"* (§2.7). Still, Alan considered that PRIMER's syllabus was large and suggested its possible division into two distinct units—one concerned with computer science and the other concerned with practical project development.

Professor Edwin told us SKETCH aimed to provide students with the minimal foundations to be able to learn autonomously afterward. He noted that a syllabus any shorter than getting to introduce the main concepts of Object-Oriented Programming would imperil that autonomy (§4.7).

### 8.1.4 EVALUATION METHODS

All curricular units under scrutiny had required students to prepare practical exercises or projects throughout the semester, though the nature of that work and its practical supervision were heterogeneous (refer back to Table 7.1.1): As long as they employed the recommended tools, students at PRIMER were at liberty to develop a project of their liking throughout the semester, working either individually or in groups. At GRAPHIC, students were required to present two small individual exercises according to precise instructions. SKETCH's students were also given precise instructions, but in addition to two individual exercises, they were also required to present a group project at the end of the semester. WEAVE and TOUCH presented similar types of practical requirements: students had to complete one project throughout the semester, complying with multiple precise milestones that implied the use of distinct programming languages (those at WEAVE were also required to complete a theoretical exam, however). Finally, students attending FLOW were given the liberty to write one musical composition as long as they integrated any algorithmic technique mentioned in the syllabus.

During their interviews, Professors Alan and Edwin demonstrated opposing views towards their evaluation requirements. Whereas Alan disagreed with closed instructions from a principle of engaging with students' individual interests (§2.7), Edwin disagreed with open projects, as he believed those allowed students to define themselves how they would be evaluated (§4.7).

## 8.2 *Conditions and resources*

The six cases we observed evinced a great variation in configuration and conditions. In the next subdivision, we will address the differences and the professors' perspectives about class size and scheduling, whereas in §8.2.2, we will compare cases' spatial and technological conditions. In §8.2.3, we will also compare how the professors' envisioned and used online platforms and resources.

### 8.2.1 CLASS SIZE AND SESSION SCHEDULE

Though students attending PRIMER (see §2.3) and SKETCH (§4.3) were divided into multiple classes, the researcher nonetheless felt these remained too large, exceeding 40 students at the former and 30 students at the latter. Professors Bill (PRIMER) and Edwin (SKETCH) complained about class size during their interviews, with Edwin suggesting 18 students as the ideal cap to class size (§4.7). In contrast, teaching in a course that admits no more than six students every year, Professor Joel (FLOW) noted how the small class size was one of its particular strengths (§7.6). Still, Professor Alan (PRIMER) noted the greater discipline a large class demands from professors can benefit students, though both he and Bill suggested a clear distinction between theoretical lectures and practical sessions would be desirable. Not only would it provide a theoretical backbone to the entire term, but also, if budgetary concerns were to be considered, a greater number of smaller practical classes could be created without an increase in the overall number of teaching assignment hours (§2.7).

The scheduling model Professors Alan and Bill suggested would be similar to what we witnessed from WEAVE, in which the practical classes met Edwin's ideal number of students. Nonetheless, WEAVE's Professors Frank and Grace believed the two practical hours per week assigned in the unit's schedule had been insufficient (§5.6). Edwin also believed the single weekly three-hour session assigned to each SKETCH class was barely enough to teach the unit's syllabus and tutor students on their practical work, especially given the classes' size. Still, Edwin was alone in all of our cases in also having a regular office hours schedule in which he had been able to supplement students' needs.

Professors Edwin, Douglas (GRAPHIC), and Ivan (TOUCH) also complained about the irregularity of the second semester's academic calendar, which placed several long gaps between sessions due to the Easter holidays, host institutions' Academic Weeks, and multiple national holidays that coincided with scheduled sessions; Ivan told us TOUCH had faced a year of particular 'bad luck' in that regard (§6.6). Douglas' comments on how students' level of enthusiasm and commitment tends to fall with time after each session, thus acquire a particular saliency. The professor had suggested dividing GRAPHIC's schedule into biweekly sessions would be beneficial (§3.5).

### 8.2.2 SPATIAL CONDITIONS, HARDWARE, AND SOFTWARE

With the exception of two WEAVE theoretical lectures, which took place in an adequate auditorium (see §5.3.1), all sessions we observed were classified as theoretical-practical or as laboratory practice. These took place in a variety of rooms: PRIMER's large studio

that was nonetheless crowded (moreover, as outside students were allowed to use its equipment during sessions); GRAPHIC's long classroom, which had a high ceiling and bad acoustics; SKETCH's large computer laboratory, in which many students' view of the professor (and vice-versa) were obstructed by rows of large computer monitors; WEAVE's small computer lab, crowded despite students' absenteeism, as Professor Frank noted in his interview (§5.6); TOUCH's large computer studio, which exceeded the needs of the class' size; and FLOW's small music classroom, permeable to noise from the outside. Indeed, of all interviewed, only Professor Ivan believed he had good spatial conditions during TOUCH's sessions.

✦ PROJECTION SYSTEMS Possibly the most important equipment in the classrooms and studios were their display systems (projectors or, in FLOW's case, a television set). Several of these systems featured projection screens that were too small in relation to the size of their room; that was the case at the auditorium where we witnessed WEAVE's theoretical lectures or at PRIMER and SKETCH's computer studios. In other cases, such as TOUCH's studio, the room's ambient light prevented good visibility until later in the afternoon. Still, we saw projection had been a constraint even when it had been the most adequate, in rooms equipped with large screens and blinds or curtains, such as where we had witnessed GRAPHIC's sessions.

Professors were in constant need to balance the readability of code or text with the need to display a substantial part of their demonstration programs. Though smaller displays made it more difficult, all professors nonetheless had to experiment and change code font sizes throughout their lessons; in addition, students often asked professors to scroll up and down the code on display, which often confused colleagues. These difficulties were critical during FLOW's lectures, as Max's visual diagrams were harder to comprehend than text and easily grew past the confines of one screen; hence, even in the intimate environment of a class of six, students progressively disengaged from the professor's demonstrations as their complexity grew.

FLOW's Professor Joel attempted to mitigate the screen's limitations by frequently pointing at the screen and drawing diagrams on the room's whiteboard. We do note Professor Douglas complained about the lack of a board in the room where GRAPHIC sessions took place. Moreover, though projection issues were likely not the only reason for his approach, WEAVE's Professor Frank limited his lectures' demonstrations to small examples which mostly fit one screen, even when placing source code and result in windows side-by-side. In their interviews, Professors Douglas (GRAPHIC) and Edwin (SKETCH) speculated about using screen-sharing software or a real-time platform for presenting code directly to students' computers (see §4.7, §5.6).

✦ COMPUTERS AND PERIPHERALS Though we found no document that made into an explicit requirement, it was clear students were expected to own and bring their own laptop computers to classes. GRAPHIC and FLOW's theoretical-practical sessions were held in classrooms without computing equipment available for students. Moreover, the computer laboratories and studios we observed did not have enough computers in the event all students wished to use them. In any case, most students preferred to use their own laptops. They ignored the outdated machines at WEAVE's and TOUCH's computer

studios, and though some used the more up-to-date workstations which were available in SKETCH's and PRIMER's classrooms, we never witnessed any situation in which a student lacked a computer. There was specialized computer equipment available to those attending PRIMER—two high-end workstations used to drive the Oculus Virtual Reality kits –, and indeed, those computers saw frequent use. Professor Alan told us he agreed with the faculty's policy of purchasing fewer but higher-end equipment as to supplement students' own (see §2.7).

✦ SOFTWARE AND HARDWARE REQUIREMENTS With the exception of FLOW, the curricular units required students to use software that was freely available for multiple platforms. However, but for Processing and P5.js, those tools were not free software. PRIMER students wishing to distribute the projects they made in Unity would have to acquire a commercial license. Students interested in obtaining free academic licenses for the WebStorm and PHPStorm development environments recommended in WEAVE would have to sign up and confirm their academic credentials (though there are many free software alternatives). Moreover, some professors recommended online platforms, such as Codepen, P5.js', and Arduino's,* which required students to create an account. Though FLOW students could use Max outside of class in the workstations available at their school's computer laboratories, Professor Joel suggested they purchased a personal student license. Though we note the additional charges asked of those students, we find it may be argued that requiring the use of explicitly commercial software such as Max may be preferable† to promoting free (as in *gratis*) options with hidden costs (e.g., mining of personal data or onerous licensing for future commercial use).

TOUCH students were also taught the basics of how to wire and program Arduino. Professor Ivan told us the school had recently purchased enough Arduino kits for his students to use within the facilities. Each such kit included the microcontroller board alongside electronic prototyping breadboards and a selection of components, cables, sensors, and actuators. As the school did not loan the kits for home use, we saw that a group of students opted to purchase their own in order to work on their project outside of the school (and possibly as to be able to keep it).

One of WEAVE's sessions we witnessed was dominated by the difficulties that students faced in installing and configuring software in their computers. Though the researcher was no stranger to the troubles of configuring a local web server, it was felt a crucial practical session was nonetheless wasted. Though they may present their own problems, we believe the distribution of virtual machines, as suggested by Corbi and Burgos (2017), could mitigate the differences between students' personal computers that often raise issues when installing software stacks, such as is the case of a web server.

* We note that, e.g., even though the P5.js language is free software, the code hosting platform has its independent licensing and privacy policy.

† It should be noted that is the opinion of the researcher.

### 8.2.3 ONLINE SUPPORT AND RESOURCES

One particular heterogeneous aspect of our cases was in their units' use of online learning management systems and other platforms to support and provide students access to educational resources. We summarize cases' leveraging of the internet in Table 7.8.1. Though not listed, we note all also shared internal content related to the curricular units themselves (e.g., syllabi, lesson plans, notices).

***Table 7.8.1.*** *Overview of the data we collected in each case study.*

| *Case* | *Platforms* | *Support* | *Internal content* | *External resources* |
|---|---|---|---|---|
| PRIMER | Moodle LMS | Email, Forums | Coursework instructions; Open project submissions | Readings, Art, Tutorials |
| GRAPHIC | Shared folder (Google Drive) | Email | Lecture code; Coursework instructions | - |
| SKETCH | Moodle LMS, Weblog, YouTube | Email | Lecture notes, slides, code; Coursework instructions; Video lessons | Readings, Art, Tutorials |
| WEAVE | Website (Codepen) | Email | Lecture notes, slides | Tutorials |
| TOUCH | Shared folder (P5.js profile) | Email | Code | - |
| FLOW | Static website | Email | Code | Readings, Art, Tutorials |

Leading PRIMER, Professor Alan leveraged the faculty-provided Moodle installation not only to share internal (i.e., produced by the professors) and external resources, but also as a central mean to communicate with students through forums dedicated to projects' development (see §2.4); keeping with Alan's open studio strategy, Professors Alan, Bill, and Carl also requested students to submit their projects to an open forum.

In all other cases, professors favored the use of online platforms for unidirectional communication and sharing of content with students; though Professor Edwin set up forums in SKETCH's Moodle, these remained unused by students (§4.4). Outside of the faculties or schools, email remained the main medium to support students.

Professors showed varying enthusiasm for using online platforms. At GRAPHIC and TOUCH, Professors Douglas and Ivan merely set up shared folders (see §3.4, §6.3.2). At WEAVE and FLOW, Professors Frank and Joel built websites where they shared some of their lectures' content and external links (§5.4, §7.4); Frank opted for using a dynamic platform that allowed students to complete tutorials directly on the website.

Professor Edwin was the most enthusiastic at employing online platforms at the service of SKETCH's students. Besides Moodle, he also set up a weblog in which he shared lecture notes and posted about tutorials and artworks and artists students could find inspirational. In addition, though Edwin regarded video as less efficient than text for sharing programming tutorials and advice, he told us his students greatly favor video, and thus he acquiesced to their preference (see §4.7), sharing recorded lessons on a YouTube channel. Still, Edwin told us he was looking to leverage video as a means to avoid repetition and implement a 'flipped classroom' in future instances of SKETCH.

✦ A NOTE ABOUT LANGUAGE A command of the English language is perhaps an unavoidable requirement for success in higher education in most parts of the world, moreover when the focus lies on technical specialties such as computer programming. Unsurprisingly, nearly all of the external tutorials and other resources provided by professors were in English, as were most titles in the units' bibliographies; only WEAVE's included technical books in the Portuguese language. Conversely, that unit's lecture notes were provided in English, whereas SKETCH's were provided in Portuguese.

## 8.3 *Professors' strategies and behavior*

> How are you going to teach something you learned on your own?
> – *Professor Joel* (§7.6)

Professor Douglas and the three docents leading WEAVE (Frank, Grace, and Hal) had formal studies in computer science or computer engineering. However, the rest of the professors in our studies admitted they had learned computer programming by themselves. Moreover, as Professor Joel stated, those teaching arts and design students to code rarely discuss their teaching strategies with other professors. He noted how arts education requires a specific model of computing education yet to be formalized.

The professors told us the principles that underpin their approaches to teaching computer programming. All professors shared an understanding that students learn by watching them work on programming examples; Professors Carl, Douglas, and Edwin explicitly noted so during their interviews (see §2.7, §3.5, §4.7). Carl and Edwin also emphasized the importance of introducing only a few new concepts in each lesson. Moreover, Professor Alan emphasized the importance of maintaining an open studio, so all could also learn from their own colleagues.

⁂

All professors understood the subject's difficulty to first-time learners and the centrality of managing students' interest and motivation. In the following subdivision, we will review the different strategies shared by the professors, and in §8.3.2 and §8.3.3, we shall review how they translated those to their theoretical lectures and practical sessions.

### 8.3.1 MOTIVATIONAL STRATEGIES

Most professors agreed that students often face anxiety and present a negative attitude towards computer programming; Professor Douglas (GRAPHIC) even noted how some students punish themselves through repetitive tasks rather than face up to their anxiety. Professors Edwin (SKETCH) and Ivan (TOUCH) believed that students' anxiety tends to fade during the semester; Both Douglas and Professor Frank (WEAVE) told us anxious students were at risk and likely to give up on the discipline (see §3.5, §5.6).

Professors explained some of their strategies to motivate students and mitigate their fear or prejudices against computer programming. Hence, Edwin emphasized how the first few lessons are crucial, whereas Alan (PRIMER) and Douglas also emphasized the importance of demonstrating immediate results. Edwin also found showing previous

years' projects to be an important means of persuading students they will be able to overcome their difficulties (§4.7). Professors Bill and Carl persuaded students that code is one artistic technique among many; Bill explained to his students at PRIMER that they would be learning the basics, not signing a *"lifetime contract"* (§2.7).

At WEAVE, Professor Frank favored a 'divide and conquer' approach of breaking down complex problems into smaller tasks, which he supplemented with giving students positive reinforcement. Professor Grace would tell her students how one needs to make mistakes; she told us it was important not to judge. Like Edwin, Professor Joel (FLOW) would also tell his students that programming is challenging because *"computers are the opposite of intelligence,"* hence attempting a reversal of the negative assumptions of students having difficulties (§7.6). This need to negate novices' belief in a 'superbug' lurking within the computer is also established in the literature (Pea, 1986).

Professor Alan believed it was important to find students' personal interests and to let them develop open projects that interrelate programming with those interests. Professor Frank conciliated precise project requirements with an open theme, which he reckoned motivated students. Professor Ivan noted how a public exhibition of their work motivated the students attending TOUCH.

### 8.3.2 THEORETICAL LECTURES

We observed over 26 hours of content exposition across four of our cases: GRAPHIC (see §3.3.2), SKETCH (§4.3.2), WEAVE (§5.3.2), and FLOW (§7.3.2). Theoretical lectures at WEAVE were separate from the practical sessions and gathered all students at an auditorium; in the other cases, lectures were a part of theoretical-practical sessions.

✦ PHYSICALITY Though professors' personalities and preferences shaped their lecture style, we also witnessed the conspicuous influence of class sizes and of the classrooms' spatial characteristics. Professors Douglas, Frank, and Joel were likely to spend their lecture sitting at their desk; moreover, whereas Douglas and Frank spoke calmly and addressed an audience, Joel had a more conversational tone, addressing individual students. If Frank was indeed on stage at an auditorium before tens of students, Douglas addressed a group of students sometimes as small as Joel's class, but did so in a larger room where students sat further apart. Fittingly, Douglas' desk was placed on a stage.

Professor Edwin, however, was likely to remain seated for just the time he needed to write code. Sitting at his desk at the side, the professor was likely to see his students behind 'walls' of large computer monitors. Hence, Edwin stood to pace around the front of the classroom to address his students, sometimes as an audience, sometimes the individuals sitting nearest.

✦ LANGUAGE International mobility students attended SKETCH and FLOW, and thus Professors Edwin and Joel spoke in English when they were present. Local students never complained, and the professors often clarified in Portuguese what they had just said. Still, we felt the use of a second language (including to the mobility students) coincided with lesser participation in class (though we note the English-language FLOW lectures we attended addressed more complex parts of the syllabus).

- REST Whereas Professors Douglas and Edwin allowed GRAPHIC and SKETCH students a short break at some point during their three-hour sessions, neither Professors Frank nor Joel gave WEAVE and FLOW students an interval during their two-hour lectures. Still, whereas Frank kept his pace, in which brief explanations were followed by brief examples all throughout the session, Joel periodically returned to conversations about theoretical and aesthetic aspects of algorithmic composition, which were a means to allow students a break and refocus their attention.

- ENGAGEMENT We found the audiences at Douglas' and Frank's lectures rather passive; the professors rarely bid for students' attention or posed non-rhetorical questions. On the contrary, Professor Edwin made constant bids for attention during SKETCH's lectures. When posing questions to individuals, Edwin patiently waited for answers before proceeding. He also struck a humorous tone, often making jokes and employing out-of-place devices such as having a bowl of candy available to award correct answers. In doing so, the professor risked facing insolent behavior, though we appreciate the engagement he obtained from his large classes, moreover as they were attending the lectures in an unfavorable space. In that regard, Professor Joel benefitted from lecturing in a situation we saw close to ideal, notwithstanding some of the room's limitations: facing the few FLOW students at a conversational distance.

- EXPLANATIONS AND DEMONSTRATIONS All four professors under discussion resorted to computer programming demonstrations as their main lecture technique. They used the classrooms' projectors to 'mirror' their own computers' screens and then asked students to reproduce, in their own computers, the code demonstrated. Though we had not witnessed those sessions, Professor Carl (PRIMER) also told us that had also been the main teaching approach during the term's initial lessons. Though Professor Edwin told us he benefitted from 'rehearsing' SKETCH lectures while recording video lessons, he explained that he came to consider the coding demonstrations a 'defect' and was looking to implement a 'flipped classroom' in the future (see §4.7).

  Professors Edwin and Frank also used PowerPoint decks and alternated their slide presentations with their coding demonstrations. Both used simple slides with little text in them to present definitions and code excerpts; unlike Frank, Edwin also displayed notices and his exercises' instructions, as well as images and videos related to the work of pertinent artists or designers.

  At GRAPHIC and FLOW, in contrast, Professors Douglas and Joel did not use slide presentations. Douglas presented videos and documents (e.g., a research paper) related to the lesson's topic only before beginning his coding demonstration. In turn, Joel showed a particular style in which he simultaneously talked and conducted web searches and presented images or videos related to the topics under discussion. Unlike Edwin and Frank, who alternated exposition and demonstration throughout their lectures, Joel's lectures, like Douglas', presented clear boundaries between theoretical discussions and coding demonstrations.

✦ THE VALUE OF PROGRAMMING We witnessed two distinct approaches to proving computer programming's usefulness and salience to students' practice. While Professor Joel demonstrated Max's usefulness for generating music in his first lesson, Edwin waited until the seventh week to introduce Processing's active animation mode, which finally revealed results not attainable in drawing software applications. Nonetheless, we do note SKETCH students had attended a day of masterclasses and exhibitions prior to the first lesson, which had already presented the potential of computer programming. Professors Douglas, Edwin, and Joel were careful to address the aesthetic salience of the topics of their lectures, presenting examples of artworks and compositions that applied the techniques addressed, even when the reasons for using computer programming were not immediately obvious. Moreover, Professor Joel took care to demystify some of the algorithmic compositions he presented, telling students their programming knowledge was already equal to the task of creating similar work. In addition, Joel was sensitive to discussing the relationship between technological progress and artistic practice.

✦ METAPHORS AND COMPARISONS Both Professors Edwin and Joel were adept at finding metaphors to explain certain topics: Edwin explained Processing's 'active mode' main loop function as defining *"the track a car races around,"* with said car's characteristics being determined by the `setup()` built-in function. He also addressed the principles of object-oriented programming (OOP) by referring to the movie *Inception* to explain how objects may exist within the scope of other objects. Professor Joel, addressing Max's OOP characteristics, compared using the language to the setting up of an analog synthesizer—a network of inputs and outputs—which he contrasted to the sequential operation of a cash register, standing for imperative programming. While Edwin and Joel expected to teach first-time learners, Professor Frank did not. Hence, in WEAVE's lectures, he often noted similarities and engaged his students' prior experience with other programming languages in preceding curricular units.

Both Professors Edwin and Joel were likely to make remarks such as *"the computer is stupid"* throughout their lectures as to dispel students' anxiety. As they noted during their interviews, novices' difficulties often result from their erroneous assumption that machines have any intelligence (see, e.g., §7.6).

✦ PROFESSORS' CODE Though all wrote their code confidently, Professor Edwin had a few difficulties when he attempted an unrehearsed demonstration and admitted to constantly need to refer to documentation, which he presented as a natural part of any programming task. Professor Douglas, an experienced computer programmer, evinced some personal habits that may have caused difficulties for students to follow (e.g., not commenting the code he wrote, employing abbreviations for variable and function names that may have looked cryptic to novices, and typing fast and only correcting his typos while revising). Douglas admitted he rarely used Processing on his own, so even though he presented an absolute command over his algorithms, he also had to refer to documentation often. Though he also noted the use of documentation was a part of the task, we believe it would also be a good reason to annotate the code presented in class and later shared with the students.

Both Professors Edwin and Frank commented their code, employed appropriate naming conventions, and corrected any mistakes immediately. Indeed, Edwin was likely to copy a function's argument list from the Processing documentation and paste it into his code example as a comment for easy annotation. Still, as we pointed out in describing his lectures (see §4.3.2), Edwin sometimes lent his humorous tone to his code, changing variable names to reflect his jokes, confusing students following his demonstration. Throughout his lectures, Joel appeared confident in his Max proficiency. Yet, owing to the researcher's lack of familiarity with Max and the visual paradigm in general, it was harder to assess Joel's use of the language.

✦ MANAGING THE DIFFICULTY Even in a highly personalized environment, such as observed in FLOW, we felt the professors often lost track of the complexity of their own demonstrations. Professors Edwin and Joel sometimes took students' comprehension of the overall structure of their example code as granted: we witnessed multiple instances in which they iterated and improved their code while failing to check students were able to follow, especially given the limitations of screen projection. As that student at FLOW put it, *"the train did not stop"* (§7.3.3). Professor Douglas (GRAPHIC) too did not check whether his students were following his demonstrations, though he always presented a step-by-step walkthrough of his entire code upon completing an example.

As noted, Douglas' lessons presented different computer graphics techniques, yet we witnessed demonstrations that refrained from recalling concepts and techniques he had presented earlier. Still, the contrasting approach we witnessed in WEAVE's lectures, in which professor Frank only presented brief and oversimplified examples (see §5.3.2), did not appear to make an impact on students' interest.

All professors accentuated the need for autonomous study. Professors Edwin and Joel suggested small homework exercises that were independent of students' larger projects. Edwin also stressed that students ought to develop a habit of reading languages' documentation before posing questions or looking for answers in online forums.

### 8.3.3 PRACTICAL AND TUTORING SESSIONS

We estimate we observed an aggregate over 33 hours of sessions the professors dedicated to supervising students' practical work across all our cases: PRIMER (see §2.3.2), GRAPHIC (§3.3.3), SKETCH (§4.3.3), WEAVE (§5.3.3), TOUCH (§6.3.2), and FLOW (§7.3.3). Understandably, the existing conditions and the evaluation method shaped the type of supervision granted by the professors. Except for FLOW's, exclusively dedicated to presenting questions, students worked on their projects and exercises during sessions.

✦ PHYSICALITY While Professor Joel and his students discussed projects while sitting at their respective desks, placed at a conversational distance, all the other professors would pace around their room from student to student or group to group, offering their assistance or checking on their projects' progress. Only Professor Frank called on WEAVE's students to come and take a seat beside him while checking on their projects' milestone submissions; nonetheless, apart from that moment of greater formality, Frank also stood and paced around the room to check on students.

✦ AVAILABILITY The three professors leading PRIMER did not have enough time to assist most individual students or groups present in the sessions we witnessed; as we reported in §2.3.2, this happened even in the face of significant absenteeism, especially during the shorter sessions, which testifies to the excessive class size. In his interview, Professor Edwin also complained about the size of SKETCH's classes (see §4.7); indeed, though the professor was able to grant students developing their projects a reasonable amount of time, that was only possible due to significant absenteeism in the sessions we witnessed. In comparison, WEAVE's formal distinction between theoretical lectures and laboratory practice sessions—divided in classes with a reasonable number of students—typifies a more effective use of professors' allotted time.

Owing to their small classes at GRAPHIC and TOUCH, Professors Douglas and Ivan were also able to talk to every student or group that desired to do so; leading a four-hour laboratory session where students worked on their group projects, Ivan was able to spend a significant amount of time interacting with each group twice; we note TOUCH's practical session we witnessed had the least absenteeism of all cases.

✦ EXPLANATION STRATEGIES The professors employed several means to assist students making questions or reporting difficulties with computer programming. All professors were likely to give verbal explanations unsupported by any other means, though Professor Joel (FLOW), perhaps owing to the conditions of his sessions, was quick to demonstrate solutions by devising Max programs, displayed in the room's display for all students to see. Conversely, Professor Carl (PRIMER) was likely to provide long and detailed verbal explanations, often in conversation with students about how they explained their code.

Though all were seen sometimes to write down notes or diagrams as to aid their explanations, Professors Edwin (SKETCH) and Ivan (TOUCH) were particularly likely to do so, either drawing on paper or on the whiteboard; when using the whiteboard, both professors were also likely to ask for their whole classes' attention so more students could benefit from their explanation. Ivan was also adept at handling objects (e.g., pencils or bottle caps) in lieu of drawing diagrams. As he explained during his interview, Edwin looked to avoid repeating himself while tutoring students (see §4.7). Hence, alongside Professor Alan (PRIMER), Edwin exploited any similarities between students' problems and called students to come and help out their colleagues or suggested that different groups worked together.

✦ HANDLING STUDENTS' COMPUTERS With the notable exception of Professor Edwin (SKETCH), who deliberately avoided doing so, all the other professors were likely to handle students' personal computers. The researcher felt this fact was noteworthy and that multiple measures of acceptance—students' effort expectancy, self-efficacy, and computer anxiety—were at stake in those moments. Professors were likelier to handle the computers of those with the greater difficulties in learning computer programming, those being the students most likely to ask for assistance. As seen in §6.3.2, a student at TOUCH that was confident in his abilities resisted the professor's request to handle his computer.

Given the contrary gender imbalances between students (predominately female) and docents (overwhelmingly male), we were concerned about professors' handling of personal computers negatively affecting female students' outcomes, as those had been less accepting and confident about computer programming in our preliminary surveys (see **6.2**).* We shall discuss this salient observation in the next chapter. We ought to emphasize we do not suggest professors' behavior was untoward; indeed, observation led the researcher to scrutinize his own past teaching behavior and to appreciate the possible harm of actions that may appear pragmatic when leading a practical class. That Professor Edwin stated his unrelated reasons to avoid handling students' computers and admitted not to have considered other possibilities (see §4.7) testifies to this fact.

✦ WRITING STUDENTS' CODE We witnessed a few situations in which Professors Douglas and Ivan, while handling their students' personal computers, wrote code that went beyond unlocking a momentary difficulty and got closer to completing a significant part of the students' work. Though the professors asked students for further iteration, writing a significant part of students' work may harm their self-efficacy and lead them away from intending to use computer programming. In contrast, we witnessed two situations in which Professor Alan left the students' difficulties unresolved after using their computer for some time; incidents likely to raise a novice's expectation of effort.† We also witnessed situations in which Professors Frank and Ivan read students' code and asked them to explain it step-by-step; though some struggled, we characterize such challenges as beneficial.

✦ PATIENCE Most professors were patient and did not evince any negative bias towards individual students. We did witness a few situations in which Professor Bill showed some exasperation, which we at the time understood to relate to the classroom conditions and to time pressures. Frank was more likely to present a lack of patience with particular students he was tutoring; we understood there was a likely previous familiarity in such situations—e.g., the student was repeating the unit.

✦ AMBITION MANAGEMENT In PRIMER or FLOW, where students had the liberty to propose their projects within mild constraints, professors sometimes had to ask for lower ambitions owing to students' inexperience and the time that was available. Professor Carl explained such situations require care not to harm students' motivation; hence, rather than asking for lesser ambition, he would ask students to complete basic tutorials related to their projects, so they slowly understood the complexity they entailed (see §2.7).

Given precise instructions, a few students were likely to limit their commitment to the minimal requirements. Hence, we also observed situations in which the professors demanded greater ambitions from their students at GRAPHIC and SKETCH.

---

* We regret not having been able to witness sessions led by Professor Grace (WEAVE).

† We do not expect professors to be able to solve all students' problems immediately in the classroom, but rather wish to emphasize the risks of presenting low efficacy while handling students' computers.

## 8.4 *Students' characteristics and classroom behavior*

Professors Edwin (SKETCH), Frank and Grace (WEAVE), and Ivan (TOUCH) believed their students had been sensitive to the value of learning to code, though Frank and Ivan questioned students' commitment to their work (see §4.7, §5.6, §6.6). In contrast, Professors Bill (PRIMER) and Joel (FLOW) did not think their students had valued the subject of computer programming (§2.7, §7.6). Moreover, Professors Alan (PRIMER) and Joel believed their students were rather immature, though Joel also noted Music students were particularly sensitive to the importance of practicing a skill.

Professors Bill, Carl, and Edwin believed the number of students with an active interest in learning to code had remained stable since they started to teach (between 7 and 15 years earlier), though the growing availability of creative code tools has made it easier. Professors Alan, Grace, and Joel also believed that gender differences relating to students' intrinsic interest in computer programming were rightly fading.

Professors Alan and Carl believed their younger students presented increasingly better information technology skills, and conversely, their older students were likely to struggle with digital tools. In contrast, Professor Edwin believed his students brought a lack of basic IT competencies from earlier schooling. Indeed, we witnessed a significant number had great difficulties with basic computing tasks such as managing their files and folders during SKETCH's sessions (see §4.3.2).

⁂

In the next subdivisions, we shall examine aspects of students' behavior that were salient across theoretical lectures and practical sessions.

### 8.4.1 BEHAVIOR IN THEORETICAL LECTURES

In §8.3.2, we compared how professors approached lectures across GRAPHIC (see §3.3.2), SKETCH (§4.3.2), WEAVE (§5.3.2), and FLOW (§7.3.2). Though our focus was mostly held by the professors' presentations, we found salient aspects to students' behavior:

- ATTENDANCE A significant number of students did not attend WEAVE's and GRAPHIC's lectures. Several that had elected to enroll in GRAPHIC had scheduling difficulties, yet those enrolled in WEAVE were obliged to attend a certain number of sessions; given the disinterested behavior we witnessed, it is possible the lectures would have had an even lower attendance if not for that requirement; indeed, Professor Frank admitted to the 'stigma' surrounding theoretical sessions (see §5.6).

- ENGAGEMENT We witnessed significant inattentiveness from many students at WEAVE's theoretical lectures: most neither used their computer nor took notes, and several of those seen using their computer were either working for other curricular units, spending their time browsing social networks, or—in two particular instances—silently watching movies or playing computer games. Though not as blatantly disengaged, some students also spent their time at GRAPHIC's lectures performing work that was unrelated to that curricular unit. In comparison, the students attending SKETCH and

FLOW appeared to pay closer attention to lectures and frequently responded to their professor's questions and bids for engagement.

Students showed increasing difficulties in following Professors Edwin and Joel's programming examples as the semester progressed or, within each lesson, as professors iterated on their own code. Later in the semester, students attending FLOW also became uninterested if the topic at hand did not relate to their own ongoing project. Students at GRAPHIC and SKETCH were particularly shy about asking their professors questions or clarifications; Douglas ascribed that shyness to undergraduate students in general (see §3.5). At SKETCH, we saw that students preferred to ask their own colleagues for help. Those students were also visibly tired halfway through lessons (and more so during the morning class, lectured in English).

### 8.4.2 BEHAVIOR IN PRACTICAL SESSIONS

In §8.3.3, we compared professors' approaches to tutoring across all our cases: PRIMER (see §2.3.2), GRAPHIC (§3.3.3), SKETCH (§4.3.3), WEAVE (§5.3.3), TOUCH (§6.3.2), and FLOW (§7.3.3). We found some salient aspects to students' behavior while working on their projects and interacting with their professors and colleagues:

- ATTENDANCE Except for TOUCH's, there was significant absenteeism in most practical sessions we had the opportunity to observe. Professor Carl noted the likelihood of greater absenteeism in PRIMER's early morning sessions, though he also told us it was possible his students were trying to avoid admitting a lack of progress in their work. Professor Alan explained he usually carries out roll calls, which may dissuade some students from skipping class.

- PHYSICALITY Either inevitably due to classroom's design and the available seats, as in PRIMER and WEAVE, or by choice, as in TOUCH, students often sat working with their backs turned to their colleagues and the professor. In PRIMER's and TOUCH's studios, where they could sit facing each other, we saw students sat in clusters, larger than their project group, often separated by gender. We did not observe such a clear separation between students at SKETCH, where students sat in rows. As typical of studio work, students were free to move around and take breaks; either going outside for a few minutes or otherwise pausing their work to check social media or their phone.

- ENGAGEMENT WITH EDUCATIONAL RESOURCES Students were also likely to read or watch relevant resources the professors had recommended as they worked on their projects. Students at PRIMER, SKETCH, and WEAVE were likely to browse the websites and platforms (e.g., Moodle or Codepen) where their professors shared educational resources. Professors Edwin and Ivan noted the important role of video tutorials and how these meet the preferences of the current generation of students. Indeed, we observed multiple instances in which students watched video lessons and tutorials (often wearing headphones in class); those at PRIMER were particularly likely to be seen doing so, given the unit's reliance on students' individual exploration.

✦ INTEREST AND EFFICACY We witnessed situations that either testified to students' lack of motivation, commitment, or poor time management skills. Several students and workgroups at SKETCH and TOUCH were late in presenting and discussing their project ideas; later, we saw students showing little progress with little time left until their presentations. We also witnessed students requesting the postponements of one of GRAPHIC's exercise deadlines, as well as one of the intermediate TOUCH submissions, which were both granted.

A few students at PRIMER were upfront about their lack of motivation when discussing their work with their professors, and so was one group at TOUCH and one student at FLOW. Professors Carl and Douglas explained they believed younger students were more motivated, as their older colleagues were likely to have solidified their preferences (§2.7, §3.5). In contrast, Professors Grace and Ivan told us they believe older students are more dedicated to solving problems (§5.6, §6.6).

A significant portion of students at PRIMER and TOUCH admitted they had programming difficulties. Professor Edwin believed many lacked tenacity and passion for exploring the potential of programming, whereas others were likely to struggle due to their outsized project ambitions (see §4.7). Conversely, a few students evinced pride and confidence about their abilities; one individual student paced around the WEAVE computer room to aid her colleagues.

✦ AUTONOMY AND SELF-EFFICACY Most students worked autonomously and were focused on the task at hand during most of the practical sessions we observed. Indeed, but for Douglas and Grace, the remaining professors we interviewed praised their students' autonomy. Professor Alan believed PRIMER students' good grades out of secondary school signaled a predisposition for autonomous study, though he also noted the need to teach some research skills at the early stages of the curriculum (§2.7). Indeed, Douglas noted first-time learners of programming face a 'nomenclature problem' that complicates effective autonomous research (§3.5). In addition, Professor Bill told us he did not believe his students' autonomy had extended to a commitment to research about computer art and aesthetics.

## 8.5 *Students' acceptance and use of computer programming*

Besides observing their behavior, we conducted surveys of students' acceptance and use of computer programming, following the procedures described in **5**.4. A total of 206 individual students participated in these surveys, whose findings we reported in §2.6, §4.6, §5.5, §6.5, and §7.5. In order to compare responses to our first* survey applications across our cases, we employed single factor analyses of variance (ANOVA), as suggested by Richardson (2011), to investigate associations between responses and participants' provenance. Though otherwise disregarded given the few participants, we included in these analyses the responses we obtained from students at GRAPHIC ($n$=6, see §3.6) and FLOW ($n$=5, see §7.5) to single-scale measures (perceived voluntariness, programming use frequency, and use intensity).

* Only 43 participants took the second survey, which precluded a robust analysis.

✦ PERCEIVED VOLUNTARINESS We discovered a significant association between the perceived voluntariness and the specific curricular unit of participants that reported a prior knowledge of computer programming ($n$=90; $F$=3.22; $p$=.011; $\eta^2$=.17). In a transposed scale running from -3 (totally compulsory) to +3 (totally voluntary), participants attending FLOW ($n$=4; $M$=1.75; $SD$=1.50) and WEAVE ($n$=29; $M$=0.68; $SD$=1.16) saw the use of computer programming in the curricular unit as mostly voluntary; participants attending PRIMER were rather neutral ($n$=30; $M$=0.00; $SD$=1.70), and those attending SKETCH ($n$=14; $M$=-1.08; $SD$=2.35) and TOUCH ($n$=10; $M$=-0.80; $SD$=1.62) were more likely to view programming as mandatory.*

✦ ACCEPTANCE MEASURES Given their differing reliability outcomes and calculations, we could not look for meaningful associations between participants' specific curricular unit and their *Performance Expectancy, Effort Expectancy,* or *Self-Efficacy*. Still, among those without any programming knowledge, *Attitude Toward Use* ($n$=32; $F$=4.15; $p$=.015; $\eta^2$=.31) evinced a high sensitivity to the specific curricular unit. The 21 SKETCH participants (see Table 7.4.3) had a more favorable *Attitude* than their colleagues at PRIMER or TOUCH (Tables 7.2.3 and 7.6.1, respectively), which may testify to the former being motivated by the Processing Community Day event they attended in the week prior to completing the survey, as described in §4.3. We did not find further significant associations, either among the responses given by first-time learners or participants with a prior familiarity with programming.

✦ BEHAVIORAL INTENTION We found significant associations between participants' specific curricular unit and their *Behavioral Intention* across all groups. This association was strongest among those without any programming knowledge ($n$=32; $F$=11.85; $p$<.001; $\eta^2$=.56) or among those with a prior familiarity ($n$=83; $F$=13.88; $p$<.001; $\eta^2$=.35). Among first-time learners, the association between *Behavioral Intention* and their curricular unit was slight and nearly insignificant ($n$=80; $F$=2.90; $p$=.040; $\eta^2$=.10).

As seen in Tables 7.2.3 and 7.4.3, the participants lacking programming knowledge that were attending PRIMER reported less *Behavioral Intention* to learn the subject than those attending SKETCH. In addition, the four such participants enrolled in TOUCH were unanimous in their total dismissal of the subject ($M$=-3.00; $SD$=0.00) when they completed the survey halfway through their semester.

Among those participants with prior coding experience, those attending WEAVE evinced the highest *Behavioral Intention* to use computer programming in the near future ($n$=29; $M$=1.47; $SD$=1.42), followed by their colleagues attending PRIMER ($n$=30; $M$=0.77; $SD$=1.67) and those at SKETCH ($n$=14; $M$=0.21; $SD$=1.61). We note that the experienced students attending TOUCH dismissed having any *Behavioral Intention* to use programming ($n$=10; $M$=-2.03; $SD$=0.90), which perhaps relates to the curricular unit's placement at the end of their course of study.

* The three participants at GRAPHIC that claimed a prior familiarity with programming had widely different perceptions of voluntariness ($M$=-0.33; $SD$=3.06).

✦ SELF-REPORTED USE We also found significant and strong associations between all participants' specific curricular unit and their self-reported programming use frequency* and intensity.† Among the participants that reported to be learning to code for the first time ($n$=84), their reported frequency ($F$=4.24; $p$=.002; $\eta^2$=.22) and intensity ($F$=4.78; $p$=.001; $\eta^2$=.24) of use evinced clear relationships with the specific case study, yet among the participants with a prior familiarity with code ($n$=90), reported frequency ($F$=11.88; $p$<.001; $\eta^2$=.42) and intensity ($F$=11.12; $p$<.001; $\eta^2$=.41) of use evinced even greater dependence on their curricular unit. We present a comparison between studies in the following table:

***Table 7.8.2.*** *Comparison of students' programming use in different case studies.*

| | | | | *Usage frequency* (0 – 5) | | *Usage intensity* (1 – 7) | |
|---|---|---|---|---|---|---|---|
| | *Case study* | *week* | *n* | Mean | St. dev. | Mean | St. dev. |
| *First-time learners* | PRIMER | 4 | 40 | 2.25 | 0.87 | 3.87 | 1.61 |
| | SKETCH | 2 | 16 | 1.43 | 0.65 | 2.79 | 1.31 |
| | WEAVE | 3 | 21 | 2.60 | 0.68 | 4.95 | 1.43 |
| *Familiar with programming* | PRIMER | 4 | 30 | 2.47 | 0.86 | 3.93 | 1.46 |
| | SKETCH | 2 | 14 | 1.54 | 0.88 | 2.08 | 1.17 |
| | WEAVE | 3 | 29 | 3.34 | 1.05 | 4.96 | 1.32 |
| | TOUCH | 6 | 10 | 1.70 | 0.48 | 2.60 | 1.08 |

As we described in §4.6.2, students at SKETCH that completed both survey applications reported significantly higher computer programming use frequency ($n$=17; $M$=2.24; $SD$=0.66) and intensity ($M$=4.18; $SD$=1.51) by the end of their semester,‡ but nonetheless lower than what their colleagues at WEAVE reported in their third week.

Nonetheless, layered analyses did not depart in any substantial way from what we presented in Table 7.8.2. Participants attending WEAVE reported higher frequency and intensity of computer programming use even though they completed their survey earlier in the semester than their colleagues at TOUCH, who reported much lesser use of computer programming, even though they were in the midst of developing practical work at that time.

* Programming frequency had been measured as a 6-point scale coded from 0 (never) and 1 (less than weekly) to 5 (multiple times a day). See 5.4.1.

† Programming intensity had been measured as a 7-point scale coded from 1 (as little as possible) to 7 (as much as possible). See 5.4.1.

‡ We found no significant changes in students' computer programming use in the other studies where we applied a follow-up questionnaire, as described in §2.6, §5.5.2, and §7.5.

## 8.6 *Projects and learning outcomes*

Students at PRIMER, SKETCH, and TOUCH were either asked or allowed to develop group projects. Though students at SKETCH were also asked to submit individual exercises with a substantial impact on their grades, as noted by Professor Edwin, group projects allowed many to hide their difficulties or lack of interest in programming (see §4.7), moreover if there are no further mandatory curricular units about programming in their courses' curricula. Still, we saw students attending PRIMER were likely to prefer to develop individual works unless the professor leading their class explicitly called for group projects. It is possible the freedom those students were granted regarding the theme of their work motivated students' individual expression.

The school hosting TOUCH organized a student showcase exhibition halfway through the semester, and Professor Ivan had required students to participate with a milestone of their ongoing projects. But for FLOW's canceled recital, we find it noteworthy no other unit had committed to a public presentation of students' projects.

The final presentations we were able to attend took place behind closed doors. We witnessed the presentations of 70 projects developed at PRIMER, SKETCH, and TOUCH (37 by individual students and 33 by groups), as described in §2.5, §4.5, and §6.4. SKETCH and TOUCH's last sessions were dedicated to that purpose, whereas PRIMER's were scheduled as an exam and took place after the end of classes. In addition, Professor Douglas provided us with the source code for 30 individual exercises submitted by GRAPHIC's students (see §3.4). We were unable to see the visual artifacts produced for the first SKETCH exercises, as well as the websites developed at WEAVE. We were also unable to listen to the works composed by FLOW students, as their public recital was canceled, and their submission was postponed until after the end of classes.

⁂

We found some salient aspects to students' presentations and projects which we find worth comparing:

✦ PRESENTATION APPROACHES Scheduling did not reflect on the nature of the presentations. As described in §2.5, PRIMER's presentations were confirmatory, as students had been required to submit their finalized work a few days before. The three professors had already analyzed submissions and thus paced around the studio to where each workgroup or individual author was sitting, wherein they asked to see the final program running once more, made a few questions, conferred, and issued some remarks. In contrast, SKETCH (see §4.5) and TOUCH's (§6.4) professors asked their students to prepare formal presentations before the entire class. Students at TOUCH made simple presentations in which they merely held their hardware controllers for the class to see and then ran their project on a laptop connected to the classroom's projection display. In contrast, SKETCH's students prepared specific materials for presentation (e.g., slide decks about the inspiration for their work and its development, and in a few cases, videos and images demonstrating their work's practical application). Moreover, the fluidity and confidence evinced by some suggest that SKETCH's students took the time to rehearse their presentations.

✦ PROJECT QUALITY Though outliers were likely among a large number of students, the researcher was nonetheless impressed by the complexity and attention to detail of a few projects presented at PRIMER. Given the difficult conditions, it appears many students and groups were highly motivated to learn and work autonomously. Professors Alan and Bill praised the creativity of the projects, though Carl was more circumspect, noting most projects fit in a few types and presented little originality (see §2.7). Professor Carl also noted that female students were likelier to dedicate themselves further to ensuring the quality of their projects' visual aesthetics.

Professor Frank also praised the creativity of the projects developed at WEAVE (§5.6), whereas Edwin praised the visual quality of the work presented at SKETCH (§4.7). Several of the projects presented at TOUCH did not meet the professor's requirements, especially for the intermediate public exhibition organized by the school. The researcher was disappointed by the quality of those projects, both in relation to their aesthetics and to the sophistication of their programming, as the students were finalists and had almost the entire semester for development. In addition, the tutoring conditions, in terms of class size and session length, had likely been the best we witnessed across our studies. As reported in §6.4, it is possible most of TOUCH's students only worked on their projects during the sessions, given their commitments to other curricular units.

✦ STUDENTS' CODE Several students submitted GRAPHIC exercises in which they made little changes to code previously shared by Professor Douglas himself. Indeed, Douglas, Carl, and Frank complained about students taking ready-made solutions from online sources (see §3.5, §2.7, and §5.6), and Professor Edwin also complained extensively about students that copied his own code in their work (§4.7). Only Professor Ivan explicitly denied that any students had reused their projects' code (§6.6). As described in §3.4, we also saw most of Douglas' students had acquired his habit of not commenting their code.

✦ OUTCOMES Despite any relative difficulties, we gathered that most—if not all—of the students involved in the presentations we witnessed obtained a passing grade from their respective professors. We only witnessed one situation in which a TOUCH student told her professor she would rather take an exam. Indeed, all professors we interviewed believed their students had met the objectives they had set out for their curricular units; still, Joel noted FLOW's had not been ambitious (see §7.6), and Douglas (GRAPHIC) and Frank (WEAVE) noted most students had not progressed beyond a minimal set of objectives (§3.5, §5.6).

Professors Alan and Carl emphasized that most students at PRIMER had overcome their initial lack of motivation to learn computer programming, and Bill noted the good rapport he had established with students throughout the semester (§2.7). Professor Edwin told us nearly all of the SKETCH students that had presented their work had obtained a passing grade, though a few had benefitted from participating in group projects with effective colleagues (§4.7). However, he also admitted about a quarter of his students had given up on the unit, and he was concerned about having many repeating students in the following year.

Professor Edwin also noted his concern about students' knowledge consolidation. Indeed, Professor Frank told us he faced precisely such a lack of consolidated knowledge at WEAVE from students who had been expected to bring a prior familiarity with code from the previous year. This emphasizes the importance of regular contact and practice; still, persuading students might have been a challenge for professors. At the end of the term, Professor Joel believed he had been unable to convince FLOW's students of the utility of computer programming (§7.6).

✦ GENDER DIFFERENCES Following the preliminary surveys' suggestion of a relationship between students' gender and their acceptance of computer programming (see **6**.2), we asked the professors if they found any such differences in their classes and whether they were consequential to students' learning outcomes. Most denied being aware of any differences. Professor Ivan told us TOUCH's learning outcomes tended to be equally distributed over time, and in the particular instance of the unit we observed, female students had obtained the best grades (see §6.6); in contrast, Professor Frank told us male students tended to obtain better grades at WEAVE (§5.6).

Professor Edwin gave us clues for further discussion, as he had noticed that female students at SKETCH tended to be more modest in their self-appreciation and were less likely to intervene in class, though they were more dedicated and likely to obtain final outcomes as good or better than their male colleagues (§4.7). Professor Carl admitted that an expectation of greater interest in technology from male students testified about his own biases. Still, he told us the female students at PRIMER had been as interested and obtained the same grades as their male colleagues, though he noted a contrast with the other institutions where he had taught (§2.7).

## 8.7 *Limitations faced across the studies*

We faced distinct limitations to our studies in each particular case, which we described in subsections §2.8, §3.6, §4.8, §5.7, §6.7, and §7.7. Most limitations resulted from the varying delays we faced in obtaining the necessary permissions to carry out the studies, which subsequently constrained our ability to schedule visits to observe sessions. We also faced some access limitations—e.g., to the online forums used for communication between students and professors at PRIMER about their projects, and to the observation of practical sessions led by WEAVE lead professor's assistants. In addition, we also faced a relatively widespread indifference by students towards completing our surveys, which was critical at the end of the semesters and thus impeded all but the most superficial longitudinal analyses.

※

Despite the aforementioned limitations, we are confident the reader can regard these findings, which we will now discuss in the following chapter, as encouraging further research toward more systematic monitoring of the teaching and learning of computer programming within arts and design education.

*(blank page)*

*(blank page)*

PART FOUR

# *DISCUSSION (FINDINGS);*

*(blank page)*

# 8

# *Discussion of our findings*

★ SYNOPSIS We discuss the findings reported in the two previous chapters, which we connect with the scholarship about the multiple convergences between computing, higher education, and the arts, described in chapters **2** and **3**, and the specific Portuguese context, described in chapter **4**. In section §1, we reiterate our inquiry's goals and design, and consider methodological implications of our findings. In section §2, we examine how computer programming was integrated into the curricula across our studies, considering reasons, objectives, and conditions. In section §3, we discuss the teaching strategies and how they relate to the research about computing and arts and design pedagogies. Finally, in section §4, we focus on the students and discuss their familiarity with computing, their commitment and difficulties, and their learning outcomes.

## 1 METHODOLOGICAL SUMMARY AND IMPLICATIONS

N. L. Gage (1991) investigated whether social and educational research yielded 'truisms' *"that could have been predicted without doing the research."* Indeed, understanding that students having a prior familiarity with a technology are more willing to accept and use it, or that giving students more time to learn and practice a subject will generate better outcomes may appear commonsensical. Yet, the devil is in the details, or as the author states, *"to enhance the truism with the specifics that make it have value for theory and practice, the research does become necessary"* (Gage, 1991). For instance, some of our observations upended the truism that smaller classes learn best and present the best work. Yet, we do not mean to imply we 'debunked' a truism. On the contrary, it was far more likely that other factors, whose association with learning outcomes could also be seen as commonsensical if discussed in isolation—e.g., courses' distinct curricula or students' seniority—came to the fore with a more vigorous intensity within the specific contexts of our observations. As Gage (1991) stated, *"people tend to regard as obvious almost any reasonable statement made about human behavior."* We do have reasonable statements to offer in the coming discussion. Still, their application is not inevitably linear, nor are generalizable without contradictions. If we are permitted to jump ahead to one commonsensical conclusion, it shall be this one: more research will be needed.

## 1.1 *Summary of research goals, design, and methods*

As we had stated in chapter **5**, our main research objective was to find out how computer programming is taught and learned within Portuguese public undergraduate arts and design courses. We had set multiple initial goals relating to that main objective:

- To identify the factors that determine arts and design students and professors' acceptance of computer programming.
- To discover how curricula and professors interact with those factors.
- To determine whether and how do professors adapt their teaching strategies to make of computer programming salient to arts and design students.
- To discover how professors value teaching these particular students to code.
- To contextualize local practices within our review of international developments.

In order to meet these goals, we devised a two-phase research design (see **5**.3). In a preliminary stage, we conducted a quantitative survey of students attending public arts and design undergraduate courses that included at least an introduction to computer programming in their curricula and also conducted a simultaneous survey of professors with teaching assignments at those same courses. In chapter **6**, we reported the findings from our preliminary surveys, which aggregated the responses of survey applications in the Spring and in the Fall of 2018; in total, we obtained 270 valid responses from students at 27 different courses and from 80 professors at 18 different institutions.

The preliminary surveys informed the second stage of our inquiry, in which we conducted a collective case study of six distinct curricular units presenting computer programming to the students attending different arts and design licentiates in distinct institutions. Within each case, we employed mixed data gathering methods (see **5**.5): we gathered documents, observed lectures and studio classes, examined online resources, witnessed presentations and exhibitions of students' work, interviewed the professors, and obtained students' responses to our quantitative survey instrument. Our research objectives had also evolved, and we were particularly attuned to how the professors interacted with students and shaped learning strategies and commitment. In chapter **7**, we reported the findings from the six cases, which all took place during the 2018/2019 academic year.

## 1.2 *Challenges and implications*

As recommended by Flick (2004), our research design and procedures were underpinned by our pragmatic assessment of available resources and an understanding of the external constraints placed on the gatekeepers (e.g., ensuring compliance with data protection regulations) on whom we had to rely as to conduct our research. Still, we believe some of the difficulties and limitations we faced constitute findings worthy of discussion.

### 1.2.1 INSTITUTIONAL RESISTANCES

As we mentioned in 5.4.2, out of 165 emails we sent requesting support for conducting our preliminary surveys, 132 were not replied. We believe our messages were clear about who we were and about the objectives of our inquiry. Moreover, we believe our research was—and remains—in the interest of the recipients, as all held academic coordination responsibilities in courses we had determined to include computer programming as a part of their curricula (see 4.2). Our wish here is not to criticize the recipients that did not reply to our messages, which were, after all, unsolicited. Nonetheless, we believe the ratio of unanswered emails was significant.

The researcher had no prior familiarity with most higher education institutions he contacted; moreover, the studies that took place largely involved institutions which the researcher only knew on personal and anonymous capacity (e.g., visiting to attend a talk or an exhibit). Nonetheless, given the nature of the inquiry and the researcher's own teaching background, there was an undeniable insideness that can lead to resistance or fear of scrutiny (cf. Sikes & Potts, 2008).*

Nonetheless, institutional opacity is likely a marginal reason for avoidance of our queries. The reasons are perhaps more prosaic: as we have seen, academic staff at higher arts education institutions often complain about a high workload and crowded classes due to a lack of investment in additional staff and better working conditions (see also Oliveira, 2017). Little time is left after the competing demands of teaching, research, administrative work, and, in some cases, a personal practice. In this context, it is pretty understandable that aiding and hosting outside researchers takes a low priority.

We regard some of the difficulties we faced as persisting symptoms of the managerial turn that took place alongside the implementation of the Bologna Process and institutions' continued short-term outlook (cf. Nóvoa, 2018; Shreeve, 2011). In that case, we emphasize the irony in how a culture of transparency and measurable academic productivity might impede research about higher education practices.

### 1.2.2 QUANTITATIVE FINDINGS IN THE CONTEXT OF UTAUT SCHOLARSHIP

Our quantitative research instrument, which we presented in 5.4 and was based on the Unified Theory of Acceptance and Use of Technology (Venkatesh, Morris, Davis, & Davis, 2003), though very useful at the preliminary stage of our research, had a limited impact on our case study research—in some cases due to the small classes, in others due to inadequate survey participation rates. Moreover, as UTAUT measurements are variant on participants' previous experience with technology (Parameswaran, Kishore, & Li, 2015) and we thus segmented survey responses, most quantitative analyses in our case studies were merely tentative. Under these conditions, and given the model's reduced explanatory power (cf. S. A. Brown, Dennis, & Venkatesh, 2010), we found it a weak coadjuvant to qualitative observation.

* Speaking more of the researcher's love for police dramas than the actual situations, while conducting the case studies he sometimes wondered whether he was seen as akin to an 'internal affairs' agent.

⁂

Our preliminary surveys of students and professors, reported in chapter **6**, yielded the relationships predicted by UTAUT (Venkatesh et al., 2003) between participants' *Behavioral Intention* to use computer programming in the future and their *Performance Expectancy, Effort Expectancy*, and *Subjective Norm* (standing for *Social Influence*, as described in **5**.4.3). In addition, participants' *Attitude Toward Use* and *Self-Efficacy* was also found to be related to their *Intention*, matching the literature that reexamines those associations after Venkatesh and colleagues' original dismissal (Dwivedi, Rana, Jeyaraj, Clement, & Williams, 2017; Williams, Rana, & Dwivedi, 2015). Our preliminary studies' findings are thus compatible with Kreijns and colleagues' proposal holding *Attitude*, *Subjective Norm*, and *Self-Efficacy* as the major determinants of intention (Kreijns, Vermeulen, Kirschner, & Buuren, 2013).

Still, our case studies' surveys of students did not replicate that finding. Though the groups of participants under analysis were all rather small ($10 < n < 40$), *Subjective Norm* and *Self-Efficacy* did not yield significant associations with *Behavioral Intention* across all case studies. *Attitude Toward Use* did, hence proving to be a particularly relevant measurement in the context of learning and using computer programming. This also matched Dwivedi and colleagues' (2017) meta-analysis, which held *Attitude* as a more direct determinant of technology use than either *Performance Expectancy* or *Effort Expectancy*. Nonetheless, but among the small sample of ten students having a prior familiarity with programming and participating in the TOUCH case study (see **7**.6.5), *Performance Expectancy* also yielded significant relationships with *Intention* across all segments of participants in our quantitative studies, and thus our findings agreed with Williams and colleagues' review (2015) maintaining *Performance Expectancy* as the most reliable indicator of intention to use a technology.

# 2 PROGRAMMING WITHIN ARTS AND DESIGN CURRICULA

Knochel and Patton (2015) classified learning to code within the arts studio as still a rare occurrence. Yet, we found such a contact was relatively common within arts and design undergraduate courses offered at Portuguese public higher education institutions: as we reviewed in chapter **4**, 42 undergraduate courses, which were attended by almost 7,000 students in 2018/2019, included at least one unit about computer programming. As we saw, many of the curricular changes that were made to that end were in place by 2010. Still, Knochel and Patton's point may hold if one considers the extensiveness of students' contact with computer programming: fifteen of the courses we reviewed included one single mandatory unit introducing the subject, and six others only included elective curricular units.

⁂

In the following subsections, we will discuss how the units we scrutinized in our case studies were integrated within their courses' wider curricula. In §2.1, we will discuss how the professors reasoned about the importance of programming within arts and design courses. In §2.2, we will consider the specific learning objectives of the curricular units we scrutinized. In §2.3, we will briefly examine the role of the studio environment. Then, in §2.4, we will also briefly discuss the programming languages chosen for teaching.

## 2.1 *Rationales for teaching programming*

> The purpose of education has at least two irreconcilable possibilities: one, that man should be educated to become what he is; the other, that he should be educated to become what he is not. (Read, 1943/1974, p. 220)

One of the professors we interviewed within the context of our case studies, responsible for the curricular unit we referred to as FLOW, echoed the perspective of Herbert Read and posited that nurturing individuality should be one of the overarching objectives of arts education (see **7**.7.6). As we reviewed in chapter **3**, similar concerns preside over discussions of the relationship between arts and new technologies: are digital tools a homogenizing force (Murphy, 2015) that guides the artists' hands according to the tools' own affordances (Ettinger, 1988)? Or is computing that Promethean gift (cf. Reas & Fry, 2018) which will power the engines of creation, a medium for practicing individual creativity (McCormack & D'Inverno, 2012)?

Professors' statements across our studies, both in the preliminary survey (see **6**.3.4) and during the case studies' interviews (see **7**.8.1), reflected Guzdial's (2019) two main categories of motivations to teaching computer programming in broader disciplinary contexts. Some emphasized programming's utility as a tool for doing things as they emphasized the future employability benefits of learning to code and placed the subject as one among various digital competencies—e.g., as suggested by the DigComp 2.0 Framework (Vuorikari, Punie, Carretero, & Van Den Brande, 2016). Others emphasized computer programming's utility as a tool to think with, holding the subject as a more

fundamental 'uppercase Literacy' (Leu, Kinzer, Coiro, Castek, & Henry, 2017). Moreover, some referred to the notion of computational thinking (Wing, 2006) by emphasizing how students learn problem-solving strategies, whereas other participants noted how craft yields broader logical thinking skills, in closer agreement with Mateas' (2005) and Vee's (2013) conceptions of procedural literacy and computational literacy, respectively. Some of our interviewees explicitly valued how computing may underpin a desirable interdisciplinarity between the arts and the sciences, which agrees with Blessinger and Watts' (2017) view regarding the learning of creativity skills.

⁂

Looking toward the future of computing within arts education, two of our interviewees believed in a return to the cultivation of individual skill as a central tenet and noted how persistent exploration and interrogation of digital interfaces might relate to that goal. The professor responsible for the curricular unit we referred to as PRIMER commented how poor computer interfaces are a limitation for arts education (see 7.2.7), thus echoing the comments of McCormack and D'Inverno's (2012) about the limitations of mouse and keyboard interactions, or Wilks, Cutcher, and Wilks' (2012) about the poor ergonomics of laptops.

## 2.2 *Learning objectives*

David Kember and Carmel McNaught (2007) interviewed 62 award-winning professors across multiple disciplines at Hong Kong and Australian universities and compiled a set of teaching recommendations and strategies. Chief among them, and related to a view of the professor as a facilitator, which we will discuss in the next section, is the notion that professors ought to mind that *"the first skill or competence needed by students is an ability to learn on their own,"* as *"on graduation they need to be lifelong or self-managed learners"* (Kember & McNaught, 2007, p. 18). Cabeleira (2018) criticizes that having 'teaching to learn' as a central educational objective may leave students to learn over a less robust body of disciplinary knowledge. Nonetheless, either mentioning so in their curricular unit plans or telling us so during our interviews, the professors across our studies did emphasize the need to develop students' autonomous learning skills. Indeed, the professor leading SKETCH noted his syllabus comprised the breadth of contents he saw as necessary for students to *'learn to learn*,' as the parent course would offer no additional contact with programming, hence meeting Robins (2019) call for *"focus at least in part on the enabling step of functioning as an effective novice"* (p. 356).

Moreover, stimulating students' autonomy requires that professors promote the transferability of the knowledge they impart; as Robins, Margulieux, and Robinson (2019) state, *"the best way to help students transfer their knowledge is to teach them how to transfer their knowledge."* Still, Kember and McNaught (2007) also believe professors ought to prioritize depth over breadth in their lessons. The authors state, *"what is important is how much is learnt rather than how much is covered"* (p. 77). Though most professors in our case studies seemed to agree with these tenets, some also saw their goal of leading students to the point of being able to learn autonomously as implying a

need to cover a broad syllabus for the allotted session time, as the professor responsible for SKETCH told us in his interview (see 7.4.7).

Though referring to the education in computer programming received by pre-college students in the United States, Blikstein and Moghadam (2019) raised concerns about superficiality, noting the objective of fluency is rarely met. Citing Stager (2017), the authors note, *"the epistemological benefit of programming computers comes from long intense thinking."* Going back to Mateas' (2005) statement about the desirability of a certain level of difficulty in introductory computer programming units, it may be the case that a more demanding syllabus or evaluation methodology will prove beneficial toward assuring students' future autonomy.

Some of the professors leading the curricular units we scrutinized also sought to develop students' transversal competencies in teamwork and communication, meeting the recommendation of Kember and McNaught (2007, p. 19). This goal was especially evident in the units that required students to develop group projects.

## 2.3 *The studio*

At the turn of the millennium, looking toward the coming wave of mediation, a few higher education thinkers nonetheless stressed the importance of social interactions in a shared physical space (cf. Agre, 2000; Noll, 2000). This is doubly true in arts and design education, where the studio is the locus of experiential learning (Shreeve, Sims, & Trowler, 2010). Professors across our case studies understood this; during his interview, the professor responsible for the PRIMER curricular unit explicitly mentioned the importance of the interactions between students within the studio space (see 7.2.7). Studio-based learning has also been found beneficial in computer science education (cf. Carter & Hundhausen, 2011).

Nonetheless, Shreeve (2011) sees the arts studio as under the threat of managerial preoccupations in higher education. Though not the case across all our case studies, we witnessed and heard complaints about the crowded studios where students congregated for an insufficient time as to discuss their work in progress amongst themselves and with the professors. Shreeve (2011) also noted a trend toward replacing specialized equipment by all-purpose computers. Almost a decade later, we witnessed an inversion of this trend. As most studios' computers sat abandoned as students preferred to use their own devices, we witnessed a return to investment on specialized equipment (e.g., for tangible computing or virtual reality), which our interviewees confirmed was to be continued.

## 2.4 *Programming language choice*

Given their distinct affordances, the choice of which programming languages to cover was central (but not entirely sufficient) to make the subject salient to arts and design students. Our case studies evinced the distinct approaches toward language choice identified by Parker and colleagues (Parker, Beard, & Davey, 2019). Curricular units with a greater disciplinary specialization—WEAVE (web design, see 7.5) and FLOW (music, see 7.7)—evinced concerns with industry standards or employability, whereas most of

the remaining units opted for either Processing or P5.js, two languages explicitly aimed at novices and with little demand by employers. In this context, one of our case studies—PRIMER (7.2)—was interesting was students were given a choice to work with either Processing or the Unity game creation system (and associated C# scripting language), an in-demand industry standard.

Moreover, as PRIMER students had the liberty to choose their projects, the case study fully demonstrated how each programming language has its own affordances, as projects made in Processing had rather different characteristics to those made in Unity (7.2.5). Past the adequacy of Unity for game projects, we find it worth emphasizing how Unity commanded group projects while Processing did not. Some research found Unity and C# are harder to learn and not as adequate for the first contact with programming, yet its conduciveness to interdisciplinary group projects are rather attractive to students (Comber, Mayer, Motschnig, & Haselberger, 2019).

## 3 PROFESSORS' ROLE AND TEACHING STRATEGIES

Though we met a few professors with an academic background in computer science or computer engineering in our studies, many were artists and designers whose knowledge of computer programming had been acquired autonomously. One such professor we interviewed posed the question, *"how are you going to teach something you learned on your own?"* (see 7.7.6). Perhaps not that much has changed since Linda Ettinger (1988) quoted a verbal presentation by Loveless,* who stated, *"nobody really knows how to teach computers yet, although a lot of people do it."*

Still, professors' self-education goes beyond the specific subject of programming. Most participants in Kember and McNaught's (2007) study of award-winning university professors had no formal training in teaching; the authors added that phenomenon was widespread across disciplines and different countries. Kember and McNaught (2007) explain that *"at the start of their careers teaching tended to be didactic and concentrated on covering content"* (p. 124), though in time professors may progress from *"a teacher-centred/content-oriented belief, to a student-centred/learning-oriented one, consistent with the facilitative beliefs about teaching"* (p. 125). This explanation is consistent with our observations.

⁂

In the following subsections, we will discuss the approaches to teaching we observed across our case studies. In §3.1, we will explore how the professors envisioned and took their role as educators. In §3.2, we will examine the professors' pedagogical strategies in the classroom and in the studio. Finally, in §3.3, we will briefly discuss the professors' usage of online platforms and resources.

### 3.1 *Roles and duties*

Criticizing arts professors' lack of command over their medium, unlike the academic painters of yore, Thierry du Duve (1994) acknowledged a reality which is similar to what we witnessed in our case studies:

> They are facilitators, communicators, information providers, psychologists who must steer their patients through their desires. And their students are no longer apprentices; they are the teachers' passengers. (de Duve, 1994)

Though departing from the tradition of stricter apprenticeship which borne it, the role of the educator as facilitator and negotiator meets the signature pedagogies of arts education identified by Shreeve and colleagues (2010). Moreover, the perspective of the teacher as *"a facilitator of learning"* is precisely at the center of Kember and McNaught's recommendations (2007, p. 22). Though the cultivation of students' virtuosity, as defended by two of our interviewees, might have a place in the future of arts education,

* Loveless, R. (1988). The impact of new media on learning. Presentation at the National Art Education Association Convention, Los Angeles, CA, April 10.

the institutional framework of higher education does not favor a return to a paradigm based on apprenticeship. In this context, professors' command over pedagogy is possibly more salient to educational practice than a command over the medium of their specific artistic discipline; when that medium is computing, moreover, command is elusive.

⁂

Several of the professors we interviewed mentioned difficulties in balancing teaching, research, and other professional obligations. Indeed, Kember and McNaught (2007) acknowledge that professors frequently feel they ought to sacrifice the quality of their teaching in order to dedicate to research, insofar as the latter is seen as a greater factor in career advancement (p. 127). Still, that does not need to be the case. The authors note research can inform teaching, teaching can inform research, and the teaching itself can be a subject of research (pp. 128-129); indeed, the professor leading the GRAPHIC unit reflected a similar opinion (see 7.3.5). Still, training interventions may help professors' progress to the aforementioned student-centric and learning-oriented approach.

### 3.1.1 ENCOURAGING DEEP LEARNING

In tandem with a facilitative approach to teaching, professors are called to encouraging students to commit to a deep approach to learning.* Indeed, most professors across our studies invoked active learning pedagogies to demand involvement from students and encourage deep learning (Falkner & Sheard, 2019; Shephard, 2019), as we shall discuss in the next subsection. Still, the professors leading lectures in the GRAPHIC (see 7.3.3) and WEAVE (7.5.3) units were seen to undertake the aforementioned didactic approaches focused on covering the syllabus, which often failed to capture students' attention.

Choosing suitable evaluation methods may also encourage deep learning. Kember and McNaught (2007) call for 'authentic tasks' related to professional practice, which echoes how experiential learning is a signature pedagogy of arts education (Shreeve et al., 2010). As we shall discuss in the next subsection, most professors we observed sought to place students in the shoes of practitioners: students attending PRIMER or FLOW, part of two courses in which the curricula emphasize artistic exploration and skill, were tasked with open projects, whereas students attending the other courses more pointedly aimed at the needs of the job market were assigned projects with precise briefings.

### 3.1.2 ESTABLISHING RAPPORT

Kember and McNaught (2007) emphasize the importance of establishing a rapport with students, noting *"the initiative needs to come from the teacher; students are normally willing to respond, but it is hard for them to take the first step"* (p. 43). Artistic work requires the cultivation of a climate of trust. In the small class environment of the FLOW case study, we found such rapport had already been naturally established due to a prior familiarity between students and the professors.

* Though the terms are somewhat self-explanatory, we will discuss the distinction between *deep learning* and *surface learning* (Marton & Säljö, 1976) in more detail in the next section.

Though large classes are more demanding, two of the professors involved in the PRIMER study indeed mentioned the importance of getting to know students and their individual preferences (see 7.2.7), as recommended by Kember and McNaught (2007, pp. 45–46). In addition, the professor leading SKETCH (7.4) granted a specific office hours schedule for that unit; in his interview, he mentioned how several students benefitted from the additional time (7.4.7). Yet, we do note few professors held face-to-face office hours; indeed, most institutions do not assign that time to visiting professors.

### 3.1.3 RESPONSIBILITY AND DEMAND

Even though all professors that participated in our case studies either explicitly stated or implied that they took that responsibility, some were nevertheless likely to partially blame students for their lack of interest in learning to code, especially when referring to those who had given up on the curricular unit. Yet, according to Kember and McNaught (2007), the most effective professors ought to take full responsibility for motivating students, regardless of their discipline's difficulty or specialty (p. 47).

The authors also recommend that professors hold students to high expectations. In contrast, most professors we interviewed admitted they expected a low commitment from most students. The professors were likely aware that *"excessive workload tends to promote a surface approach to learning as students cut corners to cope with overwhelming demands"* (p. 48), yet they found it challenging to conciliate their own evaluation demands with those of other curricular units. In one particular instance, we believe the evaluation requirements of the unit we referred to as GRAPHIC (see 7.3) were undemanding, perhaps owing to being an elective, and indeed we witnessed a superficial commitment from most students. Kember and McNaught (2007) admit some students choose to attend the curricular units they envision as less demanding, though they note that most dislike 'flat' units and appreciate a challenge (p. 49).

## 3.2 *Teaching strategies*

Most of the teaching recommendations found in literature can be related, when not explicitly underpinned, to Cognitive Load Theory (Chandler & Sweller, 1991; Paas, Renkl, & Sweller, 2003; Sweller, 1994). CLT posits the human brain has limited working (i.e., short-term) memory and processing capacities. Though depending on learners' previous knowledge, materials have their own levels of complexity (their *intrinsic load*) and of cognitive demand on learners (presenting *germane load*); moreover, communication presents its own cognitive demands that are unrelated to the material, hence presenting a source of *extrinsic load*.

Following the tenets of the Cognitive Load Theory, teaching strategies ought to minimize extrinsic load as well as intrinsic load. Setting subgoals and presenting the material in smaller units may help with the latter. The relationship between students' previous knowledge and the intrinsic load of the material implies that professors ought to closely monitor how they apportion the content they present (Paas et al., 2003). This offers an argument for professors keeping pace with the students evincing the greater difficulties with programming or, in the alternative, for providing a 'recovery stream' to

those students, as Robins (2019) proposes. Moreover, we saw that the conditions of the classes were often a source of extrinsic cognitive load—e.g., those students that had difficulties reading the code presented on the classrooms' projectors were probably not as able to understand the code itself. As suggested by Petre (1995), visual programming languages such as Max—taught in the FLOW curricular unit (see 7.7)—hold particularly challenging cognitive demands for novices.

Some of the professors we interviewed regarded the first few lessons as crucial, holding that an early demonstration of the usefulness of computer programming would capture their students' attention and commitment. Yet, we believe professors sometimes overlooked students' difficulties in the earlier lectures we witnessed. Robins (2019) notes early difficulties accumulate, and *"once negative momentum is established, it is very hard to overcome."* As one student in the FLOW study stated, *"the train did not stop"* (7.7.3).

### 3.2.1 LECTURES AND LIVE CODING DEMONSTRATIONS

> Even in courses which use student-centred approaches like problem-based learning, a certain body of knowledge has to be mastered. Expecting students to discover all this knowledge for themselves can be time-consuming, inefficient and tedious. (Kember & McNaught, 2007, p. 67)

Some of the professors we interviewed referred to the necessity to impart foundational knowledge so that students could overcome the 'nomenclature problem' in order to become able to research about computer programming (as the professor leading GRAPHIC stated it—see 7.3.3). Yet, Kember and McNaught (2007) note most students have difficulties concentrating in lectures for more than one hour and recommend that professors periodically interrupt their exposition with activities and challenges (p. 70). Indeed, the professor responsible for FLOW (7.7.3) interrupted his lectures and launched discussions about the relevancy of programming in musical practice. The professor leading SKETCH proposed brief practical tasks (7.4.3). In contrast, a lack of interruptions may have contributed to the flatness we witnessed in WEAVE's lectures (see 7.5.3).

Professors across our studies relied on live coding demonstrations of prepared or *worked examples*, which were found to be more effective at the early stages of the learning process (Paas et al., 2003; Renkl, 2017; Rubin, 2013); indeed, many lectures consisted of little else. Kember and McNaught (2007) agree with the extensive use of demonstrations in lectures; one of their own interviewees stated, *"I believe the deadliest sin in teaching is to read a paper or read a lecture"* (Kember & McNaught, 2007, p. 69). Yet, we would venture some programming demonstrations we witnessed were not that different from a 'reading' lecture and met like engagement from the students.

Moreover, worked examples allow professors to focus on programming *strategies*, posited to be more important for novices than pure knowledge (Robins, Rountree, & Rountree, 2003). That examples include granular, meaningful, and explicit *subgoals* are considered essential to promote the transferability of knowledge (N. C. C. Brown & Wilson, 2018; Robins et al., 2019; Sweller, 1994). Many of the live coding demonstrations we witnessed in classes were given in an iterative way—for instance, when the professor leading the SKETCH study (see 7.4.3) first showed how to encapsulate a robot-drawing program within a function, then how to use function arguments to define the robot's

placement on the screen, then how to call that function multiple times in order to draw numerous robots. Indeed, Daniel Shiffman (2008) advises a *"philosophy of incremental development"* (p. xv) to learning Processing. Still, we felt that in some cases, such as in GRAPHIC's lectures (7.3.3), the professors did not break their examples in enough detail to allow transferability of small solutions to new problems.

Robins and colleagues (2019) recommend setting analogical tasks to promote transferability. Given the extent to which most of our studies focused on curricular units taught to first-time learners, there were not many opportunities to apply similar solutions in distinct situations. Indeed, Robins and colleagues (2019) state that novices are more likely to benefit from explicit subgoals, whereas the setting of analogical solutions is more beneficial for students with prior knowledge. Still, the professors responsible for SKETCH and FLOW did interrogate novice students in the recurring use of loops and conditions across different examples (7.4.3, 7.7.3); in contrast, we noted how the examples brought by the professor leading GRAPHIC missed some opportunities to establish analogous solutions.

### 3.2.2 CONSOLIDATING KNOWLEDGE

Robins and colleagues (2019) note how, when left unexercised, computer programming knowledge decays over time, and teachers frequently complain their students forgot what they had learned either in earlier sessions or in earlier curricular units. Indeed, professors across our studies issued similar complaints, despite an awareness of how working on projects would lead students to retain their knowledge better (Krajcik & Shin, 2014). Robins and colleagues (2019) note that research supports the notion that spaced practice across sessions facilitates comprehension, as does alternating between studying examples and working on problem-solving exercises.

Indeed, some of the professors we interviewed (e.g., in the PRIMER and GRAPHIC studies; see 7.2.7, 7.3.5) wished for shorter but more regular sessions in order to mitigate knowledge decay between sessions, moreover when students were seen not to work at home. Though the professors that participated in our case studies sometimes asked students to complete short exercises within the classroom or at home, all these exercises were either to be started from nothing or to iterate previous exercises. The professors did not task students with activities often recommended in computing education literature to mitigate the initial problems posed by programming syntax (N. C. C. Brown & Wilson, 2018; Robins et al., 2019), such as code completion exercises (van Merriënboer, 1990) or 'Parson's problems' (Parsons & Haden, 2006). Moreover, students evinced little commitment to in-class activities that were not related to larger project demands.

We note that the evaluation methodologies of the units we scrutinized assigned little or no weight to in-class or homework exercises; their completion was often implied as part of the 'participation' mark, amounting to a small percentage of the final grade, in comparison with the 'bigger' projects. Kember and McNaught (2007) noted, *"activities with no associated marks are likely to be left undone or skipped over, unless perhaps they are seen as precursors to assessed tasks"* (p. 106). Hence, though short exercises may be a beneficial component of an active learning pedagogy (Falkner & Sheard, 2019), their application requires a careful adjustment of the evaluation methodology.

### 3.2.3 PRACTICAL PROJECTS AND THEIR SUPERVISION

> Programming knowledge, skills, and mental models cannot be effectively acquired in the abstract—they must be anchored in rich practical experience. This highlights the importance of laboratory/practical sessions. (Robins, 2019)

In agreement with the signature pedagogies of arts education (Shreeve et al., 2010), the professors we observed emphasized practice and tasked students with the completion of substantial individual or group projects throughout the term. Shephard (2019) notes supervised computer programming projects allow *"students to demonstrate their higher-order cognitive skills such as creativity [... and] a range of outcomes involving both cognitive and affective learning, such as teamwork or behaving ethically"* (p. 285). Moreover, whereas some professors had precise project instructions, so students would experience an authentic briefing adapted to their specific discipline, others permitted open projects and valued the unexpected, which Watts and Blessinger (2017) regard as a situation where *"students are both more disposed and more incentivized to engage learning creatively."*

Multiple authors see group projects as beneficial, as they allow students to engage with more substantial tasks and more authentic circumstances (Kember & McNaught, 2007; McAllister & Alexander, 2009; Shephard, 2019; Shreeve, Wareing, & Drew, 2009). Moreover, McAllister and Alexander (2009) note group work can promote individual students' confidence and provides a venue for the development of transferrable skills. Though students at several of our cases had been required to develop group projects, in some they could opt to work individually. In doing so, professors were perhaps allowing their students to choose whether to nurture their own artistic individuality or their collaboration and teamwork skills. Perhaps a clearer distinction between individual work and group projects, as seen in the SKETCH study (7.4.3), is advisable as to ensure all students are working toward similar objectives.

One interesting notion about project supervision brought forward by Kember and McNaught (2007) concerns the desirability of professors checking group projects once or twice per session, as professors in our studies either did or, facing difficult working conditions, otherwise saw as ideal. The authors state, *"intruding too much within a group removes the benefits which accrue from the self-directing nature of group activities"* (p. 87). The authors do not argue for lack of supervision; they believe that professors ought to prevent groups from going into unproductive directions or becoming dysfunctional (p. 87), though they do favor a lighter touch. In the same vein, the authors also call for professors to avoid an overly prescriptive approach to tutoring, which may cancel the learning benefits of students' autonomous problem-solving (p. 96). They add:

> Excessive prescription turns a project into an exercise in following a recipe. An absence of directions can lead to blind alleyways or going round and round in circles. A useful technique is to use probing questions to prompt students to work out a direction for themselves. (Kember & McNaught, 2007, p. 88)

Hattie and Timperley (2007) suggest four levels of feedback professors ought to provide their students: about learners' completion of *tasks*, about learners' understanding of the *process*, about how learners conceptualize what they learned, and about learners' self-

efficacy. Nonetheless, we often witnessed interactions during the practical sessions that were tied to students' immediate difficulties; when professors initiated the conversation, their feedback most often related to aesthetics.

One specific type of interaction we witnessed during project supervision, and which we believe is worth paying particular attention to, concerns professors' frequent handling of students' own computers while addressing their programming difficulties, which we noted in 7.8.3.3. Philip Agre (1994) admonishes against handling the computer, cautioning educators to *"attend to the symbolism of the interaction."* In a new book, Golan Levin and Tega Brain (2021) second this 'no touch' policy and quote this tweet by educator Frances Hunger:

> And maybe most important rule for myself when teaching programming. Never touch the keyboard, mouse or trackpad of a students' computer. Just tell, what needs to be done—they own the keyboard, they own the problem. (Hunger, 2019)

We believe pragmatic urgency laid behind the fact that, with a single exception, most professors in our studies often handled students' devices. Moreover, professors sought to solve their students' difficulties. Yet, as Agre (1994) stated, *"your primary goal is not to solve their problem. Your primary goal is to help them become one notch more capable of solving their problem on their own."*

### 3.2.4 UNEXPLORED POSSIBILITIES

Given the small number of cases that comprised our collective case study, we believe a brief discussion about what we did *not* witness is worthwhile. As we mentioned above, the professors that participated in our studies did not ask students to complete activities that are common in computer science education, such as completion exercises and Parson's problems. The literature holds suggestions of additional activities which might be especially salient to the context of arts and design education:

> Although my research team at MIT and I develop technologies to aid the education of artists and designers, we keep a backup perspective of what to do in the event of a power blackout or dropped network. This way of thinking may seem obvious at once, in which case you understand we are all at the mercy of systems that have nothing to do without abilities to think, create, and relate, and everything to do with a new mischievous god of the earth who stands next to fire, wind, and water—technology. (Maeda, 2000, p. 425)

Holding an affinity to how Bogost (2005) envisions the acquisition of procedural literacy, 'unplugged' computing activities are held to stimulate computational thinking skills (cf. Bell, Witten, & Fellows, 1998; Curzon, McOwan, Plant, & Meagher, 2014). The activities require students to devise algorithms and create the tangible result of those algorithms themselves. These may vary from the performative—e.g., having different students represent different variables and show the execution of the algorithm—to the visual—e.g., conditional design exercises (van Mensvoort, 2013), a type of drawing activities that recall some of the principles that underpin Sol LeWitt's *Wall Drawings* series (see 3.1.1).

Knochel and Patton (2015) suggest *"the notion of software outside of the computer is a powerful theme to understand code beyond its rationalist hierarchies."*

Like Kember and McNaught (2007, p. 98), Levin and Brain (2021) also recommend requiring students to complete a weekly journal about their activities.[*] Journaling may interrelate with the desirability of constant practice in order for students to consolidate knowledge, which was recognized by our interviewees.

⁂

Some professors we interviewed mentioned the difficulties raised by the inconsistencies between the environments in students' personal computers. Indeed, we witnessed the difficulties the students attending a practical WEAVE session had in configuring the required web server software (see 7.5.3) and how those difficulties took over the entirety of the session. One possible solution, which may require some effort and possibly the backing of institutional IT departments, would be the distribution of virtual machines that include all necessary tools—e.g., a Linux distribution with the web server software pre-installed. This way, students would only be required to install the hypervisor software (e.g., VirtualBox or VMWare) rather than the entirety of the software stack required for the curricular unit. Indeed, the use of virtual machines is considered beneficial (Corbi & Burgos, 2017; Huang, Chen, & Huang, 2013), as they also provide a single package for distributing all content related to the curricular unit—e.g., examples, documentation, additional utilities.

### 3.3 *Use of online resources*

The professors across our case studies evinced great heterogeneity in their use of learning management systems and in sharing educational resources. Some participants merely employed digital platforms to host materials such as the code produced during lessons or recommended readings, whereas others used all digital tools at their disposal to give their students greater flexibility in engaging with their syllabi; as Kember and McNaught (2007) note, releasing study guides can free class time for active pedagogies (p. 74). Yet, it might have been the case that the teaching conditions, rather than an overt pedagogical stance, shaped professors' use of digital platforms across our studies; use of digital resources in the PRIMER (see 7.2.4) and SKETCH (7.4.4) curricular units appeared to be tied to the large number of students attending classes.

The professor leading SKETCH told us he was looking to implement a *flipped classroom* in the following years in order to make more effective use of the limited class time. Brown and Nelson (2013) suggest digital arts educators ought to curb their enthusiasm about using online resources. Noting the challenge of *"maximizing value from all modes of delivery,"* the authors defend face-to-face instruction as the primary mode for the later stages of development, following from the admonishments of Hubert Dreyfus, for whom *"distance learning will produce only competence, while expertise, let alone mastery, will remain completely out of reach"* (Dreyfus, 2008, pp. 47–48). Yet, this

[*] Indeed, this had been a personal teaching strategy of the researcher, after the scholarship on the benefits of students' weblogs (Muncy, 2014; Sim & Hew, 2010).

is fully compatible with the use of a flipped classroom strategy throughout the early stages of instruction. Milman (2012) regards the flipped classroom strategy as effective for communicating procedural knowledge, and a review of research by O'Grady (2018) testifies to its saliency in the teaching of computer programming.

The professor leading SKETCH also experimented with recording video lessons as to supplement his face-to-face lectures. In 2014, the researcher conducted a survey of about the use of online resources by 504 students at the University of Porto, which found that students valued the availability of lecture notes above that of audiovisual recordings of lessons (Morais, Morais, & Paiva, 2014). During his interview, the professor shared an opposing view (see 7.4.7), noting his students were more likely to engage with video than written materials; indeed, students showed a consistent engagement with video lessons and tutorials across our case studies. There are strong signals of shifting attitudes toward video lessons, which will demand research efforts, especially in the wake of the global experiment in forced adoption caused by the COVID-19 pandemic.

With the notable exception of PRIMER, where students were required to submit their project proposals and their completed work for appreciation in the unit's online forum, professors made little use of online discussion. Given the large classes, the professor responsible for the curricular unit explained he regarded the forum as an extension of the studio as a site of social interaction (7.2.7). Nonetheless, this configured a less than ideal situation. Brown and Nelson (2013) advise against excessive reliance on discussion forums during project ideation and their aesthetic development. Though one of the limitations we faced in that case study concerned access to the discussion forums and we were unable to examine messages, we believe Lovink's (2008) observation about how *"engagement is the state of exception"* in online communities is pertinent; without an explicit evaluation of forum dialogue, it is likely most students did not engage in discussion with their peers.

# 4 ACCEPTANCE, LEARNING, AND OUTCOMES

> A little learning is a dangerous thing;
> Drink deep, or taste not the Pierian spring:
> There shallow draughts intoxicate the brain,
> And drinking largely sobers us again.
> *—Alexander Pope (1688-1744)*

Kember and McNaught (2007) distinguish two distinct approaches to learning, after Marton and Säljö (1976). Students interested in a topic or an activity engage in *deep learning* and attempt to understand concepts, relate them, and integrate them into their own experience and existing knowledge. In contrast, students lacking interest in a topic engage in *surface learning*, memorizing facts without attempting to understanding and integrating them, and moreover limiting their commitment to activities to the level they believe will grant them a passing grade (Kember & McNaught, 2007, pp. 23–24). This notion that higher education students commit strategically to study was confirmed in subsequent research (Shephard, 2019); moreover, this phenomenon is not only limited to how students approach studying for examinations; it holds true in how they approach practical design projects (Shreeve et al., 2009).

As Kember and McNaught (2007) note, *"the teaching, the curriculum design, and the nature of the teaching and learning environment can make a very marked difference to students' approaches to learning and learning outcomes"* (p. 27). Though we witnessed clear evidence of students presenting the two approaches in our case studies, it is not inevitable students with a lesser prior interest in learning about computer programming will commit to a superficial approach to learning.

⁂

In the following subsections, we will discuss how the arts and design students in our studies interacted with computer programming. In §4.1, we will examine their individual characteristics and their acceptance of computer programming. In §4.2, we will examine the students' learning commitment, autonomy, and effectiveness. Finally, in §4.3, we will address the learning outcomes.

## 4.1 *Students' characteristics and acceptance of computer programming*

Our observations agreed with the research that holds as an 'urban legend' the notion that young students, as part of a 'digital native' generation (cf. Prensky, 2001), would be especially proficient at using technologies (Kirschner & De Bruyckere, 2017; Kirschner & van Merriënboer, 2013). During our interviews, we obtained contradictory testimonials about the computer proficiency of students arriving from secondary schooling, yet we witnessed situations while conducting our case studies that suggested that the shift towards mobile technology and the accompanying decline in computer use observed in Portugal after 2015 (PORDATA, 2019b, 2019a) may have weakened students' general computer use skills. Still, it is likely that distance learning during the COVID-19 pandemic increased computer use across all levels of education.

### 4.1.1 EARLY CONTACT AND THE EFFECT OF SENIORITY

> Primeiro estranha-se, depois entranha-se.*
> —Fernando Pessoa's proposed slogan for Coca-Cola (1920s)

Across our studies, participants with previous familiarity with computer programming were more likely to report a more favorable *Attitude Toward Use* and a greater *Behavioral Intention* to use it. Though, on the one hand, the literature advises against statistical association of UTAUT measurements with participants' level of prior familiarity with a technology (Parameswaran et al., 2015), on the other hand, our suggestion agrees with extant research which has found an association between students' interest in computer programming and their prior exposure to the subject (Lishinski & Yadav, 2019).

Over the years, several authors noted the benefits of contact with computing from an early age (e.g., Gouzouasis, 2006; Papert, 1980; Peppler, 2010; Resnick et al., 2009). By the time students begin their higher education, they might have acquired negative stereotypes about programming or otherwise sedimented their interests. Still, this sedimentation process continues. Our preliminary survey found the more senior students still without a contact with computer programming were less likely to harbor any intention of learning to code (see **6**.2.2).

In addition, Kember and McNaught (2007) found that students were likelier to engage in surface learning at the later stages of their courses and blamed universities for demotivating students (pp. 28–29). Still, while institutional policies and attitudes can perhaps account for a decline in students' interest in learning to code, students' more superficial approach may also be related to the sedimentation of their interests and a greater commitment to coursework regarded as more valuable, as it was particularly evident in the TOUCH case (**7**.6).

### 4.1.2 GENDER DIFFERENCES†

Though our preliminary survey evinced crisp gender differences in students' acceptance of computer programming (**6**.2.2), those were not found among the survey responses of our case studies' participants. This upholds how the professors we interviewed denied the existence of gender differences in students' commitment to programming, in the quality of their final projects, and the resulting unit evaluation outcomes.

In one particular but nonetheless critical aspect, our case studies did not conform to maximal variation and were rather more homogeneous. The students attending those courses were good; that is, the candidacy grades they brought from secondary schooling were all above average. It is possible the only institutions that offered to open their doors to us were those that had confidence in the quality of their courses and, as the two are interrelated, in the effectiveness of their students. Hence, the contradiction between the gender differences found in our preliminary survey and the homogeneity found in our

---

* Can be translated as *"At first strange, later entrenched."*

† We note our understanding of gender here is informed by Simone de Beauvoir's notion that psychological differences between men and women are socially acquired (via Bergoffen & Burke, 2020).

case studies might be resolved by positioning gender differences in relation to schooling and socioeconomic conditions.

It is widely acknowledged that women are underrepresented in both teaching and practicing computer programming in many Western countries (Lewis, Shah, & Falkner, 2019; Sack, 2019). Most people are unaware of unconsciously envisioning exact sciences practitioners as men (Tranquada & Correia, 2018). Media narratives, feeding upon and reinforcing these stereotypes, frequently portray programmers as men (furthermore, as unhygienic 'nerds'). Throughout schooling, the unconscious biases and stereotypical portrayals of programmers may lead to aggregate harm to the early interest of girls in computing (Lewis et al., 2019), besides the interest of other people that do not identify with the dominant portrayal.

The 2013 ICILS—International Computer and Information Literacy Study (Fraillon, Ainley, Schulz, Friedman, & Gebhardt, 2014) surveyed 8th grade students, many of whom likely to be enrolling in higher education at the time we conducted our studies. Though Portugal was not a participating country, we believe some of the findings of the 2013 ICILS may be relevant to explain some of our own. ICILS' finding that 8th grade *"female students generally performed relatively better on tasks that involved communication, design, and creativity, and that male students generally performed relatively better on more technical tasks"* (Gebhardt, Thomson, Ainley, & Hillman, 2019, p. 70) met the assessments of professors in our studies.

Moreover, as in our preliminary survey (see **6**.2), ICILS 2013's male participants reported higher interest and enjoyment in using computer technology and *"generally assessed their ability to perform specialized ICT tasks significantly higher on the self-efficacy scale than did their female peers"* (p. 70). However, the female participants exhibited generally higher levels of computer and information literacy (p. 69). Gebhardt and colleagues found it unlikely that teacher's gender caused gender differences in 8th graders' IT effectiveness (Gebhardt et al., 2019). We believe it is doubtful that this changed by the time these students entered higher education.

These results were replicated in the ICILS 2018 study, which, distinguishing between information literacy and computational skills, found female eighth-graders were likelier to report learning about the former, while their male peers were likelier to report learning about the latter (Fraillon, Ainley, Schulz, Friedman, & Duckworth, 2020, p. 244). Therefore, it is possible that the students entering higher education in 2023 will not behave much differently from those who did in 2018. The authors of the 2018 study concluded with a call for further research:

> Why do female students consistently show higher CIL [computer and information literacy] performance than male students and yet male students demonstrate higher levels of engagement with and generally higher achievement in the specialized ICT use associated with CT [computational thinking]? Is this pattern consistent within all countries, are there subgroups of students within countries for whom the gender differences do not follow the pattern? What can policymakers and schools do to redress these differences? This leads to a broader set of questions associated with the nature of teaching of CIL and CT in schools. (Fraillon et al., 2020, p. 268)

⁂

Cheryan and colleagues (2015) found that simple interventions, such as replacing stereotypical objects in computer rooms (*"Star Trek posters, science fiction books, and stacked soda cans"*) with neutral alternatives, were effective in reducing the differences in how different genders regarded learning computer programming. Posters about events and exhibitions were more likely to be found in the walls of higher education institutions dedicated to arts and design than *Star Trek* posters. Hence, though we found the classroom and studio environments in our case studies to be relatively neutral, we did witness instances in which the professors made stereotypical references—to *Star Trek* and *Star Wars*, no less—during their lessons.

#### 4.1.3 A NOTE ABOUT CULTURALLY RESPONSIVE COMPUTING

Gender differences were most salient given the focus of our research at higher education institutions attended by students of a relatively homogenous socioeconomic status. Nonetheless, we ought to acknowledge the nascent research on culturally responsive computing, which extends preoccupations about equity of participation in technological development to marginalized groups (cf. Scott, Sheridan, & Clark, 2014), and also testifies that addressing educational gaps at an early stage is vital to ensure opportunities for healthy citizenship.

### 4.2 *Autonomy and self-efficacy*

According to Goodyear and Ellis (2008), students' role in choosing their own learning methods had been under debate. Authors such as Clark (1994) assumed that adherence to prescribed learning methods dictated positive outcomes, whereas Goodyear and Ellis defend that granting students autonomy in how they engage with the content does not impact their outcomes.

> Moreover, university students are typically faced with more complex learning tasks than those encountered by school students [...]. This further weakens the already tenuous causal connections between instructional methods (or media) and learning outcomes. (Goodyear & Ellis, 2008)

Moreover, Dziuban and Moskal (2011) found that students' evaluation of face-to-face and blended learning courses was invariant and concluded that *"the boundaries of a traditional course were beginning to dissipate."* Still, after a more recent survey of over 1,600 students, Henderson, Selwyn, and Aston (2017) found that most were more interested in using digital technologies to deal with university logistics rather than actual learning. This disagrees with our observations.

It might be the case that the subject of computer programming is particularly suitable to online learning: languages and development environments' most up to date documentation is to be found online, and, besides the wide range of tutorials and free online courses available elsewhere, some programming languages' official websites (e.g., Processing or Unity's) include a section with video and text-based tutorials for first-time

learners. Furthermore, the wide availability of tutorials often allowed students to 'cut corners' in their learning. Though the professors we interviewed disagreed on students' autonomy, several agreed that students were likely to be acritical when studying and reusing resources found on the internet. Referring to a popular online help forum, the professor responsible for the GRAPHIC curricular unit spoke of students' use of *"Stack Overflow-based programming"* (see 7.3.5), supporting Kirschner and van Merrieënboer's (2013) comment about the *"Googlification of education."* The same professor referred to how first-time learners' lack of canonical knowledge and vocabulary (which he termed the *"nomenclature problem"*) impedes effective autonomy.

### 4.2.1 DIFFICULTIES AND REACTIONS

The students participating in our case studies evinced difficulties that were consistent with the literature reviewed by Robins (2019). Though sometimes they had trouble with aspects of writing (e.g., syntactical mistakes such as missing semicolons), these difficulties were quickly solved. Students had more severe problems with planning and designing the algorithms for their programs; *"how should we start?"* was a frequent question posed to professors.

Though the media production and design students were relatively comfortable with their problem domains (e.g., they understood principles of computer graphics such as pixels or coordinate systems), and so did the music students in the FLOW study, they had frequent problems conceptualizing the 'notional machines' (cf. du Boulay, 1986) afforded by the programming languages they were learning. These problems are most often manifested in misconceptions about execution order and control flow (i.e., loops and conditional statements). Though professors made frequent statements to the effect that no intelligence or 'superbug' (cf. Pea, 1986) inhabits the computer, it may have been the case that syllabi were too broad for the time available. It is likely that professors did not present their worked examples in enough detail for all students to construct their own clear notional machine model when initiating learning, causing an accumulation of difficulties for those who did not 'get it' then. This led many students to have trouble identifying issues and repairing their exercises and projects later on.

Perkins and colleagues categorized first-time computer programming learners as either *stoppers*, *tinkerers*, or *movers* (Perkins, Hancock, Hobbs, Martin, & Simmons, 1986). As the names imply, *stoppers* are frustrated and give up when confronted with problems, while *tinkerers*[*] attempt to guess solutions without a clear understanding of the problems. Only *movers*, who tried to understand their problems (taking care to understand error messages, for instance), are effective. Indeed, we witnessed all these three behaviors across our studies. Moreover, though professors made efforts to help students become movers by developing a habit of testing their work incrementally, tracing their code (sometimes reading it aloud), and reading the error messages, we found that the plethora of available online resources possibly pulled students into a tinkerer approach.

[*] Note that the term *tinkerer* came to acquire a positive meaning in recent years, yet that was not the case at the time.

Difficulties with programming, if unaddressed, may lead some students to give up after going down the path of what Donald Norman described as 'learned helplessness,' or an accumulation of negative momentum (cf. Robins, 2019):

> The phenomenon called learned helplessness may help explain the self-blame. It refers to the situation in which people experience failure at a task, often numerous times. As a result, they decide that the task cannot be done, at least not by them: they are helpless. They stop trying. (Norman, 1988, p. 42)

Across our observations, we witnessed situations in which students blamed themselves or otherwise dismissed programming as 'too difficult.' As we have seen in §3.2, helping students transition from self-blame to self-efficacy may require professors to proceed at a slower pace or to retread their steps while teaching or supervising students' projects; we already mentioned Robins (2019) suggestion of making recovery streams available. We see this might be incompatible with the working conditions faced by some of the professors in our studies. Yet, the significant dropout rate, especially in the large classes, argues for more substantial investment in class time.

#### 4.2.2 SELF-EFFICACY

According to a literature review by Lishinski and Yadav (2019), students' self-efficacy has a powerful influence on their persistence in learning to program. It is highly likely this relationship is bidirectional: the self-efficacy of student participants in our preliminary survey was associated with their prior knowledge of computer programming (see **6**.2.2).*

Female students across our surveys also reported significantly lower self-efficacy than their male peers, in agreement with Lishinski and Yadav's (2019) review. Yet, the authors note that the literature indicates that female programming students are quicker to internalize feedback and revise their perception of self-efficacy. This also testifies to the importance of educators considering the symbolism of their interactions and how they systematize their feedback, previously mentioned in §3.2.3.

### 4.3 *Learning outcomes*

According to Robins (2019), introductory computer programming units have historically presented high failure and dropout rates. In addition, the author tells us that outcomes tend to be bimodal: average students are rare, as most are either high-performing or low-performing. It is possible that these bimodal outcomes are associated with students' approaches: movers are likely to perform well, tinkerers are likely to perform poorly, and stoppers are likely to give up (Robins, 2019). Professors across our case studies did acknowledge that a significant number of students gave up during the semester; in agreement with Robins' review, students in the smaller classes were less likely to give up. Yet, the professors also reported low failure rates among the students that stayed until the end. However, it is possible that the evaluation method was not demanding.

* We had divergent findings in our case studies' surveys, albeit with few participants and differing self-efficacy constructs across cases.

Across most of the curricular units we scrutinized, we believe there were few measures in place to offer an assurance of learning (cf. Shephard, 2019).

Some of our interviewees acknowledged having low expectations of most students and admitted that the minimum objectives they set for giving a passing grade were undemanding. More worryingly, some seemed willing to accept some deceit from their students. Though there is a gray area between the acceptable reuse of code and the outright copying from online sources (Lancaster, Robins, & Fincher, 2019), the professors evinced nonchalance when students plagiarized large portions of source code—in some cases authored by the professors themselves. In addition, some admitted to passing students while being aware of their reliance on the work of group colleagues.

Putting aside the critical issue of how grade inflation and deflated assurance may sully students' commitment throughout the entirety of their courses, it may also be the case that lowering the bar and treating arts and design students to an easy introduction to computer programming leads to a lesser appreciation of its relevance for arts and design practices. As Mateas stated:

> A New Media introduction to CS should be a difficult course, with the challenge lying not in programming conceived of as applied mathematics, but in connecting new media theory and history with the concrete craft practice of learning to read and write complex mechanical processes. (Mateas, 2005)

Some of the professors we interviewed complained about what they saw as the secondary standing of computer programming within the respective curricula. Giving students too much latitude to commit to a superficial learning strategy may be counterproductive for raising that status:* more students need to commit to deep learning for their classes to consistently produce the high-quality work that captures attention and recognition to a particular discipline.

### 4.3.1 LEARNING FRAGILITY

Superficial commitment to study leads to brittle learning. Interviewees acknowledged the fragility of most students' learning. Robins (2019) notes the fragility of knowledge acquired over a single semester of computer programming has been reported in studies since the 1980s, and indeed, students in the WEAVE and the TOUCH studies, which did not concern introductory units, did not seem to bring a comfortable understanding of programming from earlier semesters (see 7.5, 7.6).

Students' learning fragility may relate to little progress in students' development of their computational thinking skills. Guzdial (2019) is quite skeptical about the impact of an introduction to computer programming:

> Our best evidence suggests that learning programming does not lead to improved generalized problem-solving skills. However, that evidence also suggests that students in these studies did not learn much programming. (Guzdial, 2019)

* If we are permitted to speculate, it is possible that some professors believe that learning programming ought to be optional, and thus proceed to integrate that optionality within their evaluation methodology. This ought to merit consideration by curriculum designers.

The author then notes that, even if students spent more time learning to program, that was unlikely to improve their general problem-solving skills. If true, that brings severe implications for the notion that teaching computer programming in higher education brings benefits for most first-time learners. It may also be the case that the acquisition of logical thinking skills needs to come first, at an earlier stage of schooling, as suggested by the professor responsible for PRIMER (see 7.2.7). Still, Guzdial (2019) leaves a door open: the author speculates that an adequate programming language for teaching computational literacy perhaps was not yet invented.

※

In the next chapter, we will bring this thesis to a conclusion. We will present a short summary of our findings, present our recommendations to stakeholders, and address possible avenues for future research.

*(blank page)*

# 9

# *Conclusions*

★ SYNOPSIS We conclude this thesis with one look to the past and two looks to the future. In section §1, we present a summary of our most salient findings. In section §2, we offer a set of recommendations that can be beneficial toward improving computer programming education within arts and design disciplines and thus merit the attention of policymakers, institutions, curriculum designers, and educators. Finally, in section §3, we point out possible future research possibilities.

## 1 SUMMARY OF FINDINGS

We set out to investigate how computer programming is taught and learned within arts and design undergraduate courses, examining the specific context of Portuguese public higher education. We reckon the limited scope of our research, a small sample within a specific disciplinary area within a particular educational system within an individual country. Nonetheless, after the articulation with relevant scholarship presented in the previous chapter, we believe that a set of meaningful findings merit consideration.

### 1.1 *The value of computer programming*

The professors that participated in our studies presented two main types of motivations for teaching programming to arts and design students (**6**.3.4, **7**.8.1.1, **8**.2.1). Whether code would be a 'tool for doing' or a 'tool for thinking' (cf. Guzdial, 2019) was translated into foci on students' future employability or their acquisition of new literacies (cf. Vee, 2013). Some professors emphasized the value of computing as an expressive medium, noting the as of yet still largely unrealized potential of code as a means to cultivate virtuosity.

Professors made programming salient to arts and design students through their choices of languages and tools (**8**.2.4) and their emphases on experiential learning and issuing 'authentic tasks' such as practical projects (**7**.8.1.4, **8**.3.2.3). However, we found a reluctant engagement with the students' specific course or main disciplinary area in several of our case studies.

## 1.2 *Working conditions*

We witnessed oversized classes in crowded studios in some of our case studies. Some of the professors noted the difficult conditions and the little time they had to supervise individual students' work. In addition, professors' coding demonstrations during lessons were often limited by the quality of the available projection equipment (7.8.2).

Studios' general-purpose computer equipment was mainly abandoned as students prefer to use their own devices. In this context, we witnessed the beginning of a trend towards the acquisition of more specialized equipment, which may aid in the recovery of the studio space's centrality (8.2.3).

## 1.3 *Pedagogical approaches*

Several professors that took part in our case studies did not have a formal education in computer programming or, conversely, in the main disciplines of the course where they are teaching, and thus recognized a pedagogical deficit (7.8.3). In addition, some evinced the didactic, content-oriented approach that researchers such as Kember and McNaught (2007) attribute to less experienced educators (8.3). Others saw themselves as learning facilitators and stressed that developing students' autonomy was their primary goal—i.e., 'teaching to learn.' Several interviewees believed in the need for a broad syllabus to meet that objective (8.2.2), even though scholarship advises prioritizing depth over breadth (cf. Blikstein & Moghadam, 2019; Kember & McNaught, 2007).

Professors giving lectures were reluctant to pace their explanations and coding demonstrations to ensure all students understood, even though classroom conditions were often a source of undesirable cognitive load (8.3.2). Visual programming languages (e.g., Max) were understood to be particularly challenging in this regard. The professors often pushed ahead with their lessons, asking students to engage with their difficulties in their independent study.

Perhaps owing to their admitted lack of training in computing pedagogy, we found several professors did not thoroughly plan their programming examples—e.g., setting granular subgoals or setting analogical solutions to benefit transferability. The professors also disregarded active learning activities for novices recommended in computing education literature—e.g., code completion or code rearrangement exercises (8.3.2). Moreover, the in-class and homework exercises were assigned little weight when unrelated to the major projects that students had to complete throughout the term.

### 1.3.1 ONLINE RESOURCES

The professors leading the curricular units we scrutinized made heterogeneous use of learning management systems and online resources, and that use was shaped mainly by the working conditions (7.8.2.3, 8.3.3). In relation to existing scholarship, the students participating in our case studies evinced a more favorable attitude toward video lessons.

Still, novice students often lacked the necessary knowledge to assess the suitability and understand the online resources they found. Their reliance on search engines often led to an ineffective 'tinkering' approach to programming their projects (8.4.2).

### 1.3.2 TUTORING

While tutoring, professors were more likely to try to solve students' immediate problems than to provide systematic feedback (**8**.3.2). In particular, most professors in our case studies often handled students' computers and wrote code by themselves, a behavior which may do little to keep students from internalizing a feeling of helplessness toward programming while also lowering the requisite demand for deep thinking.

## 1.4 *Students and their acceptance of computer programming*

Though the professors we interviewed had different perspectives on the computer use competencies of their students, we witnessed situations that point toward a decline in those skills, likely due to the increasing role of mobile technologies (**7**.8.4, **8**.4.1).

Previous familiarity with computer programming was associated with greater acceptance and interest in the subject (**8**.4.1), in agreement with the extant literature (cf. Lishinski & Yadav, 2019). In addition, the most consistent predictors of a future intention to use computer programming were students' preexisting *Attitude Toward Use* as well as *Performance Expectancy* (**6**.2.3, **7**.8.5, **8**.1.2.2), which testifies to the importance of demonstrating the usefulness of computing.

Students' interest in learning to program declined across our studies as they progressed through their courses (**6**.2.2, **8**.4.1). Moreover, we also saw some evidence that more senior students were less likely to commit to deep learning, a view which was also shared by professors we interviewed and which agrees with the broader literature on higher education (cf. Kember & McNaught, 2007).

Our preliminary survey found the gender differences in programming acceptance suggested by recent scholarship (**6**.2.2). In contrast, our case studies presented a more equitable portrayal (**7**.8.4). Still, our studies concerned a set of courses attended by students with a good academic background associated with families of above-average socioeconomic status (**8**.4.1), where research suggests gender differences are less significant (cf. Fraillon, Ainley, Schulz, Friedman, & Gebhardt, 2014).

While developing their projects in the studio, students showed the most difficulties with planning and designing algorithms rather than the specificities of programming syntax (**8**.4.2.1). A common question was, *"how do we begin?"*

## 1.5 *Learning outcomes*

According to the professors we interviewed, there were significant dropout rates in the curricular units attended by larger classes, for which the professors did not hold themselves responsible (**8**.4.3), in disagreement with the recommendations in literature (cf. Kember & McNaught, 2007). Yet, the failure rates of students who completed the term were minimal. Some professors admitted to setting undemanding evaluation requirements, which we observed as leading students to adopt a surface learning strategy. In addition, the manner in which several curricular units permitted group projects to be undertaken gave students opportunities to hide a lack of commitment and avoid engaging with programming.

Moreover, insufficient learning assurance measures were in place. Some of the professors paid little attention to how students reused code in their work, disregarding instances of borderline plagiarism (**7**.8.6, **8**.4.3). The interviewees admitted that students' learning was fragile, and the salience of programming was often left unproven.

## 2 RECOMMENDATIONS

Our findings hold some valuable lessons that merit consideration by educational policymakers, institutions, and professors. Noting the aforementioned small scope of our inquiry, we nonetheless present the following set of suggestions:

### 2.1 *For policymakers and higher education institutions*

- COMPUTING IN BASIC SCHOOLING Many students evinced a lack of preparation in both logical thinking and IT competencies. Early introductions to programming may allow young students to develop both sets of skills and set a proper foundation for further exploration during higher education:

  > To achieve a broader and more profound procedural literacy will require developing an extended curriculum that starts in elementary school and continues through college. Encountering procedurality for the first time in a graduate level course is like a first language course in which students are asked to learn the grammar and vocabulary, read and comment on literature, and write short stories, all in one semester. (Mateas, 2005)

  We understand there are recent initiatives in this direction. Besides the growing number of organizations dedicated to equitable programming education (see 2.3.1), governments have also paid attention: in Portugal, the Education General Directorate has launched in 2015 a pilot project introducing computer programming to 3rd and 4th grade pupils.*

- CODE EARLY IN THE CURRICULUM Aside from earlier contact with computing, it is suggested that an introduction to computer programming as soon as the curriculum allows is beneficial, even if said curriculum will offer one single semester of contact. Students are more likely to deeply engage with programming, besides being allowed the opportunity to explore it further throughout their course.

- MORE TIME FOR EACH STUDENT Though studios should be comfortable working spaces, we note professors' key complaints about class size mostly concerned the time that would be available to tutor each student face-to-face. Longer practical sessions or office hours assignments for visiting professors may be adequate alternatives to dividing students into smaller classes.

- PEDAGOGICAL TRAINING There is a rich scholarship about pertinent computing pedagogies with which the professors in our studies did not engage. Institutions can promote training initiatives and disciplinary meetings, which would be highly beneficial.

- LEARNING ASSURANCE We believe professors should be more demanding to ensure students' learning is robust. Yet, learning assurance may also require institutions to correct their incentives—e.g., adjusting the weight given to pedagogical surveys of students in professional evaluation in order to defend demanding professors.

---

* www.erte.dge.mec.pt/iniciacao-programacao-no-1o-ciclo-do-ensino-basico-apresentacao

## 2.2 *For professors*

- DEPTH OVER BREADTH The literature suggests that prioritizing a narrow and deep syllabus over a syllabus that is wide and shallow may prove more beneficial to abet students' autonomy. It may also be the case that insisting on students practicing—e.g., exercises that only use loops and conditions, will produce a greater understanding and learning that will evince lesser decay over time, setting an adequate foundation for developing computational thinking skills and for advanced learning.

- ACTIVE PEDAGOGIES Notwithstanding the availability of institutional training initiatives, educators ought to hone their craft. Computing education and higher education literature offer relevant active learning pedagogies that may have a place in the arts and design studio, such as code completion exercises (van Merriënboer, 1990), code rearrangement exercises such as 'Parson's problems' (Parsons & Haden, 2006), 'unplugged' computing (Bell, Witten, & Fellows, 1998) or conditional design activities (van Mensvoort, 2013), journaling or weblog activities (cf. Sim & Hew, 2010).

- FIRST IMPRESSIONS MATTER As students' perception of the usefulness (i.e., *Performance Expectancy*) of computer programming is predictive of their commitment to learning, how educators first present the subject ought to merit special consideration. As we witnessed during one of our case studies, beginning the semester with an event or exhibition promoting creative code may prove beneficial.*

- LEARNING EQUITY Early throughout their terms, educators ought to assess whether students bring negative attitudes and stereotypes regarding computer programming. In some cases, active remedial activities to demystify stereotypes may be required. In any case, as suggested by Cheryan et al. (2015), attention to the classroom environment and mindfulness about lessons' examples and cultural references is advised.

- SLOW DOWN Educators ought to take responsibility for students at greater risk of giving up and pay particular attention to the students evincing the most difficulties Moreover, the classroom conditions often generate cognitive constraints, especially in larger classes. A full recognition of what it is like to be a novice is needed:

> We have the responsibility to make the familiar strange—not only to rethink the uses of technologies, but also to know it again for the first time as we consider where our students may be starting. (Bruce & Hogan, 1998)

- INTERLEAVED ACTIVITIES AND SPACED PRACTICE Learning to code is challenging for most novices, and educators ought to monitor students for signs of tiredness. Inserting variety during lessons may let students 'rest while working.' Interleaving live coding examples, related discussions about arts or design, and in-class exercises may prove more effective than dedicating entire sessions to each of these modalities.

* Still, given the formative impact, educators ought to ponder whether such an event might be suitable for novices.

Moreover, interleaving in-class exercises is a form of spaced practice. Others, such as homework or the setting of regular milestones for larger projects, may also bolster a deeper engagement with learning to code. Still, evaluation methods ought to assign spaced practice the necessary weight to command students' attention.

- MINDFUL FEEDBACK Educators ought to yield to Agre's (1994) caution to *"attend to the symbolism of the interaction"* and take note of how a practice of solving students' immediate problems may disagree with the role of a facilitator. One particularly salient consequence is that educators should avoid handling students' computers, especially to write any code.

  Other authors (e.g., Kember & McNaught, 2007) also advise against regular inspections of group activities in favor of a manner of supervision aimed at correcting students' path when the need arises. We encourage educators to find systematic frameworks for providing feedback (e.g., Hattie & Timperley, 2007).

- ONLINE CONTENT CURATION Educators ought to emphasize to students how web searches can be unhelpful—or even lead to frustration—when lacking the necessary knowledge to identify relevant results. Hence, it is advisable that all educators provide their students with a set of recommended online resources and tutorials.

- SALIENT EVALUATION The evaluation methodology ought to reflect all activities students are expected to complete throughout the term. As suggested (e.g., Falkner & Sheard, 2019; Kember & McNaught, 2007), assigning specific marks to smaller in-class and homework exercises, rather than listing them under the umbrella of 'participation,' may be beneficial to securing a deeper commitment and spaced practice from students.

- LEARNING ASSURANCE Educators ought to take measures to better the robustness of the knowledge acquired in introductory computer programming classes. Evaluation can be more demanding, and special attention should be granted to students' ethical conduct—e.g., ensuring reused materials are correctly referenced and that all elements had meaningful contributions to group work.

## 3 FUTURE RESEARCH DIRECTIONS

The main phase of our research, the case studies, took place during the last 'normal' academic year before the COVID-19 pandemic. At the time of writing, it is still unclear whether the pandemic years will mark a mere interregnum or the beginning of a revolution in how higher education is regarded and practiced. Did we unwittingly build a historical portrayal? Or will 'business' go back to 'usual' once the situation subsides?

This is a strange moment to offer indications toward future research. Even if the previous status quo prevails, the mass experiment in educational technology adoption brought by the pandemic will leave a mark on an entire generation of students. Carrying out comprehensive research initiatives on the mold of the International Computer and Information Literacy Studies (e.g., Fraillon et al., 2014), but aimed at students at all levels of schooling, will be urgent to shape future institutional policies and curricula.

⁂

Though the present context presents a more brittle foundation on which to set future inquiries, we nonetheless believe our research, in both its findings and its limitations, suggests avenues that merit consideration.

- LONGITUDINAL ACCEPTANCE Our failure to gather adequate longitudinal data about students' acceptance and use of computer programming throughout our case studies was one of the main limitations in our research. It also suggests an obvious direction for future efforts: broader longitudinal surveys to determine the impact of introductory computer programming units on arts and design students' acceptance of the subject. Greater institutional support would be needed to ensure a large, timely, and perhaps mandatory participation.

- UTAUT STUDIES Given the limitations of our case studies, we were unable to conduct an interrogation of Venkatesh and colleagues' (2003) Unified Theory of Acceptance and Use of Technology. UTAUT literature suggests it would be desirable to adapt the model more closely to the technology under scrutiny (Brown, Dennis, & Venkatesh, 2010). The abovementioned longitudinal acceptance studies may offer that opportunity to examine the interplay of UTAUT constructs and analytically produce a contribution to the field of quantitative technology acceptance research.

- FOCUS ON LEARNERS Our concerns with ensuring privacy and meeting the timeframe of the doctoral study led us to a shallower engagement with the students participating in our studies. Nonetheless, we believe that interviewing students and quasi-ethnographic observation of students' work and collaborative habits outside the studio may provide vital insights into their acceptance of computer programming.

- COMPUTATIONAL THINKING AND LITERACY The fragility of students' introductory programming knowledge and Guzdial's (2019) suggestion that it has a small impact on computational thinking abilities require examination. Broad institutional support would allow robust assessments of students' computational thinking and

computing literacy skills to determine whether seniority and the number of hours spent in contact with computer programming in the studio present benefits. Moreover, arts and design students enrolled in courses without programming in their curricula would provide a natural control group.

In addition, this avenue of research may also prove helpful to determine to what extent do additional factors, such as the chosen programming language or the evaluation methodology, impact students' competencies.

✦ EDUCATIONAL MEDIA AND TECHNOLOGIES The heterogeneity of professors' use of educational technologies and resources presents a welcoming territory for further research; moreover, it is expected to gain additional relevance in the wake of the pandemic. Our findings already hinted at a growing preference for video recordings over lecture notes, and a continued inquiry into students' media choices is justified.

The specificities and constant evolution of computer programming education also present numerous paths for inquiry: into programming languages; into development environments and tools such as game creation systems; or into using technologies such as virtual networking (VNC), collaborative online environments,* or virtual machines to support learning and studio practice.

✦ PEDAGOGICAL STRATEGIES As noted, we witnessed a disconnection between the pedagogical recommendations found in computing education literature and professors' teaching practice. Still, as we mentioned at the beginning of this chapter, our studies constitute a limited sample of a small part of the educational reality. Hence, our final suggestion for future inquiry concerns the need for a continued research engagement with the educational practices within the arts and design computing studio.

※

> Everything was to be done. All the adventures are still there.
> —Kodwo Eshun, in a 2000 interview with Geert Lovink†

* Such as Codepen—codepen.io—or Replit—replit.com.

† heise.de/tp/features/Everything-was-to-be-done-All-the-adventures-are-still-there-3447386.html

*(blank page)*

# REFERENCES


**A3ES** (2020). *Relatório de monitorização da avaliação do ensino superior*. Retrieved from www.a3es.pt/pt/documentos/relatorio-de-monitorizacao-da-avaliacao-do-ensino-superior

**Adajian, T.** (2018). The Definition of Art. *The Stanford Encyclopedia of Philosophy*. Retrieved from https://plato.stanford.edu/archives/fall2018/entries/art-definition/

**Agre, P. E.** (1994). How to help someone use a computer. *The Network Observer*, *1*(5).

_____ (2000). Infrastructure and Institutional Change in the Networked University. *Information, Communication & Society*, *3*(4), 494–507. DOI:10.1080/13691180010002323

**Aires, L.** (2011). *Paradigma Qualitativo e Práticas de Investigação Educacional*. Porto: Universidade Aberta.

**Ajzen, I.** (1985). From Intentions to Actions: A Theory of Planned Behavior. *Action Control*, 11–39. DOI:10.1007/978-3-642-69746-3_2

_____ (1991). The theory of planned behavior. *Orgnizational Behavior and Human Decision Processes*, *50*(2), 179–211. DOI:10.1016/0749-5978(91)90020-T

**Ajzen, I., & Fishbein, M.** (1969). The prediction of behavioral intentions in a choice situation. *Journal of Experimental Social Psycholoogy*, *5*, 400–416.

_____ (1972). Attitudes and normative beliefs as factors influencing behavioral intentions. *Journal of Personality and Social Psychology*, *21*(1), 1–9. DOI:10.1037/h0031930

**Alexander, B.** (2016). A Devil's Dictionary of Educational Technology. Retrieved from www.bryanalexander.org/2016/09/22/a-devils-dictionary-of-educational-technology/

**Amaral, A.** (2016). *Arquitetura e Urbanismo (Ciclo de Estudos Temáticos)*. Lisboa: A3ES.

**Amos Hatch, J.** (2002). *Doing Qualitative Research in Education Settings*. SUNY Press.

**Andreessen, M.** (2011, August 20). Why software is eating the world. *Wall Street Journal*.

**Ang, I.** (1994). In the Realm of Uncertainty: The Global Village and Capitalist Postmodernity. In D. J. Crowley & D. Mitchell (Eds.), *Communication Theory Today* (pp. 193–213). Stanford University Press.

**Ascott, R.** (1964). The construction of change. *Cambridge Opinion*, *41*, 37–42.

**Ashwin, P.** (2009). Conceptualising Teaching and Learning Interactions in Researching Higher Education. In M. Tight, K. H. Mok, J. Huisman, & C. C. Morphew (Eds.), *The Routledge International Handbook of Higher Education* (pp. 37–46). Routledge.

**Aufderheide, P.** (2007). *Documentary film: A very short introduction*. Oxford University Press.

**Bailey, S.** (2011). (Only an Attitude of Orientation). In F. Allen (Ed.), *Education (Whitechapel: Documents of Contemporary Art)* (pp. 158–162). MIT Press.

**Bandura, A.** (1977). Self-efficacy: Toward a Unifying Theory of Behavioral Change. *Psychology Review*, *84*(2), 191–215.

**Barber, M., Donnelly, K., & Rizvi, S.** (2013). *An Avalanche is Coming - Higher Education and the Revolution Ahead*. London: Institute for Public Policy Research.

**Bartlett, S., & Burton, D.** (2016). *Introduction to Education Studies*. Sage.

**Beard, D. V.** (2019). Programming Languages for University Courses. In A. Tatnall (Ed.), *Encyclopedia of Education and Information Technologies* (pp. 1–6). Springer International Publishing. DOI:10.1007/978-3-319-60013-0_133-1

**Beattie, K.** (2004). *Documentary Screens: nonfiction film and television*. Palgrave Macmillan.

**Bell, T., Witten, I., & Fellows, M.** (1998). *Computer Science Unplugged*. Christchurch, New Zealand: University of Canterbury.

**Bergoffen, D., & Burke, M.** (2020). Simone de Beauvoir. *Stanford Encyclopedia of Philosophy*. Retrieved from https://plato.stanford.edu/archives/sum2020/entries/beauvoir/

**Bers, M. U.** (2017). *Coding as a Playground*. Taylor & Francis.

**Binti Shamsuddin, N.** (2009). Students perception towards the implementation of computer graphics technology in class via Unified Theory of Acceptance and Use of Technology (UTAUT) model. *Lecture Notes in Computer Science*, *5857*, 886–893. DOI:10.1007/978-3-642-05036-7_84

**Blessinger, P., & Watts, L. S.** (2017). History and nature of creative learning. In L. S. Watts & P. Blessinger (Eds.), *Creative Learning in Higher Education - International perspectives and approaches* (pp. 3–13). Routledge. DOI:10.4324/9781315659473

**Blikstein, P.** (2018). *Pre-College Computer Science Education: A Survey of the Field*. Stanford.

**Blikstein, P., & Moghadam, S. H.** (2019). Computing Education - Literature Review and Voices from the Field. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 56–78). Cambridge University Press. DOI:10.1017/9781108654555.004

**Blumer, H.** (1979). The nature of symbolic interactionism. In C. D. Mortensen (Ed.), *Basic Readings in Communication Theory* (2nd ed., pp. 102–120). Harper & Row. DOI:10.4324/9781315080918-10

**Bogdan, R., & Biklen, S.** (1994). *Investigação qualitativa em educação: uma introdução à teoria e aos métodos*. (M. J. Alvarez, Trans.). Porto Editora.

**Bogost, I.** (2005). Procedural Literacy: Problem Solving with Programming, Systems, and Play. *Journal of Media Literacy*, *52*(1–2), 32–36.

**Bolter, J. D., & Grusin, R. A.** (2000). *Remediation: Understanding New Media*. MIT Press.

**Britannica** (2018). The arts. In *Encyclopaedia Britannica*. Retrieved June 6, 2018 from www.britannica.com/topic/the-arts

**Brown, A. R., & Nelson, J.** (2013). Digital music and media creativities. In P. Burnard (Ed.), *Developing Creativities in Higher Music Education: International Perspectives and Practices* (pp. 61–74). Routledge. DOI:10.4324/9781315885223

**Brown, N. C. C., & Wilson, G.** (2018). Ten quick tips for teaching programming. *PLOS Computational Biology*, *14*(4), e1006023. DOI:10.1371/journal.pcbi.1006023

**Brown, S. A., Dennis, A. R., & Venkatesh, V.** (2010). Predicting Collaboration Technology Use: Integrating Technology Adoption and Collaboration Research. *Journal of Management Information Systems*, *27*(2), 9–54. DOI:10.2753/MIS0742-1222270201

**Bruce, B. C., & Hogan, M. P.** (1998). The disappearance of technology: Toward an ecological model of literacy. In D. Reinking, M. C. McKenna, L. D. Labbo, & R. D. Kieffer (Eds.), *Handbook of literacy and technology: Transformations in a post-typographic world* (pp. 269–281). Routledge.

**Brucker-Cohen, J.** (2013). Media Production with Arduino. In R. T. Scholz (Ed.), *Learning Through Digital Media*. Institute for Distributed Creativity.

**Bush, V.** (1945, July). As We May Think. *The Atlantic*, 1–12.

**Cabeleira, H.** (2018). We Don't Need No Education: all we need is 'learning to learn', from cradle to grave. In J. P. Queiroz & R. A. de Oliveira (Eds.), *Arte e Ensino: Propostas de Resistência* (pp. 91–106). Lisboa: Centro de Investigação e Estudos em Belas-Artes.

**Cabero-Almenara, J., & Roig-Vila, R.** (2019). The motivation of technological scenarios in Augmented Reality (AR): Results of different experiments. *Applied Sciences*, *9*(14). DOI:10.3390/app9142907

**Calado, M., & Ferrão, H.** (2013). Da Academia à Faculdade de Belas-Artes. In S. C. Matos & J. Ramos do Ó (Eds.), *A Universidade de Lisboa nos Séculos XIX e XX - vol. II* (pp. 1106–1151). Universidade de Lisboa.

**Carr, N.** (2005). The amorality of Web 2.0. Retrieved January 3, 2014, from http://www.roughtype.com/?p=110

_____ (2009). *The Big Switch: Rewiring the World, from Edison to Google*. W. W. Norton & Company.

**Carter, A. S., & Hundhausen, C. D.** (2011). A review of studio-based learning in computer science. *Journal of Computing Sciences in Colleges*, *27*(1), 105–111.

**Chandler, P., & Sweller, J.** (1991). Cognitive Load Theory and the Format of Instruction. *Cognition and Instruction*, *8*(4), 293–332. DOI:10.1207/s1532690xci0804_2

**Cheryan, S., Master, A., & Meltzoff, A. N.** (2015). Cultural stereotypes as gatekeepers: increasing girls' interest in computer science and engineering by diversifying stereotypes. *Frontiers in Psychology*, *6*. DOI:10.3389/fpsyg.2015.00049

**Childress, M.** (2016). Utopian Futures for Learning Technologies. In N. Rushby & D. W. Surry (Eds.), *The Wiley Handbook of Learning Technology* (pp. 557–570). John Wiley & Sons. DOI:10.1002/9781118736494.ch29

**Christensen, C., & Eyring, H. J.** (2011). *The Innovative University: Changing the DNA of Higher Education*, 47–53. American Council of Education.

**Clark, R. E.** (1994). Media will never influence learning. *Educational Technology Research and Development*, *42*(2), 21–29. DOI:10.1007/BF02299088

**Cohen, L., Manion, L., & Morrison, K.** (2017). *Research Methods in Education*. Routledge.

**Collins, D.** (2018). Using Graphics to Inspire Failing Students. In J. Carter, M. O'Grady, & C. Rosen (Eds.), *Higher Education Computer Science - a Manual of Practical Approaches*. Springer.

**Comber, O., Mayer, H., Motschnig, R., & Haselberger, D.** (2019). Engaging Students in Computer Science Education through Game Development with Unity. In *2019 IEEE Global Engineering Education Conference* (pp. 199–205). Dubai: IEEE.

**Compeau, D., & Higgins, C. A.** (1995). Computer Self-Efficacy: Development of a Measure and Initial Test. *MIS Quarterly*, *19*(2), 189–211.

**Corbi, A., & Burgos, D.** (2017). Open distribution of virtual containers as a key framework for open educational resources and STEAM subjects. *Electronic Journal of E-Learning*, *15*(2), 126–136.

**Cousin, G. (2009).** *Researching Learning in Higher Education: An Introduction to Contemporary Methods and Approaches*. Routledge.

**Cowling, M. A.** (2016). Navigating the Path Between Positivism and Interpretivism for the Technology Academic Completing Education Research. In B. Harreveld, M. Danaher, C. Lawson, B. A. Knight, & G. Busch (Eds.), *Constructing Methodology for Qualitative Research* (pp. 45–58). Palgrave Macmillan. DOI:10.1057/978-1-137-59943-8

**Creswell, J. W.** (2013). *Educational Research: planning, conducting, and evaluating* (4th ed.). Pearson.

**Cruz, N. C., Ferrández, M. R., Redondo, J. L., Álvarez, J. D., & Ortigosa, P. M.** (2019). Applications of Chip-8, a virtual machine from the late seventies, in current degrees in computer engineering. In *EDULEARN19 Proceedings* (pp. 1720–1729). Palma de Mallorca.

**Csikszentmihalyi, M.** (1990). *Fluir*. (M. Amado, Trans.). Lisboa: Relógio de Água.

**Curzon, P., Bell, T., Waite, J., & Dorling, M.** (2019). Computational Thinking. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 513–546). Cambridge University Press. DOI:10.1017/9781108654555.018

**Curzon, P., McOwan, P. W., Plant, N., & Meagher, L. R.** (2014). Introducing teachers to computational thinking using unplugged storytelling. In *Proceedings of the 9th workshop in primary and secondary computing education* (pp. 89–92).

**Danto, A.** (1964). The Artworld. *The Journal of Philosophy*, *61*(19), 571. DOI:10.2307/2022937

**Dasuki, S. I., Ogedebe, P., & Idemudia, E. C.** (2016). Factors that influence students' programming skills: a case study from a Nigerian university. *International Journal of Quantitative Research in Education*, *3*(4), 277. DOI:10.1504/ijqre.2016.10003213

**Davidson, C. N., & Goldberg, D. T.** (2010). *The Future of Thinking: Learning Institutions in a Digital Age*. MIT Press.

**Davis, F. D.** (1986). *A technology acceptance model for empirically testing new end-user information systems: Theory and results*. Massachussetts Institute of Technology. DOI:oclc/56932490

_____ (1989). Perceived Usefulness, Perceived East of Use, and User Acceptance of Information Technology. *MIS Quarterly*, *13*(3), 319–340. DOI:10.1016/S0305-0483(98)00028-0

**Davis, F. D., Bagozzi, R. P., & Warshaw, P. R.** (1989). User Acceptance of Computer Technology: a Comparison of Two Theoretical Models. *Management Science*, *35*(8).

_____ (1992). Extrinsic and Intrinsic Motivation to Use Computers in the Workplace. *Journal of Applied Social Psychology*, *22*(14), 1111–1132. DOI:10.1111/j.1559-1816.1992.tb00945.x

**de Duve, T.** (1994). When Form Has Become Attitude - And Beyond. In S. Foster & N. de Ville (Eds.), *The Artist and the Academy: Issues in Fine Art Education and The Wider Cultural Context* (pp. 23–40). Southampton: John Hansard.

**Deleuze, G., & Guattari, F.** (1987). *A Thousand Plateaus: capitalism and schizophrenia*. (B. Massumi, Trans.). University of Minnesota.

**Denzin, N. K. (2004).** Symbolic interactionism. In U. Flick, E. Von Kardorff, & I. Steinke (Eds.), *A Companion to Qualitative Research* (pp. 81–87). Sage.

**DGEEC** (2018). Infocursos 2018. Retrieved from https://infocursos.mec.pt/bds.asp

_____ (2019). Infocursos 2019. Retrieved from https://infocursos.mec.pt/bds.asp

_____ (2020a). Inquérito ao Registo Biográfico de Docentes do Ensino Superior (REBIDES). Retrieved from www.dgeec.mec.pt/np4/inqueritos/rebides

_____ (2020b). Inscritos em estabelecimentos de Ensino Superior - 1995/96 a 2019/20. Retrieved from www.dgeec.mec.pt/np4/EstatVagasInsc/

_____ (2020c). Rede atual de Estabelecimentos do Ensino Superior. Retrieved from www.dgeec.mec.pt/np4/38/

**DGES** (2020). Cursos de Especialização Tecnológica. Retrieved from www.dges.gov.pt/pt/pagina/cursos-de-especializacao-tecnologica-cet

**Dietrich, F.** (1986). Visual Intelligence: The First Decade of Computer Art (1965-1975). *Leonardo*, *19*(2), 159–169.

**diSessa, A. A.** (1985). Learning About Knowing. *Children and Computers*, (28), 97–124.

_____ (2018). Computational Literacy and "The Big Picture" Concerning Computers in Mathematics Education. *Mathematical Thinking and Learning*, *20*(1), 3–31. DOI:10.1080/10986065.2018.1403544

**Donnelly, D., & Grushka, K.** (2015). The digital studio as a global education site. In *Contesting and Constructing International Perspectives in Global Education* (pp. 173–186). SensePublishers.

**Dreher, T.** (2014). History of Computer Art. Retrieved February 19, 2020, from http://iasl.uni-muenchen.de/links/GCA_Indexe.html

**Dreyfus, H.** (2008). *On the Internet*. Routledge.

**Dreyfus, H., & Kelly, S. D.** (2011). *All Things Shining: Reading the Western Classics to Find Meaning in a Secular Age*. Simon and Schuster.

**du Boulay, B.** (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, *2*(1), 57–73.

**Dutton, W. H.** (2002). Toward a Digital Academe. In W. Dutton & B. Loader (Eds.), *Digital Academe: New Media in Higher Education and Learning*. Routledge.

**Dutton, W. H., Rogers, E. M., & Jun, S.-H.** (1987). Diffusion and Social Impacts of Personal Computers. *Communication Research*, *14*(2), 219–250.

**Dwivedi, Y. K., Rana, N. P., Jeyaraj, A., Clement, M., & Williams, M. D.** (2017). Re-examining the Unified Theory of Acceptance and Use of Technology (UTAUT): Towards a Revised Theoretical Model. *Information Systems Frontiers*, 1–16. DOI:10.1007/s10796-017-9774-y

**Dziuban, C., & Moskal, P.** (2011). A course is a course is a course: Factor invariance in student evaluation of online, blended and face-to-face learning environments. *The Internet and Higher Education*, *14*(4), 236–241. DOI:10.1016/j.iheduc.2011.05.003

**Engelbart, D. C.** (1995). Toward augmenting the human intellect and boosting our collective IQ. *Communications of the ACM*, *38*(8), 30–32.

**Ettinger, L. F.** (1988). Art Education and Computing: Building a Perspective. *Studies in Art Education*, *30*(1), 53–62. DOI:10.1080/00393541.1988.11650702

**Falkner, K., & Sheard, J.** (2019). Pedagogic Approaches. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 445–480). Cambridge University Press. DOI:10.1017/9781108654555.016

**Fincher, S. A., & Robins, A. V.** (2019). An Important and Timely Field. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 1–8). Cambridge University Press. DOI:10.1017/9781108654555.001

**Fish, S.** (2010, October 11). The crisis of the humanities officially arrives. *New York Times*.

**Flick, U.** (2004a). Design and Process in Qualitative Research. In U. Flick, E. Von Kardorff, & I. Steinke (Eds.), *A Companion to Qualitative Research* (pp. 146–152). Sage.

_____ (2004b). Triangulation in Qualitative Research. In U. Flick, E. Von Kardorff, & I. Steinke (Eds.), *A Companion to Qualitative Research* (pp. 178–183). Sage.

**Flick, U., von Kardorff, E., & Steinke, I.** (2004). *A Companion to Qualitative Research*. Sage.

**Fonseca, M. P., & Encarnação, S.** (2013). *O Sistema de Ensino Superior em Portugal: Análises sectoriais, vol. I*. Lisboa: A3ES.

**Fraillon, J., Ainley, J., Schulz, W., Friedman, T., & Duckworth, D.** (2020). *Preparing for Life in a Digital World - IEA International Computer and Information Literacy Study 2018 International Report*. Springer International Publishing. DOI:10.1007/978-3-030-38781-5

**Fraillon, J., Ainley, J., Schulz, W., Friedman, T., & Gebhardt, E.** (2014). *Preparing for Life in a Digital Age - IEA International Computer and Information Literacy Study 2013 International Report*. Springer International Publishing. DOI:10.1007/978-3-319-14222-7

**Friães, R.** (2017a). *Artes do Espetáculo (Ciclo de Estudos Temáticos)*. Lisboa: A3ES.

_____ (2017b). *Design (Ciclo de Estudos Temáticos)*. Lisboa: A3ES.

**Gage, N. L.** (1991). The Obviousness of Social and Educational Research Results. *Educational Researcher*, *20*(1), 10–16. DOI:10.3102/0013189X020001010

**Gebhardt, E., Thomson, S., Ainley, J., & Hillman, K.** (2019). What Have We Learned About Gender Differences in ICT? In E. Gebhardt, S. Thomson, J. Ainley, & K. Hillman (Eds.), *Gender Differences in Computer and Information Literacy* (pp. 69–73). Springer. DOI:10.1007/978-3-030-26203-7_6

**Gilster, P.** (1997). *Digital Literacy*. John Wiley & Sons.

**Giovannelli, A.** (2017). Goodman's Aesthetics. *Stanford Encyclopedia of Philosophy*. Retrieved from https://plato.stanford.edu/archives/fall2017/entries/goodman-aesthetics/

**Gitelman, L.** (2006). Introduction: Media as Historical Subjects. In *Always Already New: Media History and the Data of Culture*. MIT Press.

**Gomes, A. de J., Mendes, A. J., & Marcelino, M. J.** (2014). Computer science education research: An overview and some proposals. In *Innovative Teaching Strategies and New Learning Paradigms in Computer Programming* (pp. 1–29). IGI Global. DOI:10.4018/978-1-4666-7304-5.ch001

**Goodman, L. A., & Kruskal, W. H.** (1979). *Measures of Association for Cross Classifications*. Springer.

**Goodyear, P., & Ellis, R. A.** (2008). University students' approaches to learning: rethinking the place of technology. *Distance Education*, *29*(2), 141–152. DOI:10.1080/01587910802154947

**Gouzouasis, P.** (2006). Technology as arts-based education: Does the desktop reflect the arts? *Arts Education Policy Review*, *107*(5), 3–9. DOI:10.3200/AEPR.107.5.3-9

**Graham, C. R.** (2006). Blended learning systems: Definition, current trends, and future directions. In C. R. Graham & C. J. Bonk (Eds.), *Handbook of blended learning: Global Perspectives, local designs* (pp. 3–21). San Francisco, CA: Pfeiffer Publishing.

**Gude, O.** (2004). Postmodern Principles: In Search of a 21st Century Art Education. *Art Education*, *57*(1), 6–14. DOI:10.1080/00043125.2004.11653528

**Guzdial, M.** (2019). Computing for Other Disciplines. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 584–605). Cambridge University Press. DOI:10.1017/9781108654555.020

**Guzdial, M., & du Boulay, B.** (2019). The History of Computing Education Research. In S. A. Fincher & A. V Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 11–39). Cambridge University Press. DOI:10.1017/9781108654555.002

**Halverson, E. R., & Sheridan, K. M.** (2014a). Arts education and the learning sciences. In R. K. Sawyer (Ed.), *The Cambridge Handbook of the Learning Sciences* (pp. 626–646). Cambridge University Press. DOI:10.1017/CBO9781139519526.037

_____ (2014b). The Maker Movement in Education. *Harvard Educational Review*, *84*(4), 495–504. DOI:10.17763/haer.84.4.34j1g68140382063

**Hamer, J.** (2006). Some experiences with the" contributing student approach". In *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education* (pp. 68–72).

**Hasan, A., & Laaser, W.** (2010). Higher Education Distance Learning in Portugal - State of the Art and Current Policy Issues. Retrieved from http://www.eurodl.org/materials/contrib/2010/Abrar_Laaser.pdf

**Hattie, J., & Timperley, H.** (2007). The power of feedback. *Review of Educational Research*, *77*(1), 81–112.

**Hausman, J. J.** (2000). Arts Education in the Twenty-First Century: A Personal View. *Arts Education Policy Review*, *102*(2), 17–18. DOI:10.1080/10632910009599980

**Heimbürger, A., & Multisilta, J**. (2004). On the era of educational hypermedia computing: From simple links to more complex structures. In J. Impagliazzo & J. A. N. Lee (Eds.), *IFIP World Computer Congress* (pp. 133–146). Kluwer Academic Publishers.

**Henderson, M., Selwyn, N., & Aston, R.** (2017). What works and why? Student perceptions of "useful" digital technology in university teaching and learning. *Studies in Higher Education*, *42*(8), 1567–1579. DOI:10.1080/03075079.2015.1007946

**Huang, H., Chen, S., & Huang, C.** (2013). Computer teaching based on experiments using a virtualisation platform. *World Transactions on Engineering and Technology Education*, *11*(4), 527–531.

**Hundhausen, C. D., Narayanan, N. H., & Crosby, M. E.** (2008). Exploring studio-based instructional models for computing education. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education* (pp. 392–396).

**Hunger, F.** (2019, August 7). And maybe most important rule for myself [Tweet]. Retrieved from twitter.com/databaseculture/status/1159201338112319491

**Hurme, P., & Jouhki, J.** (2017). We shape our tools, and thereafter our tools shape us. *Human Technology*, *13*(2), 145–148. DOI:10.17011/ht/urn.201711104209

**Hutchings, M., & Quinney, A.** (2015). The flipped classroom, disruptive pedagogies, enabling technologies and wicked problems: Responding to 'the bomb in the basement.' *Electronic Journal of E-Learning*, *13*(2), 105–118.

**IEEE** (2016). The Top Programming Languages 2016. Retrieved from https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2016

**INE** (2011). População residente, segundo os Censos, com o ensino superior completo em % da população residente: total e por sexo. Retrieved from www.pordata.pt/DB/Portugal/Ambiente+de+Consulta/Tabela/5818969

_____ (2020). Estimativas Anuais da População Residente. Retrieved from www.pordata.pt/DB/Portugal/Ambiente+de+Consulta/Tabela/5818986

**James, A. R.** (2013). Lego Serious Play: a three-dimensional approach to learning development. *Journal of Learning Development in Higher Education*, (6).

**Jenkins, H.** (2006). *Convergence Culture*. New York University Press.

**Johnson, B. R.** (2017). Pedagogy - teaching the new designer. In *Design Computing: an overview of an emergent field* (pp. 184–192). Routledge.

**Kay, A., & Goldberg, A.** (2003). Personal Dynamic Media. In N. Wardrip-Fruin & N. Montfort (Eds.), *The New Media Reader* (pp. 393–406). MIT Press. (Original work published 1977)

**Kelle, U., & Erzberger, C.** (2004). Qualitative and Quantitative Methods: Not in Opposition. In U. Flick, E. Von Kardorff, & I. Steinke (Eds.), *A Companion to Qualitative Research* (pp. 172–177). Sage.

**Kember, D., & McNaught, C.** (2007). *Enhancing university teaching: Lessons from research into award-winning teachers*. Routledge.

**Kember, S., & Zylinska, J.** (2012). *Life After New Media: mediation as a vital process*. MIT Press.

**Khamis, H. (2008).** Measures of association: How to choose? *Journal of Diagnostic Medical Sonography*, *24*(3), 155–162. DOI:10.1177/8756479308317006

**Khechine, H., Ndjambou, P., & Lakhal, S.** (2016). A meta-analysis of the UTAUT model : Eleven years later. *Canadian Journal of Administrative Sciences*, *33*, 138–152. DOI:10.1002/cjas.138

**King, G., & Sen, M.** (2013). The Troubled Future of Colleges and Universities. *PS: Political Science & Politics*, (617).

**Kirschner, P. A., & De Bruyckere, P.** (2017). The myths of the digital native and the multitasker. *Teaching and Teacher Education*. DOI:10.1016/j.tate.2017.06.001

**Kirschner, P. A., & van Merriënboer, J.** (2013). Do Learners Really Know Best? Urban Legends in Education. *Educational Psychologist*, *48*(3), 169–183. DOI:10.1080/00461520.2013.804395

**Knochel, A. D., & Patton, R. M.** (2015). If Art Education Then Critical Digital Making: Computational Thinking and Creative Code. *Studies in Art Education*, *57*(1), 21–38.

**Krajcik, J. S., & Shin, N.** (2014). Project-Based Learning. In R. K. Sawyer (Ed.), *The Cambridge Handbook of the Learning Sciences* (pp. 275–298). Cambridge University Press.

**Krathwohl, D. R.** (2002). A Revision of Bloom's Taxonomy: An Overview. *Theory Into Practice*, *41*(4), 212–218. DOI:10.1207/s15430421tip4104_2

**Kreijns, K., Vermeulen, M., Kirschner, P. A., & Buuren, H. Van.** (2013). Adopting the Integrative Model of Behaviour Prediction to explain teachers' willingness to use ICT: a perspective for research on teachers' ICT usage in pedagogical practices. *Technology, Pedagogy and Education*, *22*(1). DOI:10.1080/1475939X.2012.754371

**Krosnick, J. A., Judd, C. M., & Wittenbrink, B.** (2005). The Measurement of Attitudes. In D. Albarracín, B. T. Johnson, & M. P. Zanna (Eds.), *The Handbook of Attitudes* (pp. 21–73). Lawrence Erlbaum Associates.

**Kurkovsky, S.** (2015). Teaching software engineering with LEGO Serious Play. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 213–218).

**Kurzweil, R.** (2015). Reduction in Watts per MIPS. Retrieved from http://singularity.com/charts/page129.html

**Lakhal, S., & Meyer, F.** (2019). Blended Learning. In A. Tatnall (Ed.), *Encyclopedia of Education and Information Technologies* (Vol. 64, pp. 1–6). Springer International Publishing. DOI:10.1007/978-3-319-60013-0_41-1

**Lancaster, T., Robins, A. V., & Fincher, S. A.** (2019). Assessment and Plagiarism. In S. A. Fincher & A. V Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 414–444). Cambridge University Press. DOI:10.1017/9781108654555.014

**Latchem, C.** (2014). Opening up the educational technology research agenda. *British Journal of Educational Technology*, *45*(1), 3–11. DOI:10.1111/bjet.12122

**Latour, B.** (1991). Technology Is Society Made Durable. In J. Law (Ed.), *A Sociology of Monsters: Essays on Power, Technology and Domination* (pp. 103–131). Routledge.

**Lau, W. W. F., & Yuen, A. H. K.** (2014). Teaching in Visual Programming Environments. In M. Khosrow-Pour (Ed.), *Encyclopedia of Information Science and Technology* (3rd ed., pp. 2600–2604). IGI Global. DOI:10.4018/978-1-60566-026-4

**Leddy, T.** (2019). Dewey's Aesthetics. *The Stanford Encyclopedia of Philosophy*. Retrieved from https://plato.stanford.edu/archives/win2019/entries/dewey-aesthetics/

**Lefebvre, H.** (2014). *Critique of Everyday Life: The Three-Volume Text* (eBook). Verso Books. (Original work published 1981)

**Leu, D. J., Kinzer, C. K., Coiro, J., Castek, J., & Henry, L. A.** (2017). New Literacies: A Dual-Level Theory of the Changing Nature of Literacy, Instruction, and Assessment. *Journal of Education*, *197*(2). DOI:10.1177/002205741719700202

**Levin, G.** (2009, February). Art that looks back at you [Video]. Retrieved from www.ted.com/talks/golan_levin_art_that_looks_back_at_you

**Levin, G., & Brain, T.** (2021). *Code as Creative Medium: A Handbook for Computational Art and Design*. MIT Press.

**Levine, L.** (2011). The Great American Art Machine. In F. Allen (Ed.), *Education (Whitechapel: Documents of Contemporary Art)* (pp. 54–56). MIT. (Original work published 1971)

**Lewis, C. C., Fretwell, C. E., Ryan, J., & Parham, J. B.** (2013). Faculty Use of Established and Emerging Technologies in Higher Education: A Unified Theory of Acceptance and Use of Technology Perspective. *International Journal of Higher Education*, *2*(2), 22–34. DOI:10.5430/ijhe.v2n2p22

**Lewis, C. M., Shah, N., & Falkner, K.** (2019). Equity and Diversity. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 481–510). Cambridge University Press. DOI:10.1017/9781108654555.017

**Lieser, W.** (2009). *Arte Digital*. (H. Morbey, Trans.). H. F. Ullmann.

**Lishinski, A., & Yadav, A.** (2019). Motivation, Attitudes, and Dispositions. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 801–826). Cambridge University Press. DOI:10.1017/9781108654555.029

**Lista de instituições de ensino superior de Portugal** (2020). In *Wikipedia*. https://pt.wikipedia.org/w/index.php?title=Lista_de_instituições_de_ensino_superior_de_Portugal&oldid=59243284.

**Livingstone, S.** (2014). Critical reflections on the benefits of ICT in education. *Oxford Review of Education*, *28*(1), 9–24. DOI:10.1080/03054985.2011.577938

**Lovink, G.** (2008). *Zero Comments: Blogging and Critical Internet Culture*. Routledge.

_____ (2012). What Is the Social in Social Media ? *E-Flux*, (40).

**Luke, C.** (2000). Cyber-schooling and technological change: Multiliteracies for new times. In B. Cope & M. Kalantzis (Eds.), *Multiliteracies: Literacy learning and the design of social futures* (pp. 67–88). Routledge.

**Madden, T. J., Ellen, P. S., & Ajzen, I.** (1992). A comparison of the theory of planned behavior and the theory of reasoned action. *Personality and Social Psychology Bulletin*, *18*(1), 3–9.

**Maeda, J.** (1999). *Design by Numbers*. MIT Press.

_____ (2000). *Maeda @ Media*. Thames & Hudson.

**Manovich, L.** (2013). *Software Takes Command*. Bloomsbury.

**Marangunić, N., & Granić, A.** (2015). Technology acceptance model: a literature review from 1986 to 2013. *Universal Access in the Information Society*, *14*(1), 81–95. DOI:10.1007/s10209-014-0348-1

**Marton, F., & Säljö, R.** (1976). On Qualitative Differences in Learning: I-Outcome and Process. *British Journal of Educational Psychology*, *46*(1). DOI:10.1111/j.2044-8279.1976.tb02980.x

**Marzoni, A.** (2020, March). The Trolls of Academe. *The Baffler*. Retrieved from https://thebaffler.com/salvos/the-trolls-of-academe-marzoni

**Mateas, M.** (2005). Procedural Literacy: Educating the new media practitioner. *On the Horizon*, *13*(2), 101–111. DOI:10.1108/10748120510608133

**McAllister, G., & Alexander, S.** (2009). Key aspects of teaching and learning in computing science. In H. Fry, S. Ketteridge, & S. Marshall (Eds.), *A Handbook for Teachning and Learning in Higher Education* (3rd ed., pp. 282–290). Taylor & Francis.

**McCormack, J., & D'Inverno, M.** (2012). Computers and Creativity: The Road Ahead. In J. McCormack & M. D'Inverno (Eds.), *Computers and Creativity* (pp. 421–425). Springer. DOI:10.1007/978-3-642-31727-9

**McFarland, D. J., & Hamilton, D.** (2006). Adding contextual specificity to the technology acceptance model. *Computers in Human Behavior*, *22*, 427–447. DOI:10.1016/j.chb.2004.09.009

**McLean, C. A., & Wiggins, G.** (2012). Computer Programming in the Creative Arts. In J. McCormack & M. D'Inverno (Eds.), *Computers and Creativity* (pp. 235–252). Heidelberg: Springer. DOI:10.1007/978-3-642-31727-9

**McLuhan, M.** (2008). *Understanding Media: The Extensions of Man*. Routledge. (Original work published 1964)

**Meeken, L.** (2012). System Opacity and Student Agency in the New Media Landscape. *Voke*, (pilot issue).

**Meinefeld, W.** (2004). Hypotheses and Prior Knowledge in Qualitative Research. In U. Flick, E. Von Kardorff, & I. Steinke (Eds.), *A Companion to Qualitative Research* (pp. 153–158). Sage.

**Merkens, H.** (2004). Selection Procedures, Sampling, Case Construction. In U. Flick, E. Von Kardorff, & I. Steinke (Eds.), *A Companion to Qualitative Research* (pp. 165–171). Sage.

**Merriam, S., & Tisdell, E.** (2016). *Qualitative Research: A guide to Design and Implementation* (4th ed.). Jossey-Bass. DOI:10.1192/bjp.112.483.211-a

**Mertens, D. M.** (2010). *Research and Evaluation in Education and Psychology* (3rd ed.). Sage Publications.

**Meyers, E. M., Erickson, I., & Small, R. V.** (2013). Digital literacy and informal learning environments: an introduction. *Learning, Media and Technology*, *38*(4), 355–367. DOI:10.1080/17439884.2013.783597

**Milman, N. B.** (2012). The flipped classroom strategy: What is it and how can it best be used? *Distance Learning*, *11*(4).

**Montfort, N.** (2016). *Exploratory Programming for the Arts and Humanities*. MIT Press.

**Moore, G. C., & Benbasat, I.** (1996). Integrating Diffusion of Innovations and Theory of Reasoned Action models to predict utilization of information technology by end-users. In K. Kautz (Ed.), *Diffusion and Adoption of Information Technology* (pp. 132–146). Springer.

**Morais, E., Morais, C., & Paiva, J. C.** (2014). Myths and Realities of E-Learning: exploratory survey of higher education students. *E-Learning and Digital Media*, *11*(3), 300. DOI:10.2304/elea.2014.11.3.300

_____ (2018a). Computer Programming Acceptance By Students In Higher Arts And Design Education. In L. G. Chova, A. L. Martínez, & I. C. Torres (Eds.), *ICERI2018 Proceedings* (pp. 2058–2065). Seville: IATED.

_____ (2018b). The Perspective of Higher Arts and Design Educators on Teaching Computer Programming. In L. G. Chova, A. L. Martínez, & I. C. Torres (Eds.), *ICERI2018 Proceedings* (pp. 2043–2050). Seville: IATED.

_____ (2019). Higher Arts and Design Students' Attitudes Towards Learning Computer Programming. In M. Carmo (Ed.), *Education and New Developments 2019* (Vol. II, pp. 115–119). Porto: InScience Press.

**Morozov, E.** (2011). *The Net Delusion: How not to liberate the world*. Penguin Press.

**Morton, T.** (2013). *Hyperobjects: Philosophy and Ecology after the End of the World*. University of Minnesota.

**Muncy, J. A.** (2014). Blogging for Reflection: The Use of Online Journals to Engage Students in Reflective Learning. *Marketing Education Review*, *24*(2), 101–114. DOI:10.2753/mer1052-8008240202

**Murphy, P.** (2015). *Universities and Innovation Economies: The creative wasteland of post-industrial society*. Ashgate Publishing.

**Nake, F.** (1971). There Should be No Computer Art. *Bulletin of the Computer Arts Society*, (October).

_____ (2012). Construction and Intuition: Creativity in Early Computer Art. In J. McCormack & M. D'Inverno (Eds.), *Computers and Creativity* (pp. 61–95). Heidelberg: Springer. DOI:10.1007/978-3-642-31727-9

**Nebel, S., Schneider, S., & Rey, G. D.** (2016). Mining learning and crafting scientific experiments: a literature review on the use of Minecraft in education and research. *Journal of Educational Technology & Society*, *19*(2), 355–366.

**Nelson, T.** (1987). *Computer Lib, Dream Machines*. Microsoft Press. (Originally published 1974)

**Noble, J.** (2009). *Programming interactivity: a designer's guide to Processing, Arduino, and OpenFrameworks*. O'Reilly.

**Noll, A. M.** (2000). Technology and the Future of the University: a Sober View. *Information, Communication & Society*, *3*(4), 645–647. DOI:10.1080/13691180010002170

**Norman, D.** (1988). *The Design of Everyday Things*. MIT Press.

**Nóvoa, A.** (2018). A modernização das universidades: Memórias contra o tempo. *Revista Portuguesa de Educação*, *31*, 10–25. DOI:10.21814/rpe.15076

**Nunnally, J.** (1978). *Psychometric methods*. New York: McGraw-Hill.

**O'Grady, M.** (2018). The Flipped Classroom. In J. Carter, M. O'Grady, & C. Rosen (Eds.), *Higher Education Computer Science - a Manual of Practical Approaches* (pp. 29–49). Springer.

**OECD** (2007). *Revised Field Of Science And Technology (FOS) Classification In The Frascati Manual*.

**Oliveira, M.** (2017). O Ensino Superior Artístico de Belas Artes: perspectivas e dilemas. *Palíndromo*, *9* (19), 67–86. DOI:10.5965/2175234609192017067

**Overby, A., & Jones, B. L.** (2015). Virtual LEGOs: Incorporating Minecraft into the art education curriculum. *Art Education*, *68*(1), 21–27.

**Paas, F., Renkl, A., & Sweller, J.** (2003). Cognitive Load Theory and Instructional Design: recent developments. *Educational Psychologist*, *38*. DOI:10.1207/S15326985EP3801_1

**Paik, N. J.** (2003). Cybernated Art. In N. Wardrip-Fruin & N. Montfort (Eds.), *The New Media Reader* (pp. 227–228). MIT Press. (Original work published 1964)

**Paiva, J. C., Morais, C., Costa, L. A., & Pinheiro, A.** (2015). The shift from "e-learning" to "learning": Invisible technology and the dropping of the "e." *British Journal of Educational Technology*, *47*(2), 226–238. DOI:10.1111/bjet.12242

**Palhares, S.** (2018). O Ensino da Arte Está em Perigo? In J. P. Queiroz & R. A. de Oliveira (Eds.), *Arte e Ensino: Propostas de Resistência* (pp. 153–161). Lisboa: Centro de Investigação e Estudos em Belas-Artes.

**Papert, S.** (1980). *Mindstorms: Computers, children, and powerful ideas*. Basic Books.

_____ (1985). Different Visions of LOGO. *Computers in the Schools*, *2*(2–3), 3–8.

**Parameswaran, S., Kishore, R., & Li, P.** (2015). Within-study measurement invariance of the UTAUT instrument: An assessment with user technology engagement variables. *Information and Management*, *52*(3), 317–336. DOI:10.1016/j.im.2014.12.007

**Parker, K. R., Beard, D. V., & Davey, B.** (2019). Programming Language Selection for University Courses. In A. Tatnall (Ed.), *Encyclopedia of Education and Information Technologies* (pp. 1–12). Springer. DOI:10.1007/978-3-319-60013-0_131-1

**Parsons, D., & Haden, P.** (2006). Parson's programming puzzles: a fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52* (pp. 157–163).

**Pea, R. D.** (1986). Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research*, *2*(1), 25–36.

**Peppler, K. A.** (2010a). Media Arts: Arts Education for a Digital Age. *Teachers College Record*, *112*(8), 2118–2153.

_____ (2010b). The New Fundamentals: Introducing Computation into Arts Education. In E. P. Clapp & M. J. Bellino (Eds.), *20Under40: Reinventing the Arts and Arts Education for the 21st Century*. Bloomington, IN: AuthorHouse.

**Perkins, D. N.** (1985). The Fingertip Effect: How Information-Processing Technology Shapes Thinking. *Educational Researcher*, *14*(7), 11–17. DOI:10.3102/0013189X014007011

**Perkins, D. N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R.** (1986). Conditions of learning in novice programmers. *Journal of Educational Computing Research*, *2*, 37–55.

**Petre, M.** (1995). Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming. *Communications of the ACM*, *38*(6), 33–44. DOI:10.1145/203241.203251

**Pinheiro, M. M., Machado, I., & Sarrico, C. S.** (2018). *Caracterização do Corpo Docente das Áreas de Educação e Formação do Ensino Superior Português*. Lisboa: A3ES.

**PORDATA** (2019a). Agregados domésticos privados com computador, com ligação à Internet e com ligação à Internet através de banda larga. Retrieved from www.pordata.pt/DB/Portugal/Ambiente+de+Consulta/Tabela/5820598

_____ (2019b). Computadores com ligação à Internet em % do total de computadores no ensino básico e secundário. Retrieved from www.pordata.pt/DB/Portugal/Ambiente+de+Consulta/Tabela/5820599

_____ (2019c). Indivíduos com 16 e mais anos que utilizam computador e Internet em % do total de indivíduos: por grupo etário e por sexo. Retrieved from www.pordata.pt/DB/Portugal/Ambiente+de+Consulta/Tabela/5820600

_____ (2020a). Alunos matriculados no ensino superior: por subsistema e tipo de ensino. Retrieved from www.pordata.pt/DB/Portugal/Ambiente+de+Consulta/Tabela/5818938

_____ (2020b). Alunos matriculados no ensino superior: total e por nível de formação. Retrieved from www.pordata.pt/DB/Portugal/Ambiente+de+Consulta/Tabela/5818937

_____ (2020c). Diplomados no ensino superior: total e por nível de formação. Retrieved from www.pordata.pt/Portugal/Diplomados+no+ensino+superior+total+e+por+área+de+educação+e+formação-222-4246

**PORDATA** (2020d). Estabelecimentos de ensino superior: total e por subsistema de ensino. Retrieved from www.pordata.pt/DB/Municipios/Ambiente+de+Consulta/Mapa/5818788

_____ (2020e). População residente com 30 a 34 anos com o ensino superior completo em % da população residente: total e por sexo. Retrieved from www.pordata.pt/DB/Portugal/Ambiente+de+Consulta/Tabela/5818970

**PORTUGAL** (2005). Portaria 256/2005 - Classificação Nacional das Áreas de Educação e Formação. *Diário Da República 53/2005, Série I-B*.

_____ (2006). Decreto-Lei 74/2006 - Regime jurídico dos graus e diplomas do ensino superior. *Diário Da República 60/2006, Série I-A*.

**Postman, N.** (1992). *Tecnopolia*. (J. Pinheiro, Trans.). Difusão Cultural.

**Potts, A.** (2008). Researching academic staff: The case of colleges and new universities. In P. Sikes & A. Potts (Eds.), *Researching Education from the Inside: Investigations from within* (pp. 159–174). Routledge. DOI:10.4324/9780203932353

**Prensky, M.** (2001). Digital Natives, Digital Immigrants Part 1. *On the Horizon*, *9*(5), 1–6. DOI:10.1108/10748120110424816

**Quaranta, D.** (2015). Situating Post Internet. In V. Catricalà (Ed.), *Media Art. Toward a New Defnition of Art in the Age of Technology*. Gli Ori, Pistoia.

**Rad, M. S., Nilashi, M., & Dahlan, H. M.** (2018). Information technology adoption: a review of the literature and classification. *Universal Access in the Information Society*, *17*(2), 361–390. DOI:10.1007/s10209-017-0534-z

**Radclyffe-Thomas, N.** (2008). White Heat or Blue Screen - Digital Technology in Art and Design Education. *International Journal of Art & Design Education*, *27*(2), 158–167.

**Read, H.** (1974). *Education Through Art*. Pantheon Books. (Original work published 1943)

**Reas, C.** (2004). {Software} Structures. Retrieved from https://artport.whitney.org/commissions/softwarestructures2016/text.html

**Reas, C., & Fry, B.** (2018). A Modern Prometheus - The History of Processing. Retrieved June 1, 2018, from https://medium.com/processing-foundation/a-modern-prometheus-59aed94abe85

**Reas, C., McWilliams, C., & Barendse, J.** (2010). *Form + Code in Design, Art, and Architecture*. Princeton Architectural Press.

**Renkl, A. (2017).** Learning from worked-examples in mathematics: students relate procedures to principles. *ZDM*, *49*(4), 571–584.

**Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., … Kafai, Y.** (2009). Scratch: Programming for all. *Communications of the ACM*, *52*(11), 60–67. DOI:10.1145/1592761.1592779

**Rheingold, H.** (2000). *Tools for Thought: The history and future of mind-expanding technology*. MIT Press. (Original work published 1985)

**Richardson, J. T. E.** (2011). Eta squared and partial eta squared as measures of effect size in educational research. *Educational Research Review*, *6*(2), 135–147. DOI:10.1016/j.edurev.2010.12.001

**Ritchie, H., & Roser, M.** (2017). Technology Adoption. Retrieved from https://ourworldindata.org/technology-adoption

**Rizwan, O.** (2015). Bring on the Real Computer Revolution. *Intersect*, *8*(2).

**Robins, A. V.** (2019). Novice Programmers and Introductory Programming. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 327–376). Cambridge University Press. DOI:10.1017/9781108654555.013

**Robins, A. V., Margulieux, L. E., & Morrison, B. B.** (2019). Cognitive Sciences for Computing Education. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 231–275). Cambridge University Press. DOI:10.1017/9781108654555.010

**Robins, A. V., Rountree, J., & Rountree, N.** (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137–172. DOI:10.1076/csed.13.2.137.14200

**Roser, M., & Ritchie, H.** (2013). Technological Progress. Retrieved from https://ourworldindata.org/technological-progress

**Rubin, M. J.** (2013). The effectiveness of live-coding to teach introductory programming. *SIGCSE 2013 - Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, 651–656. DOI:10.1145/2445196.2445388

**Rushkoff, D.** (2010). *Program or Be Programmed: ten commands for a digital age*. Or Books.

**Sack, W.** (2019). *The Software Arts*. MIT Press.

**Santos, C.** (2011). *História da Universidade do Porto*. Universidade do Porto.

**Santos, R., & Azevedo, J.** (2016). Profiling Internet Use of Portuguese Higher Education Students. In *Handbook of Research on Engaging Digital Natives in Higher Education Settings* (pp. 46–71). IGI Global. DOI:10.4018/978-1-5225-0039-1.ch003

**Sassen, S.** (2002). Towards a Sociology of Information Technology. *Current Sociology*, *50*(3), 365–388. DOI:10.1177/0011392102050003005

**Scott, K. A., Sheridan, K. M., & Clark, K.** (2014). Culturally responsive computing: a theory revisited. *Learning, Media and Technology*, *9884*(January), 1–25. DOI:10.1080/17439884.2014.924966

**Selwyn, N.** (2002). Learning to Love the Micro: The discursive construction of "educational" computing in the UK, 1979-89. *British Journal of Sociology of Education*, *23*, 427–443.

_____ (2013). Digital technologies in universities: problems posing as solutions? *Learning, Media and Technology*, *38*(1), 1–3. DOI:10.1080/17439884.2013.759965

_____ (2015). Technology and Education — Why It's Crucial to be Critical. In C. Bigum, S. Bulfin, & N. F. Johnson (Eds.), *Critical Perspectives on Technology and Education* (pp. 245–255). Palgrave Macmillan.

_____ (2016). The Dystopian Futures. In N. Rushby & D. W. Surry (Eds.), *The Wiley Handbook of Learning Technology* (pp. 542–556). John Wiley & Sons. DOI:10.1002/9781118736494.ch28

**Shephard, K.** (2019). Higher Education Pedagogy. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 276–291). Cambridge University Press. DOI:10.1017/9781108654555.011

**Shiffman, D.** (2008). *Learning Processing*. Morgan Kaufmann.

**Shirky, C.** (2012). Napster, Udacity, and the Academy. Retrieved from https://web.archive.org/web/20160402012727/http://www.shirky.com/weblog/2012/11/napster-udacity-and-the-academy/

**Shneiderman, B.** (1983). Direct manipulation. *IEEE Computer*, *16*(8), 57–69. DOI:10.1145/800276.810991

**Shreeve, A.** (2011). The Way We Were? Signature pedagogies under threat. In E. Bohemia, B. B. de Mozotaand, & L. Collina (Eds.), *1st International Symposium for Design Education Researchers* (Vol. 2, pp. 112–125). Paris: Cumulus Association and DRS.

**Shreeve, A., Sims, E., & Trowler, P.** (2010). "A kind of exchange": learning from art and design teaching. *Higher Education Research & Development*, *29*(2), 125–138. DOI:10.1080/07294360903384269

**Shreeve, A., Wareing, S., & Drew, L.** (2009). Key aspects of teaching and learning in the visual arts. In H. Fry, S. Ketteridge, & S. Marshall (Eds.), *A Handbook for Teachning and Learning in Higher Education* (3rd ed., pp. 345–363). Taylor & Francis.

**Sikes, P., & Potts, A.** (2008). Introduction: What are we talking about? And why? In P. Sikes & A. Potts (Eds.), *Researching Education from the Inside: Investigations from within* (pp. 3–11). Routledge. DOI:10.4324/9780203932353

**Sim, J. W. S., & Hew, K. F.** (2010). The use of weblogs in higher education settings: A review of empirical research. *Educational Research Review*, *5*(2), 151–163. DOI:10.1016/j.edurev.2010.01.001

**Snyder, I., & Bulfin, S.** (2007). Digital Literacy: what it means for Arts Education. In L. Bresler (Ed.), *International Handbook of Research in Arts Education* (pp. 1297–1310). Springer.

**Souleles, N., Savva, S., Watters, H., & Annesley, A.** (2017). Perceptions of art and design faculty on the instructional value of iPads. *Technology, Pedagogy and Education*, *26*(2), 211–223. DOI:10.1080/1475939X.2016.1199390

**Sousa Santos, B.** (2008). A Universidade no Século XXI: Para uma reforma democrática e emancipatória da Universidade. In *A Universidade no Século XXI: Para uma Universidade Nova*. Coimbra: Almedina.

**Sroczkowski, P.** (2020). GitHub languages statistics. Retrieved from https://github.com/oprogramador/github-languages

**Stager, G. S.** (2017). A Modest Proposal [Weblog]. Retrieved from http://stager.tv/?p=4153

**Star, S. L.** (1996). Working together: Symbolic interactionism, activity theory, and information systems. In *Cognition and Communication at Work* (pp. 296–318). Cambridge University Press. DOI:10.1017/CBO9781139174077.013

**Steenkamp, J. B. E. M., & Baumgartner, H.** (1998). Assessing measurement invariance in cross-national consumer research. *Journal of Consumer Research*, *25*(1), 78–90. DOI:10.1086/209528

**Sweller, J.** (1994). Cognitive Load Theory, Learning Difficulty, and Instructional Design. *Learning and Instruction*, *4*, 295–312. DOI:10.1016/0959-4752(94)90003-5

**Taber, K. S.** (2014). Methodological Issues in Science Education Research: A Perspective from the Philosophy of Science. In *International Handbook of Research in History, Philosophy and Science Teaching* (pp. 1839–1893). Springer Netherlands. DOI:10.1007/978-94-007-7654-8_57

**Taber, K. S.** (2018). The Use of Cronbach's Alpha When Developing and Reporting Research Instruments in Science Education. *Research in Science Education*, *48*(6), 1273–1296. DOI:10.1007/s11165-016-9602-2

**Tan, M.** (2019). Makerspaces. In A. Tatnall (Ed.), *Encyclopedia of Education and Information Technologies* (pp. 1–9). Springer. DOI:10.1007/978-3-319-60013-0_114-1

**Taylor, S., & Todd, P. A.** (1995). Understanding information technology usage: A test of competing models. *Information Systems Research*. DOI:10.1287/isre.6.2.144

**Tenenberg, J.** (2019). Qualitative methods for computing education. In *The Cambridge Handbook of Computing Education Research* (pp. 173–207). Cambridge University Press.

**Thompson, R. L., Higgins, C. A., & Howell, J. M.** (1991). Personal Computing: Toward a Conceptual Model of Utilization. *MIS Quarterly*, *15*(1), 125–143.

**Tranquada, S., & Correia, N.** (2018). The Perseverance of Gender Stereotype. *Proceedings of British HCI*.

**UNESCO** (1988). International standard nomenclature for fields of science and technology. Retrieved from https://unesdoc.unesco.org/ark:/48223/pf0000082946

_____ (1997). International Standard Classification of Education (ISCED). Retrieved from http://uis.unesco.org/en/topic/international-standard-classification-education-isced

_____ (2013). International Standard Classification of Education (ISCED) - Fields of education and training 2013.

**van Dijk, J.** (2006). *The Network Society: social aspects of new media*. Sage.

**van Mensvoort, K. M.** (2013). How conditional design changed the world. In *Conditional Design Workbook*. Valiz Publishers.

**van Merriënboer, J.** (1990). Strategies for programming instruction in high school: Program completion vs. program generation. *Journal of Educational Computing Research*, *6*(3), 265–285.

**Vee, A.** (2013). Understanding Computer Programming as a Literacy. *Literacy in Composition Studies*, *1*(2), 42–64.

**Venkatesh, V., & Bala, H.** (2008). Technology acceptance model 3 and a research agenda on interventions. *Decision Sciences*, *39*(2), 273–315. DOI:10.1111/j.1540-5915.2008.00192.x

**Venkatesh, V., Brown, S. A., & Bala, H.** (2013). Bridging the Qualitative-Quantitative Divide: Guidelines for Conducting Mixed Methods Research in Information Systems. *MIS Quarterly*, *37*(1), 21–54. DOI:10.25300/MISQ/2013/37.1.02

**Venkatesh, V., & Davis, F. D.** (2000). A Theoretical Extension of the Technology Acceptance Model: Four Longitudinal Field Studies. *Management Science*, *46*(2), 186–204. DOI:10.1287/mnsc.46.2.186.11926

**Venkatesh, V., Morris, M. G., Davis, G. B., & Davis, F. D.** (2003). User acceptance of information technology: Toward a unified view. *MIS Quarterly*, *27*(3).

**Venkatesh, V., Thong, J. Y. L., & Xu, X.** (2012). Consumer Acceptance and Use of Information Technology: extending the unified theory of acceptance and use of technology. *MIS Quarterly*, *36*(1), 157–178.

**Venkatesh, V., Thong, J. Y. L., & Xu, X.** (2016). Unified Theory of Acceptance and Use of Technology: A Synthesis and the Road Ahead. *Journal of the Association for Information Systems*, *17*(5), 328–376.

**Victor, B.** (2012). Learnable Programming. Retrieved from http://worrydream.com/LearnableProgramming/

**Viola, B.** (2003). Will There Be Condominiums in Data Space? In N. Wardrip-Fruin & N. Montfort (Eds.), *The New Media Reader* (pp. 464–470). MIT Press. (Original work published 1982)

**Vuorikari, R., Punie, Y., Carretero, S., & Van Den Brande, L.** (2016). DigComp 2.0: The Digital Competence Framework for Citizens. Publications Office of the European Union. DOI:10.2791/11517

**Watts, L. S., & Blessinger, P.** (2017). The future of creative learning. In L. S. Watts & P. Blessinger (Eds.), *Creative Learning in Higher Education - International perspectives and approaches* (pp. 213–230). Routledge. DOI:10.4324/9781315659473

**Weintrop, D., & Wilensky, U.** (2015). To block or not to block, that is the question. In *Proceedings of the 14th International Conference on Interaction Design and Children* (pp. 199–208). New York: ACM. DOI:10.1145/2771839.2771860

**Weston, R., & Gore, P. A.** (2006). A Brief Guide to Structural Equation Modeling. *The Counseling Psychologist*, *34*(5), 719–751. DOI:10.1177/0011000006286345

**Wiener, N.** (2003). Men, Machines, and the World About. In N. Wardrip-Fruin & N. Montfort (Eds.), *The New Media Reader* (pp. 67–72). MIT Press. (Original work published 1954)

**Wilensky, U., & Papert, S.** (2010). Restructurations : Reformulating knowledge disciplines through new representational forms. In *Constructionism 2010*. Paris.

**Wilks, J., Cutcher, A., & Wilks, S.** (2012). Digital Technology in the Visual Arts Classroom: an [un]easy partnership. *Studies in Art Education*, *54*(1), 54–65.

**Williams, M. D., Rana, N. P., & Dwivedi, Y. K.** (2015). The unified theory of acceptance and use of technology (UTAUT): a literature review. *Journal of Enterprise Information Management*, *28*(3), 443–488. DOI:10.1108/JEIM-09-2014-0088

**Williams, R.** (2003). The technology and the society. In N. Wardrip-Fruin & N. Montfort (Eds.), *The New Media Reader* (pp. 291–300). MIT Press. (Original work published 1972)

**Wing, J. M.** (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35. DOI:10.1145/1118178.1118215

**Yurdugül, H.** (2008). Minimum sample size cronbach's coefficient alpha: A Monte-Carlo study. *Hacettepe Egitim Dergisi*, (35), 397–405.

# LIST OF ANNEXES ON THE CD

*Data collected throughout our studies*

Files are redacted in order to preserve participants' anonymity:

| | |
|---|---|
| **Interview transcripts** | *PDF document* |
| **Survey questionnaires** | *PDF document* |
| **Raw survey responses** | *zipped CSV files* |

| | |
|---|---|
| *Communications related to our studies* | *PDF documents* |

*(this page was intentionally left blank)*