FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

A Visual Editor for Creating Living Software Architecture Documentation

Ana Matilde Barra

Mestrado em Engenharia Informática e Computação

Supervisor: Prof. Filipe Correia

Co-Supervisor: Prof. Paulo Queiroz

July 26, 2024

© Ana Matilde Barra, 2024

A Visual Editor for Creating Living Software Architecture Documentation

Ana Matilde Barra

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

President: Prof. Nuno Flores
Examiner: Prof. Neil Harrison
Supervisor: Prof. Filipe Correia

July 26, 2024

Abstract

Visual diagramming tools are essential to ensure software documentation understanding and consistency, whether manually or automatically generated.

The main goal of this thesis is to improve the Developer Experience (DevEx) in a visual editor for software architecture diagrams, contributing to making documentation easier to keep updated and consistent and ensuring it retains its value over time. To achieve that, the visual editor must allow for traceable changes in the automatically generated diagrams and customization features like, for example, rearranging the diagram components.

Reviewing past literature regarding the best practices in visual diagramming tools for software documentation and DevEx or User Experience (UX) in these types of editors makes it possible to find improvement techniques worth exploring. Ensuring that changes made are easily recognizable, making errors and next steps easily identifiable, and offering a sense of control, flexibility, and customization are examples of practices and improvements that may increase DevEx in visual editors. *Infragenie*, an automatic architecture diagram-generating tool developed in a previous thesis of the Masters in Informatics and Computing Engineering (M.EIC), will be used to apply the findings and test their efficacy.

Research questions are defined to try to understand the positive effects of using a visual editor instead of a textual editor for editing automatically generated diagrams. The community's acceptance of the tool and its usability are also relevant when determining the improvements visual editors present compared to textual editors.

An empirical study supported by an experiment with a control group and an experimental group allowed for the collection of the relevant data needed to evaluate the developed visual editor prototype. Participants are asked to complete six tasks and answer questions regarding the Technology Acceptance Model (TAM), System Usability Scale (SUS), and the importance of specific functionalities and standards. The results are analyzed to draw conclusions on the behavior of the visual editor when compared to a text-based alternative.

The prototype developed is depicted by the experiment participants as an intuitive, useful, easy-to-use, and not cumbersome tool. With the conclusions drawn from this study, we can safely say that visual model editors are worthy alternatives to text-based diagram editors. These editors, in turn, increase the documentation quality, allowing for better and more understandable projects. Software projects that are easier to comprehend make it easier to share knowledge more clearly, contributing to a more open and well-informed software development community.

Keywords: Developer Experience, Usability, Visual Editors, Software Documentation, Documentation Consistency and Maintenance.

Resumo

Ferramentas de visualização de diagramas são essenciais para garantir a compreensão e consistência da documentação de *software*, quer esta seja gerada manual ou automaticamente.

O principal objetivo desta tese é melhorar a *Developer Experience (DevEx)* num editor visual para diagramas arquiteturais de *software*, contribuindo para tornar a documnetação mais facilmente atualizável e consistence e garantir que mantém o seu valor com a passagem do tempo. Para alcançar isso, o editor visual deve permitir registo de alterações nos diagramas automaticamente gerados e também funcionalidades de personalização como, por exemplo, reorganizar os componentes do diagrama.

A revisão de literatura passada sobre as melhores práticas em ferramentas visuais de criação de diagramas para documentação de *software* e *DevEx* ou *User Experience (UX)* neste tipo de editores permite encontrar técnicas interessantes a ser exploradas. Garantir que mudanças são facilmente reconhecíveis, facilitar a identificação de erros e de próximos passos a seguir e oferecer um sentido de controlo, flexibilidade e customização são exemplos de práticas e melhorias que podem aumentar a *DevEx* em editores visuais. *Infragenie*, uma ferramenta de geração automática de diagramas arquiteturais desenvolvida numa tese do Mestrado em Engenharia Informática e Computação (M.EIC), vai ser utilizada para aplicar as técnicas encontradas e testar a sua eficácia.

As perguntas de investigação são definidas de modo a tentar comprender quais os efeitos positivos do uso de um editor visual em comparação com um textual para edição de diagramas automaticamente gerados. A aceitação da ferramenta pela comunidade e a sua usabilidade também são relevantes no ato de determinar quais as melhorias fornecidas por editores visuais quando comparados com editores com base em texto.

Um estudo empírico suportado por uma experiência com um grupo de controlo e um grupo experimental permitiu reunir dados relevantes que são necessários para avaliar o protótipo de editor visual desenvolvido. Foi pedido aos participantes que completassem seis tarefas e respondessem a questões relacionadas com o *Technology Acceptance Model* (TAM), a *System Usability Scale* (SUS) e a importância de certas funcionalidade e *standards*. Os resultados obtidos são analisados para formular conclusões sobre o comportamento do editor visual quando comparado com uma alternativa baseada em texto.

Os participantes da experiência consideram que protótipo desenvolvido é uma ferramenta intuitiva, útil, fácil de usar e simples. Com as conclusões retiradas do estudo, podemos afirmar com confiança que editores visuais de modelos são alternativas válidas para editores de diagramas baseados em texto. Estes editores, por sua vez, aumentam a qualidade da documentação, permitindo que os projetos se tornem melhores e mais compreensíveis. Com projectos de *software* mais fáceis de compreender é mais fácil de partilhar conhecimentos de uma maneira mais clara, contribuindo para uma comunidade de desenvolvimento de *software* mais aberta e bem-informada.

Acknowledgments

This work could not have been completed without the help and support of a large group of extraordinary people.

First, I thank my parents and brother, who supported me throughout my academic journey, making me see the reason and still maintaining a sense of humor when I was getting lost in deadlines. Thank you for always being there for me. And to my cats, who, even without understanding what I was complaining about, were my mostly silent companions on this journey.

I would also like to thank the group of amazing friends that FEUP has given me, who always ensured that we had some fun times in the middle of the five years we have been working hard. You know who you are. Thank you for all the jokes and emotional support.

Finally, I want to thank Prof. Filipe Correia and Prof. Paulo Queiroz, as well as the whole Infragenie team and other students oriented by Prof. Filipe. Prof. Filipe, thank you for trusting me when you approached me to do this project. Prof. Paulo (Gabriel), I appreciate all your valuable opinions and suggestions throughout my thesis.

All of you helped and guided me in some way through this dissertation, and I am grateful for that. This experience was great, and I'm sure you are the people I can blame for that.

Ana Matilde Barra

"Being fearless doesn't mean you have no fears.
It means you're strong enough to face them."

Taylor Swift

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Motivation . . . 2
- 1.3 Goals and contributions . . . 3
- 1.4 Dissertation Structure . . . 4

2 Background **5**
- 2.1 Developer Experience (DevEx) . . . 5
  - 2.1.1 Feedback Loops . . . 6
  - 2.1.2 Cognitive Load . . . 6
  - 2.1.3 Flow State . . . 7
- 2.2 Living Documentation . . . 7

3 State of the Art **9**
- 3.1 Literature Review Protocol . . . 10
- 3.2 Dimensions of DevEx and Visual Model Editors . . . 11
  - 3.2.1 Methods used to influence Feedback Loops . . . 12
  - 3.2.2 Methods used to influence Cognitive Load . . . 13
  - 3.2.3 Methods used to influence Flow State . . . 16
- 3.3 Practices for designing Visual Editors . . . 17
- 3.4 Conclusions . . . 19

4 Research Strategy **24**
- 4.1 Scope . . . 24
- 4.2 Thesis Statement . . . 25
- 4.3 Methodology . . . 26

5 A Visual Editor for Infragenie **28**
- 5.1 General Prototype Goals . . . 28
- 5.2 Development Process . . . 29
- 5.3 Refactoring Infragenie . . . 30
  - 5.3.1 File system organization . . . 30
  - 5.3.2 State Management . . . 31
  - 5.3.3 Duplicate code . . . 32
  - 5.3.4 Including external libraries . . . 32
- 5.4 Evolving the previous Infragenie frontend . . . 33
- 5.5 Visual Editor Prototype . . . 34
  - 5.5.1 Features . . . 34

6 Empirical Study **42**
6.1 Goals . . . 42
6.2 Methodology . . . 43
6.2.1 Experiment Structure . . . 44
6.2.2 Sample . . . 45
6.2.3 Environment . . . 46
6.2.4 Profile Questions Form . . . 46
6.2.5 Evaluation Form . . . 47
6.3 Experiment Execution . . . 51
6.4 Variables . . . 51
6.5 Data analysis . . . 52
6.5.1 Participant Data . . . 52
6.5.2 Performance on Tasks . . . 53
6.5.3 Technology Acceptance Model (TAM) . . . 56
6.5.4 System Usability Scale (SUS) . . . 60
6.5.5 Feedback . . . 63
6.6 Discussion . . . 66
6.7 Threats to validity . . . 68
6.8 Empirical Package . . . 70

7 Conclusion **71**
7.1 Contributions . . . 73
7.2 Future Work . . . 73

A Infragenie Mockups **81**

B Profile Questions Form **86**

C Evaluation Form **91**
C.1 Control Group (PlantUML Editor) . . . 91
C.2 Experimental Group (Visual Editor) . . . 106

D Additional Evaluation Data **121**

List of Figures

1.1 Infragenie Textual Editor Page . . . 2

2.1 Three core dimensions of DevEx . . . 6

3.1 Inclusion and Exclusion process in the SLR . . . 11

4.1 Steps of the Methodology . . . 26

5.1 Frontend File System - Django vs. React . . . 31
5.2 New and Updated Pages . . . 33
(a) Bad Repo Page . . . 33
(b) Not Found (404) Page . . . 33
5.3 PlantUML Editor side-by-side view . . . 34
5.4 Infragenie Homepage . . . 36
5.5 Infragenie Visual Editor . . . 37
5.6 Editor Property Views . . . 38
(a) Element Properties View . . . 38
(b) Link Properties View . . . 38
(c) Component Element Properties View . . . 38
5.7 Graph Details - List of Elements and Links . . . 39
5.8 Adding a new link and editing it . . . 40
(a) Adding a new link . . . 40
(b) New Link - Add Source and Target . . . 40
5.9 Editor Page Submission form . . . 41

6.1 Empirical Study Methodology . . . 43
6.2 Professional experience in software development (Q1) . . . 53
6.3 Likert Answers to Profile Questions . . . 54
6.4 Individual Task duration distribution . . . 55
6.5 Total Tasks duration distribution . . . 55
6.6 Likert Answers to TAM Questions . . . 58
6.7 Likert answers to SUS Questions . . . 61
6.8 Votes for editor features . . . 64

A.1 Infragenie Logos and Identity . . . 81
A.2 Infragenie New Pages Mockups . . . 82
A.3 Infragenie Homepage Mockups . . . 83
A.4 Infragenie PlantUML Editor Mockups . . . 84
A.5 Infragenie Visual Editor Mockups . . . 85

List of Tables

3.1 Methods used to influence Feedback Loops in Visual Model Editors 13
3.2 Methods used to influence Cognitive Load in Visual Model Editors 15
3.3 Methods used to influence Flow State in Visual Model Editors 17
3.4 Extra Visual Model Editor features found in literature review 19
3.5 Dimensions of DevEx and Visual Model Editors 21
3.5 Dimensions of DevEx and Visual Model Editors (cont.) 22
3.6 Practices for designing Visual Editors 23

6.1 Statistical difference test for profile questions 53
6.2 Statistical difference test for task durations 54
6.3 Correct answers for each task 56
6.4 Statistical difference test for TAM Questions 57
6.5 Statistical lower test for TAM Questions 2 and 6 58
6.6 TAM Scores 59
6.7 Statistical difference test for SUS Questions 60
6.8 Extra statistical difference test for SUS Questions 2 and 3 61
6.9 SUS Scores 62

D.1 TAM Usefulness and Ease of use comments (Control) 122
D.1 TAM Usefulness and Ease of use comments (Experimental) 123
D.2 SUS Usability comments 124
D.3 Comments on Feature Importance and UML compliance 125
D.3 Comments on Feature Importance and UML compliance (cont. 1 126
D.3 Comments on Feature Importance and UML compliance (cont. 2) 127
D.3 Comments on Feature Importance and UML compliance (cont. 3) 128
D.4 Additional Comments and Suggestions 129

Abbreviations

FEUP	Faculty of Engineering of the University of Porto
MEIC	Master in Informatics and Computing Engineering
MSc	Master of Science
DevEx	Developer Experience
UX	User Experience
UML	Unified Modelling Language
SLR	Systematic Literature Review
CASE	Computer Aided Software Engineering
UI	User Interface
TAM	Technology Acceptance Model
SUS	System Usability Scale
MVP	Minimum Viable Product

Chapter 1

Introduction

This chapter introduces the context and motivation behind this thesis and defines the goals and contributions of the completed work. The last section presents the structure of this dissertation and can be used as a road map for this document.

1.1 Context

Software documentation is simultaneously recognized as being of key importance and challenging [1]. Throughout the years, to address these challenges and make the process of documenting easier for developers, many best practices have been identified [2, 3, 4], and tools developed [5, 6, 7, 8, 9]. Yet, complete and up-to-date documentation seems difficult to achieve in many software projects. Software architecture is particularly challenging to document, as it is surrounded by human aspects [10] and complex information.

A common task required of newcomer developers on a software engineering project is understanding the software architecture of the system in development. To be useful and effectively communicated, software architecture knowledge must be "explicitly captured and represented as software architecture documentation" [11].

Martraire [1] states that documentation should help bridge 'old' and 'new' collaborators, as it helps remember and organize larger sets of knowledge. The author also describes documentation as the process of transferring valuable knowledge to other people now and to people in the future.

As software documentation is useful for transferring knowledge between collaborators, its value depends on "its ability to convey accurate information" [12]. Software Architecture Diagrams are a useful way to convey abstract information efficiently [13] because they are one of the most important components of software architecture documentation, providing an abstract visual description of the components of a system, whether simple or complex [14].

Software Architecture Models or Diagrams can be created using multiple notations, but a common one among programmers is the Unified Modelling Language (UML) [15, 16]. The diagrams created with this language are viable for their usefulness as software visualization and are

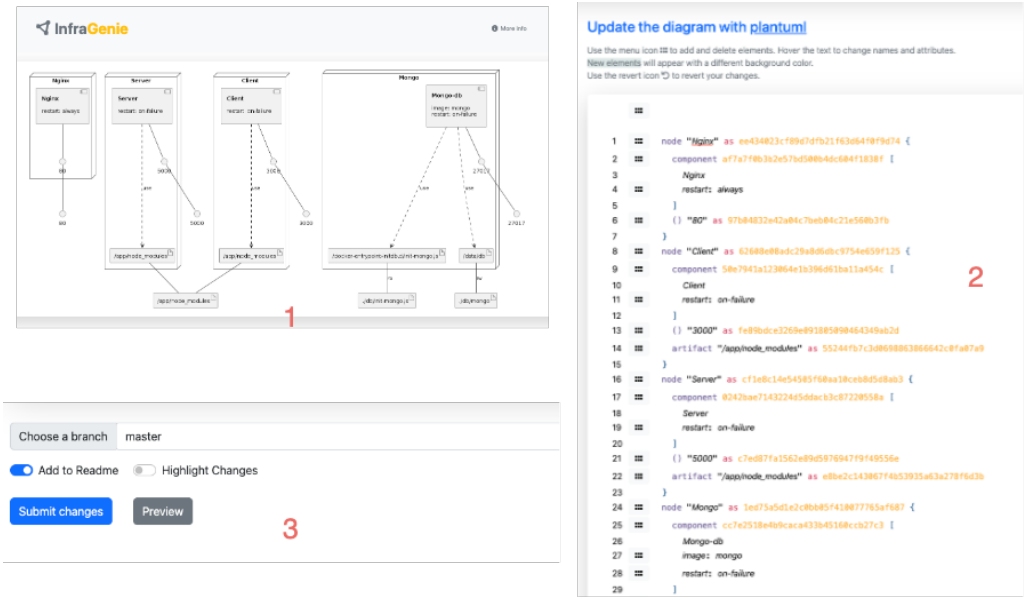


Figure 1.1: Infragenie Textual Editor Page

also found to be “a convenient mechanism for software engineers to represent high-level system designs" [17].

Martraire [1] points out that documentation takes a "lot of time to write and maintain" and can be "a fantastic source of frustration". The author also describes documentation as useful, but the writing process is tiresome for developers.

In an attempt to reduce the time consumption of the task of writing documentation, tools like Infragenie [18, 19] can be helpful. Infragenie is a tool that automatically generates software architecture diagrams from information recovered from infrastructure artifacts like docker-compose files. It allows "users to annotate the final model and modify it to describe abstractions that were not captured" through a textual editor. This editor, inserted in the tool developed by Ricardo Ferreira in his MSc Thesis in MEIC, can be seen on the right in Figure 1.1.

Textual editors have their usefulness, but visual editors could be considered a better option for modifying diagrams because many "people perceive graphic images better than textual information" [20]. Visual editors also offer a higher level of versatility not typically found in text-based editors, ensuring that the automatically generated diagrams are "powerful but also flexible" [13]. For these reasons, some works have proposed visual notations and tools as alternative to text-based counterparts [21, 22, 23, 24].

1.2 Motivation

Diagrams generated by infrastructure recovery tools, like Infragenie, may need to be corrected by human intervention to ensure they are integrated with no errors. Manual intervention "helps to identify abstractions not considered in the automatic models" [18], increasing the value of the documentation generated.

This tool currently features a text editor with PlantUML [25], a "versatile component that enables swift and straightforward diagram creation" using a simple textual language. The PlantUML text editor provides a solution that is compliant with the UML standard and allows for a usable but not very flexible way of updating and correcting diagrams. This editor allows for specific changes like adding new elements and connections and changing element properties. However, it doesn't allow changes to the position and sizing of the elements, which could make the architecture depicted by the diagram easier to understand. To ensure that these types of tools that automatically generate diagrams are increasingly accepted by developers, they must be likable and have high usability [20].

A software architecture diagramming tool may not be used if it does not provide users with effectiveness, efficiency, and satisfaction when creating, updating, or maintaining diagrams. This means the tool should perform what it is expected to do quickly and satisfactorily [26]. To increase usability and overall experience when creating and maintaining documentation and, consequently, acceptance by the community for tools that automatically generate architecture models, their editors must include certain characteristics that support effectiveness, efficiency, and user satisfaction.

One of the improvements requested by users in the evaluation survey of the Infragenie tool [18] is the better quality of the diagrams generated and greater freedom of editing. A visual editor developed with a focus on improving the Developer Experience (DevEx), which is "how developers feel about, think about, and value their work" [27], in the software architecture documentation process can be a good solution for satisfying this need.

1.3 Goals and contributions

This thesis's main goal is to improve the editor of the Infragenie tool [18] by adding a visual editor that allows for the following:

1. More flexibility in changes made to the original generated diagram;
2. Fast visual feedback on updates made on the diagram;
3. An effective, efficient, and more satisfying diagram creation process with a focus on developer experience.

The contributions of the work executed in the context of this thesis are a **state-of-the-art review**, a **visual editor prototype**, and an **empirical study** that can be replicated. The following paragraphs detail these expected contributions.

A state-of-the-art review is done to find methods and practices that enhance the DevEx and usability of diagram software architecture documentation tools that were applied to create the visual editor.

By creating a visual editor prototype for Infragenie, we expect to give users more freedom and flexibility when updating and changing the automatically generated architecture diagrams. Additional freedom and flexibility are expected to provide a better DevEx, make the tool more

usable, and possibly increase the community's acceptance of these types of tools to improve their productivity and enjoyment in documentation creation processes.

This thesis also describes an empirical study on the developed editor that consisted of executing an experiment comparing the two available editors, PlantUML and Visual. The experiment yielded relevant data that allowed us to elaborate conclusions on the positive aspects of using a visual editor in documentation creation and maintenance, as well as the usefulness, ease of use, and usability of the prototype developed compared to the textual alternative.

1.4 Dissertation Structure

This thesis is organized into seven chapters, each with a specific purpose:

- **Chapter 1** (this chapter): Introduces the work to be presented, highlighting its context, motivation, goals and contributions, and structure.
- **Chapter 2**: Provides a small background on developer experience and living documentation topics essential for understanding the other chapters.
- **Chapter 3**: Contains the state-of-the-art review of methods and practices to improve DevEx with applicability in visual model editors. It also describes the systematic literature review conducted to obtain the relevant publications to analyze.
- **Chapter 4**: Describes the scope, thesis statement and methodology. This chapter also contains the hypothesis and research questions related to the goals defined in Section 1.3.
- **Chapter 5**: Contains the description of the visual editor prototype developed on top of Infragenie.
- **Chapter 6**: Provides the goals, methodology, and data analysis of the empirical study developed to evaluate the prototype described in Chapter 5. The experiment allowed for a discussion regarding the data collected, as well as raising some possible threats to the study's validity. The details about the empirical replication package are also provided.
- **Chapter 7**: Provides an overview of the completed work and answers to the research questions defined in Chapter 4. The contributions of this thesis and possible future work are also discussed.

Chapter 2

Background

This chapter introduces some relevant concepts for understanding the next parts of this thesis, mainly Developer Experience and Living Documentation.

The contents of this chapter are heavily based on the descriptions of Noda et al.'s article "DevEx: What Actually Drives Productivity" [27] and Cyrille Martraire's book "Living Documentation: Continuous Knowledge Sharing by Design" [1].

2.1 Developer Experience (DevEx)

Developer Experience (DevEx) encompasses how developers feel about, think about, and value their work [27]. It explains how "developers experience the practice of developing software, both technically and socially" [28].

DevEx is a similar term to User Experience (UX), but instead of considering the user of a system, service, or product, it is "the experience of developers developing software in a complex social and technical context" [28] and the experience of a developer acting as a user of software development artifacts, being highly dependent on developers' feelings and perceptions while developing software. Myers et al. [29] stated that the same Human-Computer Interaction (HCI) methods applied to users can also be applied to developers to improve their experience with the tools they use.

Developer Experience consists of the lived experience of developers and the friction points they encounter in their everyday work process. By measuring and focusing on DevEx, business performance can be improved by increasing productivity, efficiency, product quality, and employee retention. The process of monitoring DevEx is highly contextual and can differ radically across teams or roles [27].

Positive influences on DevEx can be the definition of clear tasks, the use of well-organized code, and pain-free releases. On the other hand, interruptions, unrealistic deadlines, and friction in development tools, like excessive build times, can negatively impact DevEx [27].

Condori-Fernández et al. [30] refer that "usability of tools for developers is not usually taken care of since these tools are going to be used by people with a high level of knowledge in computer

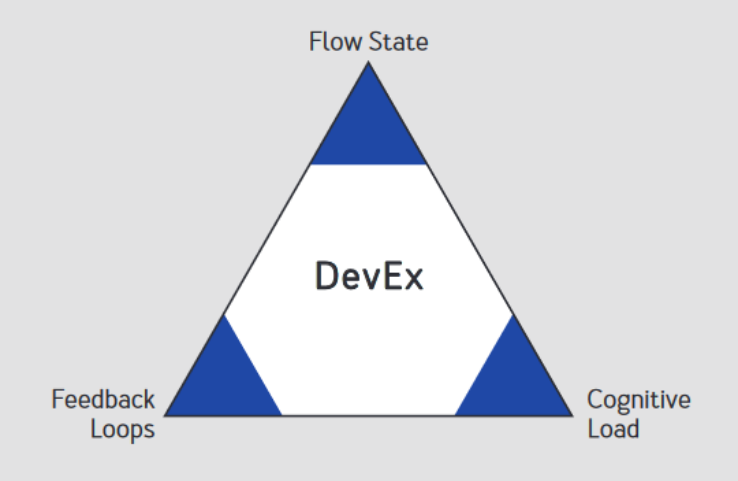


Figure 2.1: Three core dimensions of developer experience - Fig. from Noda et al.'s article [27]

science". It is essential to have DevEx in mind when developing tools for developers to ensure a good experience when developing all kinds of software.

According to Noda et al. [27], three dimensions of DevEx can be defined to more easily determine what can be measured and used to improve it. Those dimensions are **Feedback Loops**, **Cognitive Load**, and **Flow State** (Figure 2.1). Together, these three dimensions "encapsulate the full range of friction types encountered by developers". Some friction types can even belong to different dimensions, as they overlap and depend on each other.

In the following subsections, each dimension is described briefly to allow for a better understanding of this concept.

2.1.1 Feedback Loops

Feedback Loops can be characterized by the speed and quality of responses in reaction to actions performed. Faster feedback loops "allow developers to complete their work quickly with minimal friction" [27]. In contrast, slow feedback loops "interrupt the development process, leading to frustration and delays as developers wait or decide to switch tasks" [27].

Shortening feedback loops' is important to improving DevEx [27] because developers can become more productive and satisfied with their work if they know faster that they're "doing the right thing" and are "on the right track" [31].

2.1.2 Cognitive Load

The Cognitive Load dimension "encompasses the amount of mental processing required for a developer to perform a task" [27]. Some cognitive load is good because the mental effort of a task makes it challenging, but if it is too much, it becomes overwhelming for developers [31]. Because of that, increases in cognitive load are seen as unfavorable, while decreases can be seen as positive impacts on DevEx.

Cognitive load "varies according to how external information is presented and increases when mental processing is required for translating information into longer-term domain knowledge and models" [27].

Regarding documentation use, it is important to note that poor documentation increases the cognitive load because more time and mental effort are needed to understand it [27]. Therefore, improving the documentation quality may positively influence DevEx, as it can decrease developers' cognitive load.

2.1.3 Flow State

Flow State is "a mental state in which a person performing an activity is fully immersed in a feeling of energized focus, full involvement, and enjoyment" [27]. It can also be related to the feeling of "being in control", "having autonomy and stimulation", clearly grasping the goals of the project, and having uninterrupted time to develop [31].

Achieving a flow state is possible when developers' external and internal distractions are minimal, which means they can actually focus on the essential, which is developing [28].

Frequent flow state is good because it leads to "higher productivity, innovation, and employee development" [27]. Interruptions and delays in the development process, often related to the feedback loop dimension, may harm the flow state, as they can easily distract and cause a loss of focus on developers [27].

2.2 Living Documentation

As C. Martraire explains in "Living documentation: Continuous knowledge sharing by design" [1], "Documentation is the process of transferring valuable knowledge to other people now and also to people in the future" because it helps us to remember and organize large sets of knowledge.

Documentation is valuable when it accurately conveys relevant information [12], allowing the passage of knowledge between 'old' collaborators and newcomers [1, 11].

The process of writing and maintaining documentation can be "a fantastic source of frustration" because despite being useful, writing documentation is often "boring for developers" [1].

As it is not a very desirable process, developers who need to write documentation often adopt a method of "brain dumping" documentation or manually transcribing information [1]. These methods are inefficient because they take a lot of time, and the resulting documentation might be harder to understand and even update and fix [1].

Documentation might become untrustworthy and outdated if important information is missing, obsolete, or misleading. Using several sources with redundant knowledge can also lead to this problem when one of the sources changes, and the others are not updated [1].

To prevent this issue, related information must be kept near each other to ensure that changes made to related contents are made at the same time to ensure consistency and higher accessibility and usability [1, 12, 18].

Living documentation is kept reliable and accurate at any point in time, without much effort needed to maintain it, is constructed collaboratively, and is insightful and easily understandable. This 'type' of documentation tries to push stigmergy to the max, which means achieving coordination between actions so that work performed by someone can stimulate subsequent work by themselves or their colleagues [1].

Chapter 3

State of the Art

Diagrams in software architecture documentation are helpful to convey abstract information about the system developed in a project efficiently [13]. The most mainstream diagram language is the Unified Modelling Language (UML) [15], which allows users to create diagrams that are useful for software visualization, as they allow for a convenient mechanism to represent high-level and complex system designs understandably [17].

Visual Model Editors allow more freedom in visualizing and making changes to these diagrams, providing a more interactive approach and making the models easier to maintain. Modeling editors support the creation of precise and modifiable product documentation, possibly leading to improvements in the quality of the software by providing easier access to information about the product and other relevant features [32]. This type of tool must have high usability to obtain good acceptance from its users [30].

Improving the developer experience (DevEx) in visual model editors is essential to ensure that the process of creating good documentation is easier and less straining for developers and also that the documentation itself is more understandable [1, 27].

This chapter explores methodologies and patterns to achieve just that by improving usability in modeling tools and providing a better user experience (UX) to their users, who, in this case, are developers. The analysis is done to answer this literature review question:

- **Q1**: What improves the Developer Experience in a visual model editor?

  Q1.1: What improves the several dimensions of DevEx in a visual editor?

  Q1.2: What are good practices for designing visual editors?

Later sections of this chapter detail the findings obtained, reviewing the relevant articles found and some related work of importance to the scope. These results will allow for an understanding of what is currently being done in terms of usability, UX, and DevEx improvements in visual editors for modeling tools, and that is useful for achieving the thesis goals detailed in Section 1.3.

3.1 Literature Review Protocol

A systematic literature review (SLR), guided by the ACM SigSOFT Empirical Standards [33], was performed to examine the state-of-the-art of visual model editors to enhance DevEx and usability of diagram-based software architecture documentation tools. The only platform used was Google Scholar, as the results obtained using it seemed to encompass all the results we found using other search tools and repositories. The review was performed with the following study criteria:

- **Inclusion Criteria**:

  IC1: Publications on the topic of software documentation, visual modeling tools, usability evaluation, and user experience.

  IC2: Original research publications, mainly research articles, MSc and Doctoral Theses and Books.

- **Exclusion Criteria**:

  EC1: Publications with no relevant information regarding the research scope in the title or the abstract.

  EC2: Publications in any language that is not Portuguese or English.

All relevant findings were recorded and analyzed in a spreadsheet in Google Drive.

The search query was in constant evolution, adapting with each consecutive search and change made to the review question. The most relevant search terms are "user experience" (UX), "developer experience" (DevEx), "model editors" and "diagram editors". In the end, the full query used in Google Scholar, which resulting publications were analyzed to perform this review, was the following:

```
1 (intitle:UX OR intitle:"user experience" OR intitle:DevEx OR intitle:"developer
    experience" OR intitle:usability) AND ("model editor" OR "model editors" OR "
    diagram editor" OR "diagram editors")
```

This query yields 53 results, of which 52 were actual publications. After analyzing the titles and abstracts of those publications, 26 were excluded, either because of their distance from the scope of research (EC1) or the fact that they weren't written in English or Portuguese (EC2). Of the 26 left, 8 were considered most relevant and worth further analysis. On the other hand, 18 were considered adjacent to the scope and were only analyzed further if the others could not yield information to answer the literature review question in depth. This process is illustrated in Figure 3.1.

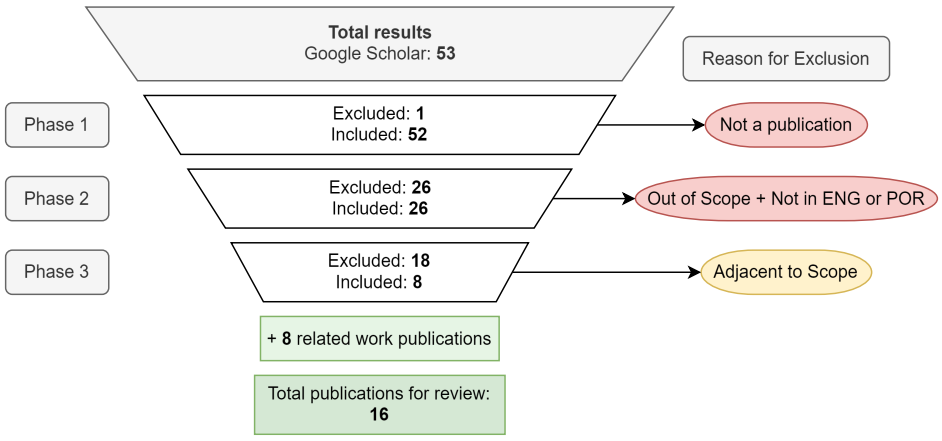


Figure 3.1: Inclusion and Exclusion process in the SLR

3.2 Dimensions of DevEx and Visual Model Editors

Documentation helps developers to remember what they did and where it is located so that they "don't end up redoing things that are already done" [1]. It can also work as a strong foundation for new and different projects because it allows for transferring valuable knowledge between people. Software architecture diagrams are a valuable way to convey abstract information successfully. Creating good and descriptive diagrams takes time, so a tool that automatically generates architectural diagrams from pre-existing information in the source code is important for helping with this task. These automatic tools may enforce a specific notation on the diagrams, and that can be helpful to ensure it is easier for others to understand.

All types of outdated documentation, including diagrams, can lead to additional cognitive load for developers because extra effort is needed to determine which parts of the documentation are up-to-date with the code and which aren't anymore. To preserve its relevance, documentation must be maintained and used; if not, it shouldn't be needed in the first place [1].

Visual modeling tools help developers create good documentation faster, which can be helpful when they are faced with maintenance tasks. If a Visual Model Editor is implemented with developer experience and high usability in mind, creating, updating, and using documentation can become a fun and low-time-consuming task, and the tool itself might be more easily accepted by the community [30].

Terekhov et al. [20] point out that "tools should help their users solve problems and not bring them new ones" and that "the more likable and usable a tool, the more user acceptance it might have". Tools should provide solutions and not setbacks so that developers feel that the tool makes their complex tasks more straightforward and faster, allowing them to execute their tasks effectively and efficiently and freeing up more time for other software development tasks that users prefer. They also refer that usability problems in a tool might be related to two dimensions: **Tool-related**, i.e., issues inside the tool itself, or **Language-related**, i.e., problems inherent in the

programming/modeling language used by the tool.

The literature review allowed the discovery of several methods that can be used to influence DevEx positively. In the sections below, these methods are presented according to the dimension in which they belong the most from the three defined by Noda et al. [27]: **Feedback Loops**, **Cognitive Load**, and **Flow State**.

The methods were recovered through reading the publications resulting from the literature review process, and their assignment to each of the dimensions was done through reflection on the findings of the papers reviewed. This assignment was made in the most objective way possible, even though it is a process that involves a degree of subjectivity. The confidence level in each assignment can be seen in Table 3.5 in this chapter's last section; the lower the confidence level, the higher the subjectivity of the assignment process.

3.2.1 Methods used to influence Feedback Loops

Short feedback loops allow developers to complete their work quickly with minimal friction [27], making it faster to observe what is being developed. Feedback loops can also be shortened by collaborating with co-workers and listening to their points of view about what is being created, allowing for more valuable work to be done in a smaller amount of time. The findings detailed below are on the scope of development in general and also on visual editor tools. They can also be observed in a more graphical view in Table 3.1.

As previously described, collaboration and co-worker feedback is one of the many ways to obtain a shorter feedback loop. These practices also nurture transparency [28] inside teams and organizations, allowing for fast problem-solving and adaptability, leading to fewer bottlenecks and errors, as multiple people are constantly checking the consistency [32]. These activities divided by several people lead to less individual strain on developers.

Feedback loops can also be nourished through regular formal verifications and testing [14], even though these are typically executed on top of the code and not the documentation itself. These tasks provide the same benefits as the ones described in the previous paragraph, as they also contribute to 'shared code-ownership', delegating important and high-responsibility activities to several people and dividing the strain and cognitive load between members of the projects.

The graphical aspect of feedback loops is also essential, as users (in this case, developers) "learn better from words and pictures than from words alone", and any interaction made with a system "should be visible and not hidden", as it is better to recognize changes rather than recalling were to find them [13, 17]. It is safe to say that changes should be easily recognizable to allow for faster feedback loops that will speed up development. However, the feedback time should also be considered, as "immediate feedback might be slowing some users down", distracting them from their current task and disrupting their flow state. In these cases, delayed feedback might be a better option [13].

Regarding the specific case of Visual Model Editors, Kalantari and Lethbridge refer in [14] that features that provide graphical and textual support, like the use of dialogs, for instance, and

Table 3.1: Methods used to influence Feedback Loops in Visual Model Editors

	References					
Method	[32]	[28]	[14]	[13]	[17]	[34]
Collaboration	X					
Co-worker Feedback	X					
Consistent checks	X					
Transparency		X				
Formal verifications			X			
Testing			X			
Changes should be easily recognizable				X	X	
Consider the time of feedback (delayed or immediate)				X		X
Provide graphical and textual support when possible			X			
Gamification functionalities			X			
Changes should be committable to avoid confusion					X	

Authors:

- Bobkowska & Reszke [32]
- Nylund [28]
- Kalantari & Lethbridge [14]
- Wybrow [13]
- Mayr [17]
- Giraldo et al. [34]

gamification of diagram-making process are beneficial. These features can highlight to the developer what they are not doing correctly 'in runtime' and provide more effective and fun feedback, motivating them to keep doing their work. Gamification might be a good approach, but it can also have a negative impact on the acceptance of the tool, as some developers may not be inclined to use a tool if it leads them to believe that they are not being taken seriously by the tool's creators or that their work is not serious enough because they are having fun.

In a survey made in the context of Mayr's research in [17], the respondents referred that changes made to the diagram or any of its properties should be committable (with the 'ENTER' key, for instance) to avoid possible confusion. In another survey, in the context of the research of Giraldo et al. [34], it is discovered that users expect that their changes made are automatically reflected in the diagram. With both these ideas in mind, it is believable that users prefer a visual editor that reflects their changes in a fast, visible, and free-of-confusion way, perhaps compiling their changes after a key press, but without much delay added to obtain the visual feedback on their changes.

3.2.2 Methods used to influence Cognitive Load

Cognitive load is the amount of mental effort required for a developer to perform a specific task [27]. The higher the cognitive load, the more mental processing is needed, and, in most

cases, more time is going to be spent performing the task. As before, the findings detailed below are on the scope of development in general and then on visual editor tools. They can also be seen in a more graphical view in Table 3.2.

Complex problems affect DevEx negatively [28]. To help developers complete tasks efficiently, it is imperative to reduce cognitive load and, therefore, development time [30]. The following paragraphs specify the methods found to influence this dimension of DevEx.

Diagram-making tools like Infragenie [18] can be considered Computer Aided Software Engineering (CASE) tools. Bobkowska and Reszke [32] consider that CASE tools "are expected to reduce the amount of work required to produce a work product and present software artifacts at a level of abstraction that allows for effective decision-making". Reducing the amount of work and supporting effective decision-making is important to reduce cognitive load in software development because of the high number of decisions developers must make every day.

Community and co-worker support throughout the development is also relevant for reducing cognitive load because the shared understanding that can be formed between people working on similar things or the help tutorials other developers might have done is highly beneficial for the decrease of mental effort needed to perform a task [28, 32, 14].

By using diagrams to visualize the software architecture, the developers are provided with insight and understanding that reduces the complexity of the existing software system under consideration, making it easier to understand [17]. That being said, it is important that the Visual Model Editor offers features to "maximize readability or emphasize important information" in the diagrams to provide this clearer insight into the project's software architecture [13].

Visual Editor tools can also reduce cognitive load by making errors easily identifiable [34] and also by eliminating complicated ways of doing tasks and features that invite mistakes [32]. Removing unnecessary features is important to make it easier to find the options the user is searching for, and, on the other hand, making errors more straightforward to identify makes it easier and faster to locate where they can be fixed.

Turning the models created or edited with the Visual Model Editor into auditable documents is a good way to allow for versioning, traceability, and review of the created models. Auditable documents allow users "to assess at any time who, how, why and what has been produced, by tracking information regarding the authoring process" [12]. This approach might be relevant to allow the comparison of different versions of a created model and, if necessary, return to them or merge them with one another [32, 14].

Tools that recover infrastructure information to generate architecture diagrams automatically, like Infragenie, can also benefit from displaying information about the origin of each piece of knowledge presented on their editor. This functionality could reduce cognitive load, allowing the developer to recognize the source of each piece of information presented in the final diagram faster and making it simpler to know which files are relevant for the final software architecture diagram result.

Features like undo and redo stacks for each diagram (or collaborator if there are multiple editing at once) and delegation of repeated actions through options in the editor are also interesting

Table 3.2: Methods used to influence Cognitive Load in Visual Model Editors

	References						
Method	[32]	[28]	[14]	[13]	[17]	[34]	[12]
CASE tools should reduce the amount of work needed to complete each task	X						
Community and Co-worker support throughout development	X	X	X				
Eliminate complicated ways of doing tasks	X						
Eliminate features that invite mistakes	X						
Turn diagrams in documentation into auditable documents							X
Related information should be kept near						X	
Compare different model versions and Model diff/merge functionalities	X		X				
Additional features (Code generation, Reverse Engineering, Exportation, Versioning + Collaborative modeling, templates, synchronization with other documentation artifacts, Exportation, Automatic modeling recommendation, AI-empowered software assistants)	X		X				
Make errors easily identifiable						X	
Descriptive icons should be used to ease the identification of options to select						X	
Maximize readability or emphasize important information				X			
Undo and redo stacks for the diagram				X	X		
Origin information about each piece of knowledge should be presented on the editor							

Authors:

- Bobkowska & Reszke [32]
- Nylund [28]
- Kalantari & Lethbridge [14]
- Wybrow [13]
- Mayr [17]
- Giraldo et al. [34]
- Correia et al. [12]

to reduce the cognitive load of the diagram creation and maintenance tasks [17, 13]. These features reduce the mental effort needed to remember the full task process, making it easier and faster for the developer to conclude them.

Descriptive icons must be used to facilitate and expedite identifying options or components to select, and related information should be kept near [34]. This means that if a certain type of diagram is being created, then the connectors that are specific to it must be more accessible than the ones that aren't. As previously stated, features that reduce the mental effort needed to complete

the task are relevant for reducing cognitive load.

Some additional features like collaborative modeling, templates, synchronization with other documentation artifacts, code generation, reverse engineering, automatic modeling recommendations, AI-empowered software assistants, and exportation can also be of interest to some users [14, 32]. These types of features are not essential but were deemed relevant by these authors and can lessen the cognitive load of developers as well.

3.2.3 Methods used to influence Flow State

Flow State is a mental state in which a person performing an activity is fully immersed in a feeling of energized focus, full involvement, and enjoyment. Maintaining this mental state frequently makes developers more productive and innovative, but it can be easily broken by interruptions and delays [27]. As before, the findings detailed below are on the general development scope and also relative to visual model editor tools. They can also be viewed in Table 3.3.

To increase the chances of a user reaching a flow state when using a tool, it should be developed in a way that allows the user to feel control of their actions and reduces the probability of the user getting stuck in a problem, as it affects flow state negatively [28, 32]. The tool should be flexible (e.g., by providing multiple viewports), customizable, and possibly offer opportunities for integration and interaction with other tools [14]. It is important to give the user a sense of control over their actions while ensuring they can be seen on screen to provide sufficient feedback. Instant consistency checking [14] and a help system [26] might be good options to provide an 'easy way out' of problems and possible errors to avoid being stuck and losing the flow state.

To minimize chances of interruptions in the flow state, annoying features must be eliminated, and actions that involve eliminating components must delete them at once throughout all the artifacts, whether textual or graphical [34, 32]. That way, there should be no need to switch between artifacts to check if everything is updated, avoiding possible conflicts and ensuring consistency. This also means that changes made to the visual editor to be developed should preferably be reflected in the existing textual editor and vice versa.

Smart solutions can also be added to expedite the tasks to be done and make it simpler to complete them while maintaining focus. Some examples are having associated objects like components and connections, shortcuts (whether digital or physical, like through keys on the keyboard), bookmarks, and hand menus [32]. Defining custom key bindings for shortcuts is also a good option for developers who like to have flexibility in performing tasks, improving flow state maintenance [13].

Regarding Visual Editor tools, Mayr [17] also defines guidelines that may enhance the flow state of users by providing easier and faster access to important features and settings. Those guidelines are repositioning the tool taskbar at the top of the screen, allowing the users to change to a property view of the diagram, enforcing the vertical stacking of elements and labels for better understanding, defining customizable palettes for UML elements and automatic cursor focus onto an element by clicking its name field on the properties view.

The positioning of the features is also important for maintaining the flow state. Wybrow [13] states that "corresponding words and pictures should be presented near rather than far from each

Table 3.3: Methods used to influence Flow State in Visual Model Editors

	References							
Method	[32]	[28]	[14]	[13]	[17]	[34]	[12]	[26]
Give the users the feeling of control in their actions	X	X						
Remove annoying features	X							
Flexibility			X					
Customization			X					
Integration and Interaction with other tools			X					
Instant consistency checking			X					
In-tool Help System								X
Ensure consistency in deleting operations						X		
Vertical stacking of elements and labels					X			
Corresponding words, pictures, and features should be presented near each other				X				
Provide smart solutions (associated objects, shortcuts, bookmarks, and hand menus)	X							
Repositioning of the taskbar to the top of the screen					X			
Property view of diagram elements					X			
Customizable palettes for UML elements					X			
Automatic cursor focus on elements (when clicking its name field on the properties view)					X			
Custom shortcuts / key bindings				X				

Authors:

- Bobkowska & Reszke [32]
- Nylund [28]
- Kalantari & Lethbridge [14]
- Wybrow [13]
- Mayr [17]
- Giraldo et al. [34]
- Correia et al. [12]
- Teruel et al. [26]

other on the page or screen". Using that principle to position the features allows for faster recognition of the next action to execute and consequently maintains high productivity levels and a strong flow state.

3.3 Practices for designing Visual Editors

As previously stated, Visual Model Editors allow more freedom in visualizing and making changes to UML diagrams [17], providing a more interactive approach and making the models easier to maintain. As "people perceive graphic images better than textual information" [20], diagrams might be better for documenting software architecture than textual descriptions, and consequently,

adding changes to them in a graphical manner instead of textual might be more appealing to developers.

UML modeling tools for creating diagrams support precise and modifiable product documentation, as well as effective team communication and software quality improvement [32]. The diagrams created must efficiently convey abstract information while having a comprehensible and intuitive structure to ensure understanding from viewers [13]. It is important that developers have a "large amount of control over the layout" of the diagrams [13] so that they can ensure the knowledge is presented in the clearest way possible to other people. The tool should also "do what is expected in a short period of time", ensuring that the creation process is as effective and efficient as possible [26].

The specific User Interface (UI) of the Visual Editor is also relevant, as interfaces might have low usability even if their functionalities are useful [29]. The UI must be convenient and preferably adaptable to convince users to use the tool frequently, meaning that the layouts and features available can be different whether the user is a beginner or an expert with the tool and the language being used [20]. The UI should also be usable in different screen sizes [17], in the sense that users with different PC screen sizes should be able to use the tool in the same or at least similar way, with the interface adapting to those environments.

The Visual Editor layout must be organized to fully support the user without hindering them, organizing functionalities in "categories of hierarchy to aid in the composition of the diagrams and navigation" [17]. When adding new functionalities, their placement should be carefully considered, and they should preferentially be placed "where the user is expecting them to be" [29] to ensure the tool's intuitiveness. Graphical elements, such as icons, should be descriptive of what they represent (i.e., the domain in which they are involved) and accompanied by a textual description to avoid potential sources of confusion [35, 26].

The list of features offered by the Visual Model Editor is also important for its usability and acceptance. The publications reviewed indicate several extra features that increase the usability of the tools, which are described below and can be seen in a more condensed way in Table 3.4.

- **Linkers:** UI elements that are displayed near the diagrams' components when users select these elements. These allow for easier and more efficient connection-making [20];
- **Embedded Control Widgets:** These elements work as 'property views' of each of the elements. They are used to show and manipulate properties' values, ensuring that relevant information appears on screen together [20];
- **Splittable links:** The links, which are the connections between elements, are repositioned when adding or deleting components, ensuring some consistency is kept after these changes. The previous existing links are kept and altered to fit in the new elements [20];
- **Element Groups:** Creating groups of elements allows for the extension of the diagrams by adding new elements and positioning between existing ones [20]. This feature also ensures that related information is kept together [17];

- **Extensions to modeling languages:** Allow to use multiple modeling languages in the same editor [17].

Table 3.4: Extra Visual Model Editor features found in literature review

Reference	Visual Editor Features
Terekhov et al. [20]	• Linkers • Embedded Control Widgets • Splitable links • Element Groups
Mayr [17]	• Extensions to modeling languages

Using open-source tools to base the Visual Editor can also be beneficial. In Orni's *HORM Diagramming Tool* [35], the open-source '*draw.io*'[1] tool was used to help develop their Visual Editor. The author states that "it is evident that using open-source software and reusing code is beneficial in terms of cost and resources and also can achieve good usability", given that it is already a thoroughly tested tool. Orni also points out that "the tool is openly accessible and can be used in other related projects, as well", recommending this approach to similar projects.

3.4 Conclusions

This State of the Art review started with the primary goal of answering a literature review question: "*What improves the Developer Experience in a visual model editor?*". To provide an answer to this question, it was necessary to first break it down into two sub-questions: Q1.1 "*What improves the several dimensions of DevEx in a visual editor?*" and Q1.2 "*What are the good practices for designing visual editors?*".

To answer Q1.1 "*What improves the several dimensions of DevEx in a visual editor?*", several publications were reviewed to find practices and improvements related to each of the three dimensions, cf. Section 3.2. The main findings from all sub-sections are described below and can be reviewed in Table 3.5, in which the methods found for each dimension of DevEx are indicated, and the confidence level for each attribution is given. In this table, there are methods that have more than one dimension attributed to them because they resolve friction points that overlap between dimensions. The attribution confidence level is related to the subjectivity of the assignment to a dimension; the lower the confidence, the more subjective the assignment of that method.

To improve and shorten the **feedback loops**, cf. Section 3.2.1, the visual editor must ensure that changes are easily recognizable and that the feedback time follows their user's needs. More

[1] Documentation for '*draw.io*' available at https://www.drawio.com/doc.

external factors like collaboration and co-worker feedback, transparency, and testing can also influence the speed of the feedback loops. Other improvements, like always providing graphical and textual support, gamification, and commitable changes, were also found to improve this DevEx dimension.

When considering **cognitive load**, cf. Section 3.2.2, removing unnecessary features, enforcing proximity of related information, transforming diagrams into auditable documents, and emphasizing important information, like making errors more identifiable or using descriptive icons, were found to reduce it. Community and co-worker support is an external factor that positively influences this dimension of DevEx, decreasing the effort needed to finish tasks. More improvements, like providing versioning, exportation, model diff/merge, and undo and redo stacks, were described as features that would also decrease the cognitive load of developers when creating diagrams in a visual editor.

Regarding the **flow state** dimension of DevEx, cf. Section 3.2.3, the visual editor must offer a feeling of control of the outcome to its users while removing annoying features, providing flexibility and customization functionalities, and ensuring consistency in deleting operations throughout the different possible views. Other improvements, like related information proximity, vertical stacking of elements and labels, smart solutions, custom shortcuts, help systems, and design choices, were also found to maintain the flow state in developers while using visual model editors.

For the sub-question Q1.2 "*What are the good practices for designing visual editors?*", the same publications were reviewed to discover practices and features to design a more usable visual editor, cf. Section 3.3. Several design practices were found to improve visual editors' usability and the tool's acceptance. The most important findings from this section are indicated in Table 3.6. A high focus on convenience, adaptability, user control, and mindful positioning of features and functionalities was found to be critical while designing and considering requirements for visual editors.

All the findings sustained the answer to the main question "*What improves the Developer Experience in a visual model editor?*" by reinforcing the importance of usability and flexibility to improve the developer experience of visual editors. Even though external factors were also found to influence DevEx significantly, the development of tools like visual model editors should only be able to improve the internal factors for each developer, given that the community and company environments related to the external ones can't be altered by changing the focus of a tool. With the development of a visual model editor considering the practices and improvements found in the literature review, enhancements on DevEx and usability are expected to increase the acceptance of the tool created.

Table 3.5: Dimensions of DevEx and Visual Model Editors

Method	Related dimensions of DevEx	Attribution confidence level
Collaboration	Feedback Loops & Cognitive Load	High
Co-worker Feedback	Feedback Loops & Cognitive Load	High
Consistent checks	Feedback Loops & Flow State	High
Transparency	Feedback Loops	Medium
Formal verifications	Feedback Loops & Cognitive Load	High
Testing	Feedback Loops & Cognitive Load	High
Changes should be easily recognizable	Feedback Loops & Cognitive Load	High
Consider the time of feedback (delayed or immediate)	Feedback Loops	High
Provide graphical and textual support when possible	Feedback Loops	High
Gamification functionalities	Feedback Loops	High
Changes should be committable to avoid confusion	Feedback Loops & Cognitive Load	Medium
CASE tools should reduce the amount of work needed to complete each task	Cognitive Load	High
Community and Co-worker support throughout development	Cognitive Load	High
Eliminate complicated ways of doing tasks	Cognitive Load	High
Eliminate features that invite mistakes	Cognitive Load & Flow State	Medium
Turn diagrams in documentation into auditable documents	Cognitive Load	Medium
Related information should be kept near	Cognitive Load & Flow State	High
Compare different model versions and Model diff/merge functionalities	Cognitive Load	High
Additional features (Code generation, Reverse Engineering, Exportation, Versioning + Collaborative modeling, templates, synchronization with other documentation artifacts, Exportation, Automatic modeling recommendation, AI-empowered software assistants)	Cognitive Load	Medium

Table 3.5: Dimensions of DevEx and Visual Model Editors (cont.)

Method	Related dimensions of DevEx	Attribution confidence level
Make errors easily identifiable	Cognitive Load & Flow State	Medium
Descriptive icons should be used to ease the identification of options to select	Cognitive Load	High
Maximize readability or emphasize important information	Cognitive Load & Flow State	Medium
Undo and redo stacks for the diagram	Cognitive Load	High
Origin information about each piece of knowledge should be presented on the editor	Cognitive Load	High
Give the users the feeling of control in their actions	Flow State	High
Remove annoying features	Flow State & Cognitive Load	High
Flexibility	Flow State	High
Customization	Flow State	High
Integration and Interaction with other tools	Flow State	High
Instant consistency checking	Flow State & Feedback Loops	Medium
In-tool Help System	Flow State & Feedback Loops	High
Ensure consistency in deleting operations	Flow State & Cognitive Load	High
Vertical stacking of elements and labels	Flow State & Cognitive Load	Medium
Corresponding words, pictures, and features should be presented near each other	Flow State & Cognitive Load	High
Provide smart solutions (associated objects, shortcuts, bookmarks, and hand menus)	Flow State	Medium
Repositioning of the taskbar to the top of the screen	Flow State	Medium
Property view of diagram elements	Flow State	Medium
Customizable palettes for UML elements	Flow State	Medium
Automatic cursor focus on elements (when clicking its name field on the properties view)	Flow State	Medium
Custom shortcuts / key bindings	Flow State	High

Table 3.6: Practices for designing Visual Editors

Reference	Design Practices
Wybrow [13]	• Offer a large amount of control to developers
Teruel et al. [26]	• Have a high focus on effectiveness and efficiency (do what is expected in a short period of time) • Graphical elements should be accompanied by a textual description to avoid confusion
Terekhov et al. [20]	• UI should be convenient and adaptable to several types of users
Mayr [17]	• UI should be usable in different screen sizes • Have categories of hierarchy to aid in diagram construction and navigation
Myers et al. [29]	• Placement of new functionalities must be where users expect them to be
Orni [35]	• Graphical elements should be descriptive of what they represent • Use of open-source tools to fasten and ease development

Chapter 4

Research Strategy

This chapter further clarifies the ideas behind the research by defining the scope and thesis statement, along with three research questions and a hypothesis that guided the work concluded in this thesis.

Making use of the practices and methodologies found in the previous chapter, the methodology for creating a visual model editor that improves developer experience of software architecture documentation tools is defined, with the final goal of applying it to improve Infragenie [18].

4.1 Scope

As previously referred, the value of software documentation depends on "its ability to convey accurate information" [12]. The documentation itself takes a "lot of time to write and maintain" and can be "a fantastic source of frustration" [1].

To reduce the time consumption of this task, Infragenie [18] automatically generates software architecture diagrams from information recovered from infrastructure artifacts, like docker-compose files, while allowing changes to be made through a textual editor. Textual editors have their usefulness, but visual editors provide more flexibility for positioning and sizing changes as well as better graphical feedback.

This thesis's main goal is to provide extra flexibility and usability when adding changes to diagrams generated by the Infragenie tool. A visual editor prototype was developed to achieve those improvements. Having the practices and improvements found in the literature review in mind and experimenting with them, the final result is expected to be a good alternative to the textual editor already available in the tool.

To achieve an end result that increases the developer experience of the Infragenie tool, it is necessary to make the process of creating software architecture documentation more enjoyable and fast, ensuring that it is easier to maintain and update the documentation of these types of projects.

4.2 Thesis Statement

Considering the information found in the literature review and the goals of this thesis, the following research hypothesis was defined:

> "A visual model editor that allows adding manual changes to automatically generated diagrams would improve the developer experience of creating and maintaining software architecture documentation."

Developers using this new visual editor would have a faster and more enjoyable experience when creating and maintaining documentation. Consequently, because of the more positive overall experience, the tool is expected to become more accepted by the software engineering community, and developers may be more open to this type of tool to agilize their documentation processes.

This research aims to contribute with an evidence-based comparison between text editors and visual editors for diagram software documentation tools by using Infragenie as a foundation. To achieve that, the following research questions (RQs) were identified:

- **RQ1 - What are the positive effects of using a visual editor instead of a textual editor for editing diagrams that are automatically generated by Infragenie?**

  The goal of this research question is to find the positive effects in each of the DevEx dimensions (Feedback Loops, Cognitive Load, Flow State) of using a visual editor instead of a textual editor for adding changes to the diagrams that were automatically generated by Infragenie. To what extent are feedback loops improved in visual editors when compared to textual editors? What decreases the cognitive load by using a visual editor? How does the visual editor ensure flow state maintenance?

- **RQ2 - Can a visual editor be more accepted by the community when compared to the previously available editor?**

  The visual editor is a graphical tool that allows changes and additions to software architecture diagrams. If developers are more willing to use Infragenie with this new editor in their work and feel more satisfied by doing it, the tool itself is more accepted. The Technology Acceptance Model (TAM) questionnaire [36] will allow the estimation of the perceived usefulness and the perceived ease of use through 12 questions. What are the scores of the Infragenie tool with and without the visual editor?

- **RQ3 - Does the developed editor achieve a good usability result (above 70%) in the System Usability Scale [37] (SUS)?**

  The System Usability Scale is a tool for measuring the usability of software. The score is calculated through a questionnaire that consists of 10 questions with five response options for respondents (from strongly disagree to strongly agree). Jeff Sauro [38] defines 10 Grades for the SUS score. Scores between 50-70 (D to C) are considered OK, above 70 to 80 (C+ to A-)is considered good, 80-84 (A) is Excellent, and from 85-100 (A+) is considered the best

imaginable. If the score is above 70, it can be concluded that the tool offers good usability, which is sufficient in this case.

4.3 Methodology

The methodology displayed in Figure 4.1 was defined to gather evidence to validate the thesis hypothesis.

This thesis starts with a state-of-the-art review with the goal of finding practices and improvements for improving DevEx and usability in visual editors for diagram-generating software architecture documentation tools. The findings allowed for the formulation of the hypothesis and research questions to guide this work further.

After that, a visual editor prototype is developed to allow us to gather evidence that sustains that hypothesis. The new editor was developed iteratively with several internal reviews and evaluations by the research team, composed of two Ph.D. researchers (Filipe Correia and Paulo Queiroz) and three master's students (Ana Matilde Barra, Duarte Figueiredo, and Henrique Nunes) who are all working on theses based on the Infragenie tool.

After the development stage, an experiment was conducted to gather relevant data to evaluate the developed visual editor prototype. The experiment was conducted with two groups: control and experimental. The experimental group tested the visual editor prototype, while the control group tested an improved version of the PlantUML editor. The experiment included an evaluation form in which the participants were asked to execute tasks in the diagram editors and respond to several questions about the tool's usefulness, ease of use, and usability. The results obtained from this form were analyzed to draw conclusions that allowed us to respond to the research questions.

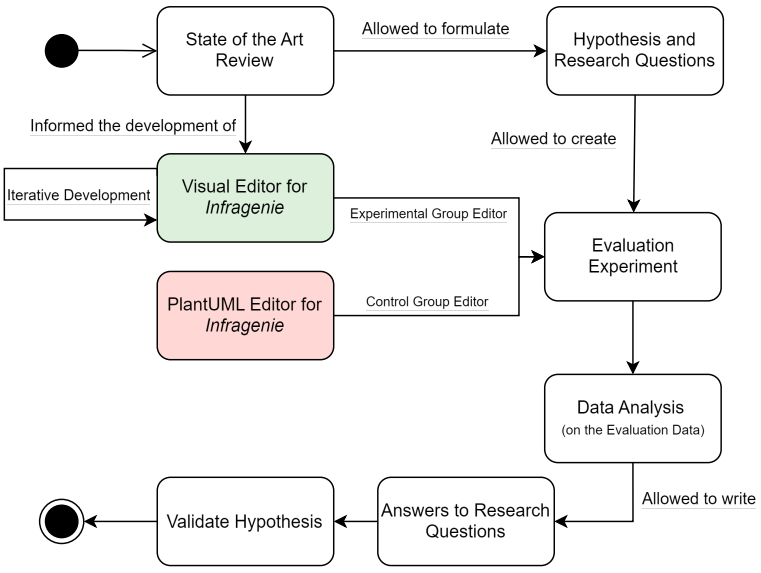


Figure 4.1: Steps of the Methodology

After the research questions were answered, there was enough data to discuss the validity of the hypothesis, providing a relevant contribution to this area of software engineering.

Chapter 5

A Visual Editor for Infragenie

This chapter describes the process and contents of the Visual Editor developed for Infragenie to attempt to improve the Developer Experience of documenting software projects, namely by easily creating diagrams that accurately represent the architecture of that software.

The approach was carefully defined to achieve the final result, and an initial refactoring stage was needed to ensure that previous work was a solid base for new developments that improved the tool's usability.

5.1 General Prototype Goals

This thesis's primary goal is to provide extra flexibility and usability to the process of creating and editing automatically generated software architecture diagrams, which we will achieve by developing a visual editor.

Generally, the visual editor should provide all the must-have features of a diagram editor, using the relevant findings detailed in Chapter 3.

The Visual Editor prototype should include the following:

- **Display of architecture diagrams.** - The user should be able to understand the displayed diagram easily.
- **Perform edits to the diagram elements.** - The user should be able to make changes to the diagram intuitively and with low effort involved. Those changes can include changing names, positions, and other relevant properties.
- **Submit and store changes.** - The changes made should be replicable and loadable to make them more appealing to software projects in collaborative environments.
- **Load previous changes.** - As said before, previous changes should be loadable, just like in the beginning when loading the initial graph. It should also be possible to make changes on top of these.

The development process of the visual editor prototype will be made on top of Infragenie. The features included in the final version should improve this tool's usability, DevEx, and acceptance by the community.

5.2 Development Process

The development process of the Visual Editor for Infragenie spanned four months, from February to May 2024. It was carefully planned and reviewed as it was ongoing to ensure that all setbacks and possible delays were handled.

The process included **four** main stages:

- **Review** - February & March

  The prototype is made on top of Infragenie, so a review of the previous codebase is needed before starting any development.

  This review was done with the input and participation of the research team referred to in Chapter 4, and it was primarily completed in a collaborative effort with Henrique Nunes. This review improved Infragenie by fixing bugs and enhancing file organization.

- **Planning & Exploratory Study** - March until May

  At this stage, the visual editor features were planned, and tools and libraries were explored to allow for visual representation and interaction with diagrams on Infragenie. Mockups for the editor were also created with the Figma [1] web app to better plan the interfaces to be developed. The mockups themselves, accompanied by detailed descriptions, are available in Appendix A. The initial plan was constantly reviewed and altered following the course of the project.

  The libraries explored were *jointJS* [2] and *maxGraph* [3]. The most promising drawing library out of the two was ***maxGraph***, as it was open-source and completely free of charge. Maxgraph is an iteration of the deprecated *mxGraph* library used by successful visual diagram editors like draw.io. However, Infragenie needed some changes before they were completely matchable.

- **Refactor & Updates** - March & April

  To ensure that the *maxGraph* library was correctly integrated, the Infragenie frontend code was **refactored** to use **React.js** and **Node.js** instead of Django. The Infragenie web application was also **updated** with CSS changes and improvements to the PlantUML Editor.

  This stage of the development process is described in detail in the next section, cf. Section 5.3.

[1] **Figma** is a collaborative interface design tool, available at https://www.figma.com.

[2] **jointJS** is a paid JavaScript Diagramming library with a free trial. Their website is available at https://www.jointjs.com.

[3] **maxGraph** is an open-source JavaScript library for display and interaction with diagrams, available at https://github.com/maxGraph/maxGraph.

- **Development** - April & May

  This stage encompassed the development of the features for the visual editor prototype itself.

  All relevant details of this phase are described in Section 5.5.

At the end of the development process, minor changes were added to the prototype to prepare for the evaluation experiment detailed in the next chapter (cf. Chapter 6).

5.3 Refactoring Infragenie

As previously referred, the previous code base presented some challenges in integrating libraries for user interaction that would be useful in later feature development. The frontend code was re-developed to solve those issues using React and Node.js.

React is a frontend development framework that focuses on user interaction and usability. Our prototype can benefit from this tool, as one of the goals is to provide a good experience when building and modifying software architecture diagrams.

Using React, we can easily integrate existing libraries, like Bootstrap, enabling us to quickly redo the features available in the tool. Besides libraries like Bootstrap, the HTTP requests needed for the correct workflow of the Infragenie web app are made with the Axios [4] library, which helps to set up requests and response handling quickly.

After refactoring the previous code, we identified some possible refinements to the features available in the previous version of Infragenie. These improvements were executed when they were considered to enhance the tool further. Section 5.4 contains a detailed description of the updates made.

In the following sections, we identify the issues found in the Infragenie Django frontend that are reduced with the refactor to React. The improvements to tackle those problems are described in detail in each corresponding section.

5.3.1 File system organization

While reviewing the code for the previous version of Infragenie, we observed a somewhat confusing folder structure. The file tree contained some unused or empty files, and sometimes, it did not provide a strong organization to the contents of each file. This made finding the specific files to add new features slower, which made the development process slower.

Refactoring the frontend allowed for better organization of the files because when working with React, developers are encouraged to use a specific file structure that makes it easier to find each needed file rapidly. A visual representation of the refactored file structure and its comparison to the previous one can be seen in Figure 5.1. Even though the resulting file system structure appears bigger in React, it is beneficial because it provides a more logical organization of the necessary files for the web app.

[4] **Axios** is a promise-based HTTP request and response handling library. Documentation is available at `https://axios-http.com/docs/intro`.

Remaking the previously available features was also made easier because of this improvement. Creating the previously available pages was faster by reusing the HTML code available in the Django frontend code in a more logical location.

The source code of a React project is typically divided into three folders: **assets**, **components**, and **pages**. The **assets** folder contains images and icons that the static pages import and display. On the other hand, the **components** folder includes all functional components created. The benefits of using components will be detailed in Section 5.3.3. At last, the **pages** folder stores the static and non-static pages of the web application. Most of the HTML code is in this folder; the components are used in these files to construct the web app.

In the specific case of the visual editor for Infragenie, three additional folders were added: **plantUML**, **utils**, and **visualEditor**. Each folder includes javascript files with functions and classes that add functionalities to the web pages created.

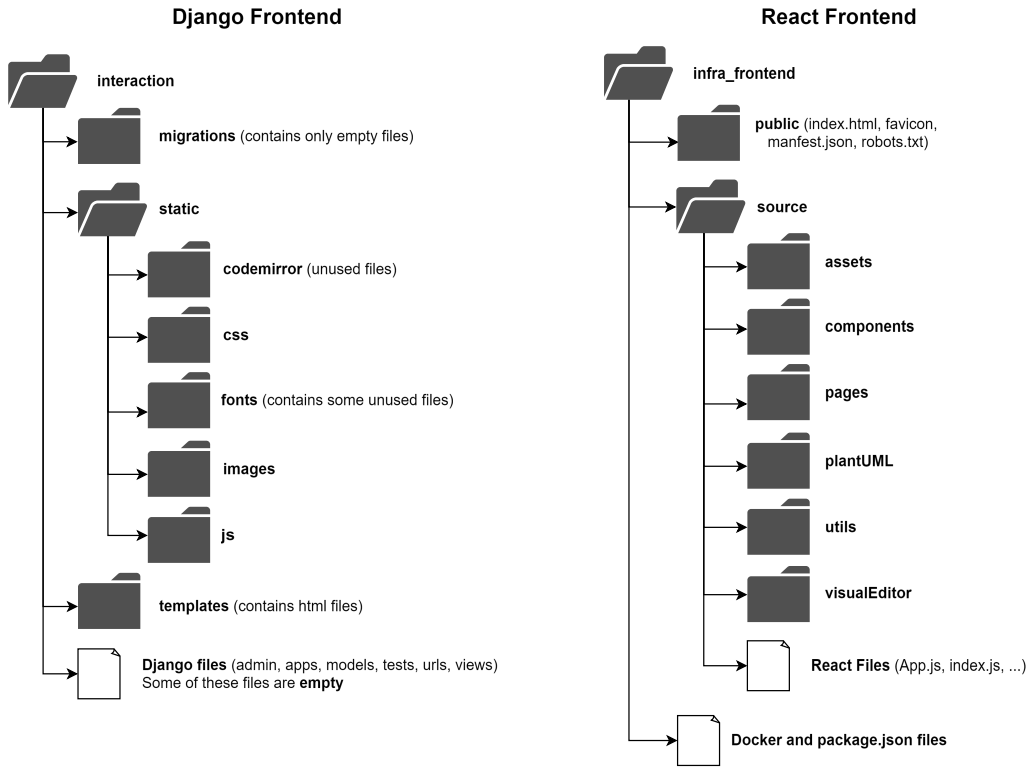


Figure 5.1: Frontend File System - Django vs. React

5.3.2 State Management

The Django frontend code relied mainly on global variables defined in JavaScript files to store the state of the web app pages. Each time one of these global variables changed, the page reload was triggered manually. This design choice made the code less modular, flexible, and scalable. This

issue needed to be addressed when developing a visual editor prototype to scale and improve the Infragenie tool.

React State Management helps to easily track changes to data displayed and changed in each page component, quickly triggering the re-rendering of that component when its state is changed.

Effectively managing state is essential in projects that constantly ask for and handle user input, like the prototype visual editor developed. State management makes it possible to quickly manage and update the frontend, keeping the feedback loops fast to ensure a better web application user experience.

5.3.3 Duplicate code

The Django frontend template HTML files developed for each web page contained **duplicate code**. This code smell was present because components like the navbar, footer, and repository form were repeated through most of the available pages.

This code smell was treated by extracting duplicate code pieces and creating React Components that contained that code. These components are modular pieces of code that can be reused in multiple places. They are written as JavaScript functions that return HTML and can include properties that store their state.

The main benefits of using components are their **reusability** and their **modularity**.

Component **reusability** helps to reduce the duplicate code problem and to decrease development time for pages with pieces of code with similar representation and functionality.

On the other hand, because each component is **independent** of others, it is updated independently when its specific state changes. This independence removed the need to re-render the whole page each time there's a change, making displaying the most recent information faster at any given point in the execution.

5.3.4 Including external libraries

In the previous frontend with Django, one needed to add the minified versions to the project to include external libraries. The *maxGraph* library wasn't available in a minified version, and all alternatives tried before to solve this issue (e.g., personally minifying the library files; adding npm to Django) were unsuccessful and, therefore, unusable.

React provides more support and makes it easier to include external libraries because new packages can be installed through the Node.js Package Manager (npm) [5].

To add new libraries to a project that uses npm, one must execute the command `npm install` in the project root folder with the wanted library as the last argument. The libraries available can be quickly found on the package registry website.

The full list of packages added to a project can be obtained by reviewing the *package.json* file or by typing the command `npm list` in the terminal opened on the project's root folder.

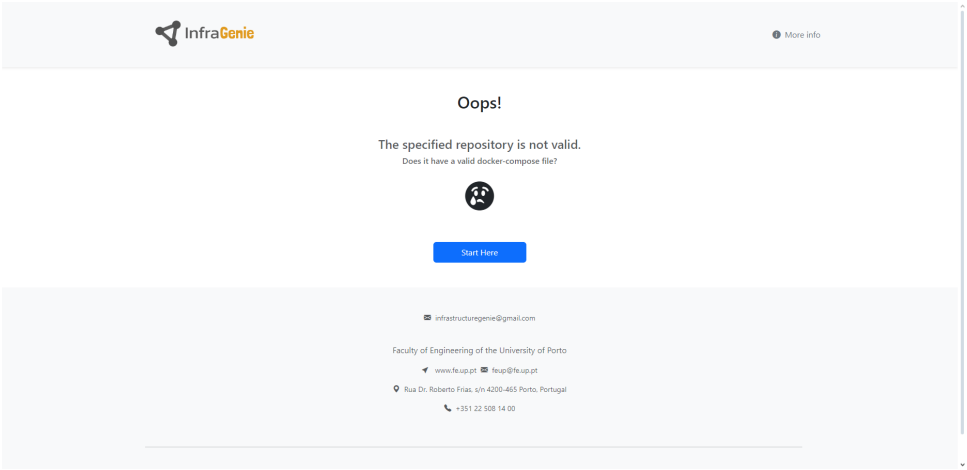
[5] **npm** is a public collection of packages of open-source code. The package registry website is available at `https://www.npmjs.com`, while the documentation is available at `https://docs.npmjs.com`.

5.4 Evolving the previous Infragenie frontend

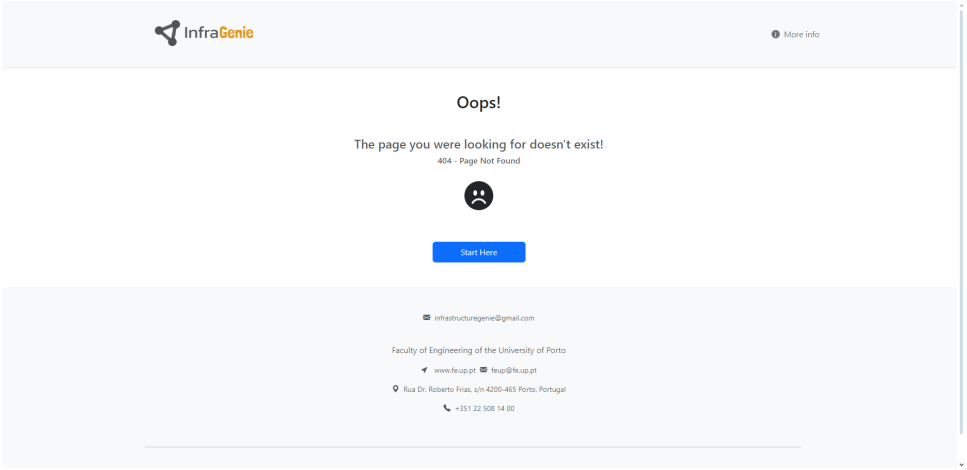
The refactoring of previously available code made improving the quality of the frontend code and user interface possible. This stage also made it possible to update the backend for the prototype by updating and creating new API endpoints to ensure consistency; these changes are thoroughly detailed in Section 5.5.

The CSS web app code was improved, mainly changing the view of the homepage, the PlantUML Editor page, the website navbar, and the repository branch selection form.

The static page 'Bad Repo' that appears in case of an error reading the selected repository and branch had its CSS improved, and a 'Not Found' page was created in case the user inserts a non-recognized URL on the browser, i.e., when a 404 error occurs. Figure 5.2 displays the interface of these pages.



(a) Bad Repo Page



(b) Not Found (404) Page

Figure 5.2: New and Updated Pages

The PlantUML editor page was improved by adding a side-by-side view that displays the PlantUML code on the left and a preview of the diagram on the right when the browser window is

wide enough. This new interface is displayed in Figure 5.3.

The behavior of the page after clicking the 'Preview' button was also changed, preventing the page from scrolling to the top each time that button is pressed. The icons available before in the editor were replaced by equivalent icons from the bootstrap-icons library.

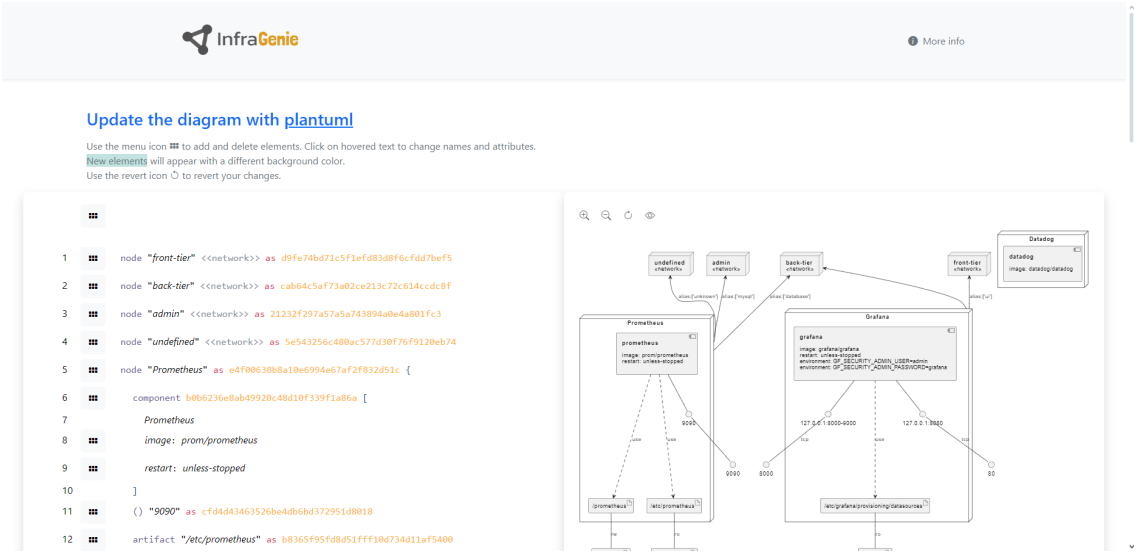


Figure 5.3: PlantUML Editor side-by-side view

5.5 Visual Editor Prototype

In this stage, all the most important features were developed. These included, among others, loading diagrams, adding, deleting, and editing elements, as well as submitting the changes to a GitHub repository.

The process of adding new features was made with the care of maintaining consistent CSS throughout the whole web app to ensure a rewarding experience.

At first, the static version of the page was created, and the backend HTTP endpoints were updated to support the visual editor operations. After that foundation was developed, more complex features were added iteratively until the evaluation experiment's Minimum Viable Product (MVP) was completed.

As previously referred, the interface mockups developed to guide prototype development are described and available in Appendix A.

The following section explains each relevant feature added to Infragenie in the context of the visual editor prototype developed (cf. Section 5.5.1).

5.5.1 Features

This section thoroughly describes the features developed for the visual editor prototype. The features are organized by their 'main purpose' in the following subsections. Each functionality is

accompanied by a screenshot of the web app that visually represents the features described.

The screenshots and descriptions in this section can be interpreted as a typical workflow guide for using the prototype.

5.5.1.1 Generating and Displaying the diagram

The Infragenie Visual Editor prototype allows users to rapidly access a diagram representation of the architecture of their software engineering GitHub public projects.

After opening the web app, users are presented with a homepage, which can be seen in Figure 5.4. Along with a general description of the tool and its purpose, this page contains a Repository Information form at the end (Figure 5.4, 1).

This form allows users to choose the repository (input field, 2) and branch (dropdown, 3) that will be the target of the tool analysis, as well as if they wish that the result is added to a README file (checkbox, 4). The two buttons below allow users to analyze their repository quickly ('Analyze' button, 4) and to access the PlantUML and Visual Editors ('Preview & Edit' button and dropdown, 5). As the analysis and PlantUML Editors were available in the previous version and the added features belong to the visual editor, the workflow described is only related to that specific usage.

To open the visual editor prototype, a user must define a repository and branch and then click the 'Preview & Edit' button (or select the visual editor option in the dropdown at the right side of that button).

After the visual editor is opened, a request is sent to the backend to load the graph JSON object on the front end. The response received is parsed to create the diagram elements and maxGraph objects that will allow for the correct display of the architecture diagram. While this response is not received, the web app displays a loading screen.

The diagram displayed on the editor page is constructed with the docker-compose file contents found and the change logs found, if any. The analysis of the change logs allows the editor to load previous changes to the diagram, such as position and sizing changes to each element.

When the loading process is completed, all the components of the visual editors are presented, as seen in Figure 5.5. Those components include a description of the visual editor (1), the editor legend (2), the visual editor window itself (3), the scrollable elements and links list (4), and the repository diagram submission form (5). The description and editor legend presents the users with information to help them comprehend each element and feature. The visual representation of the architecture diagram elements was designed to be as close as possible to UML to improve the general comprehension of the diagram.

This organization of elements on the page is expected to be intuitive and straightforward for users, which might make it faster to find some specific features than in the PlantUML Editor.

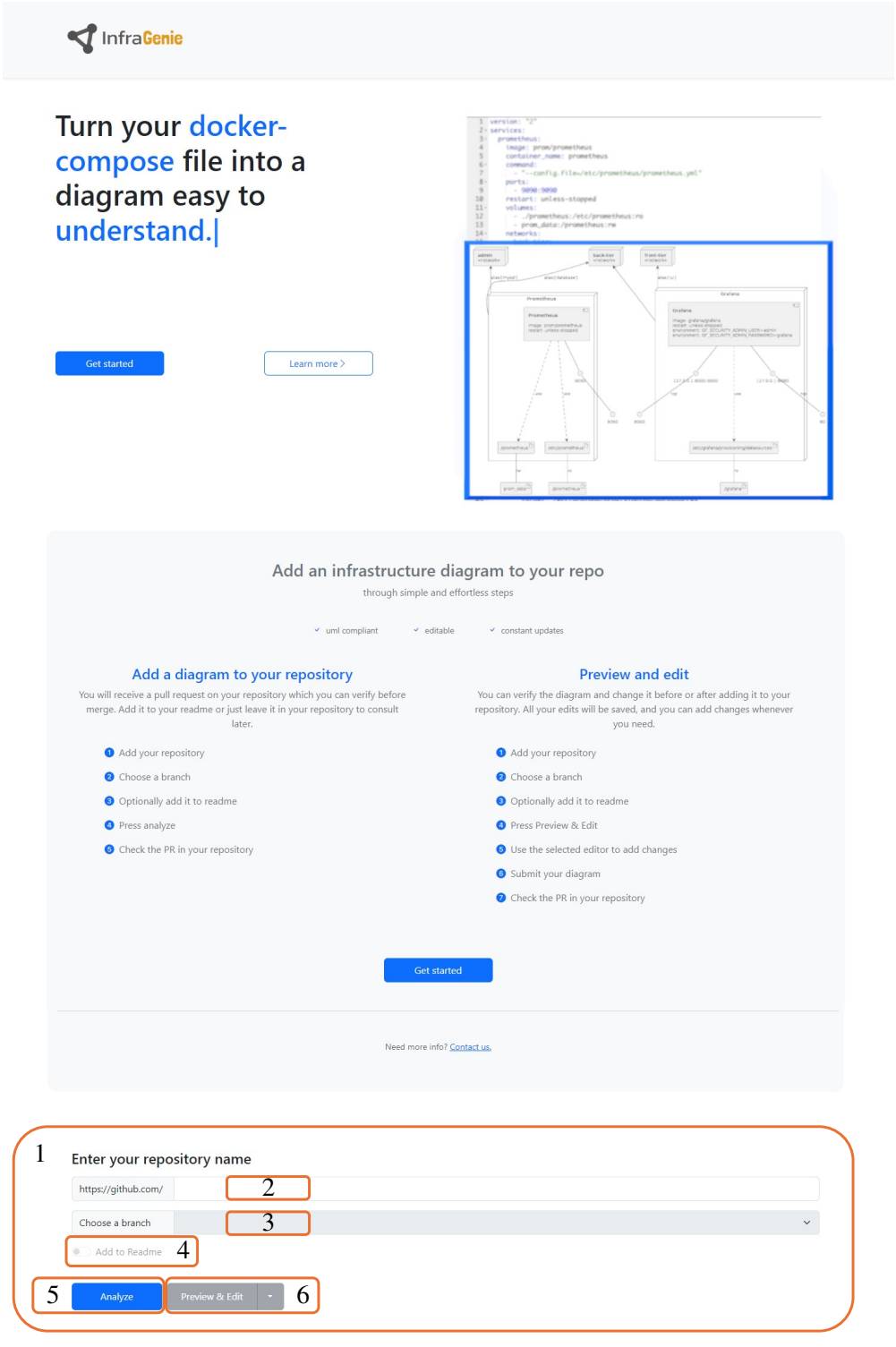


Figure 5.4: Infragenie Homepage

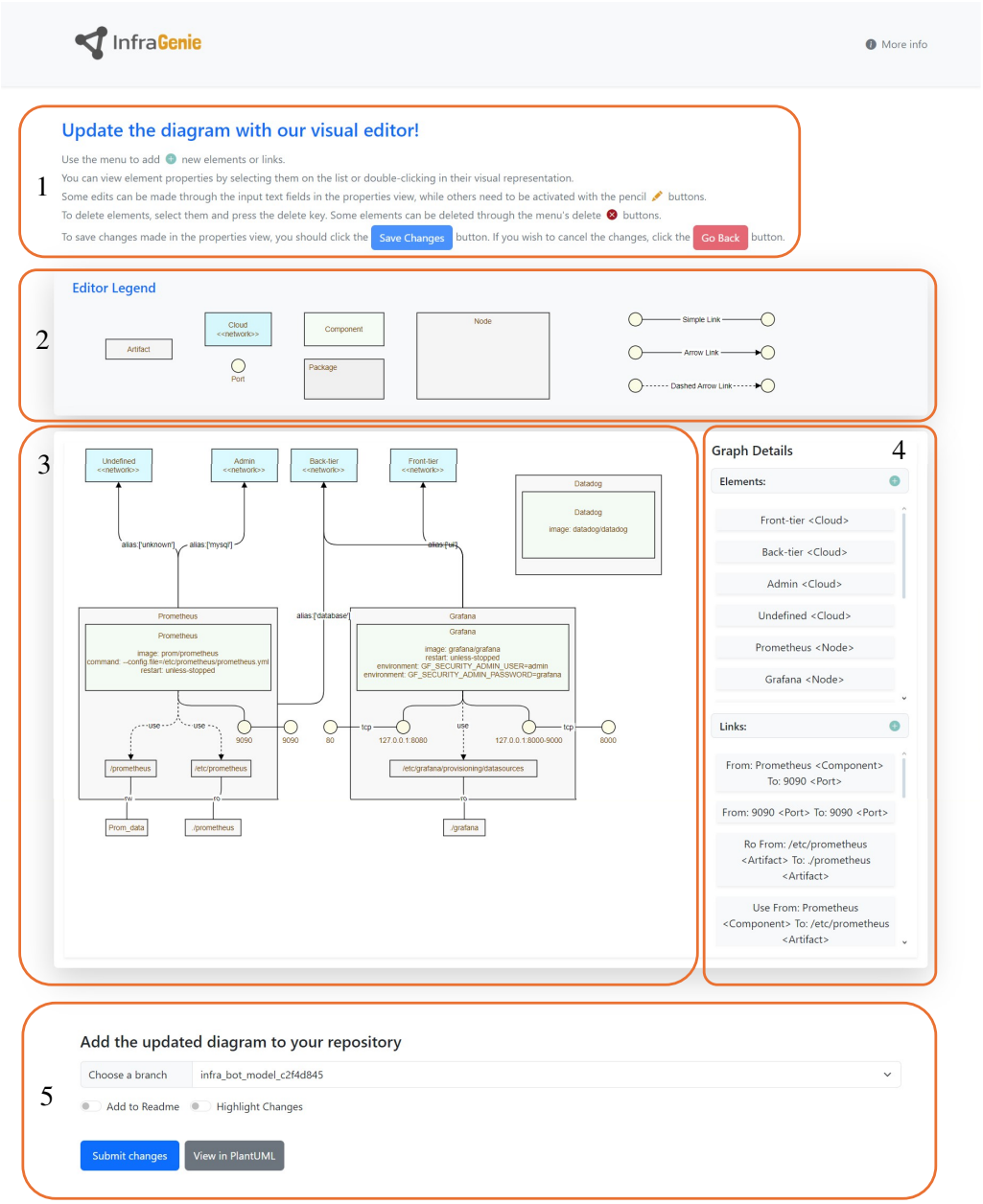


Figure 5.5: Infragenie Visual Editor

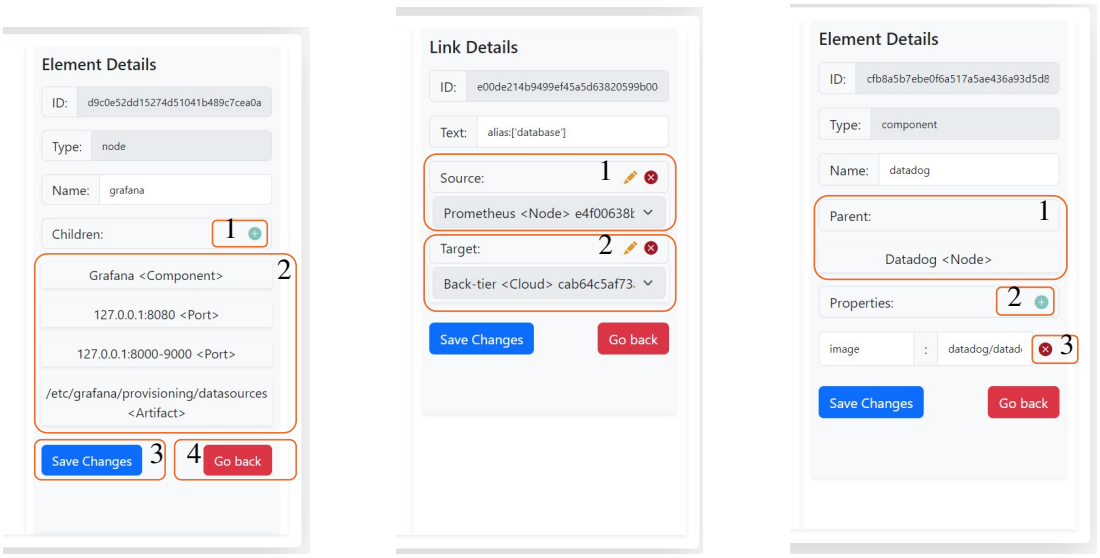
5.5.1.2 Edit diagram elements

The editor prototype allows multiple changes to each element's properties. Position and size can be changed by directly moving the elements in the view, which can be panned through by right-clicking. Each validated change made to the diagram is added to a change log, which is saved and later submitted when the user finishes the editing process.

To access the element properties that can be altered, the user can click on the corresponding element in the list of elements and links visible on the right side of the editor (Figure 5.5, 4). The

properties are also accessible by double-clicking the wanted element in the diagram view.

Once the properties view is visible, the element properties can be seen. Depending on the element type, a different set of properties may appear in this view. The ID is the only property common and visible in all element and link types. All elements have the type and name properties visible, from which only the name is editable (visible in Figure 5.6a). All links have an editable text property and a reference to a source and a target, which can be changed (Figure 5.6b, 1 and 2).



(a) Element Properties View

(b) Link Properties View

(c) Component Element Properties View

Figure 5.6: Editor Property Views

Regarding specific properties, elements with a parent defined or children connected to them have this information in this view, allowing quick access to related elements (Figure 5.6c, 1; Figure 5.6a, 2). Only elements of type node or package have the option to have children. Component elements have component properties associated with them, which are displayed, and each key and value is editable (Figure 5.6c).

Any change made in the property view must be committed by clicking on the 'save changes' button (Figure 5.6a, 3). If the user doesn't press that button or instead presses the 'go back' button (Figure 5.6a, 4), the edit executed is not saved in the change log.

The property views of elements and links allow users to view all the relevant information on those structures in the same place and at the same time, possibly making it more enjoyable and easy to understand and make changes to the diagram element properties compared to the PlantUML textual editor.

5.5.1.3 Add new elements to the diagram

The elements and links list (Figure 5.5, 4) allows new elements and links to be added to the diagram root. New children can be added to a specific parent through that element's property view (Figure 5.6a, 1).

The elements available to add through the 'add element' button (Figure 5.7, 1) are artifacts, clouds/networks, components, nodes, packages, and ports. Regarding adding elements as children of others, nodes are the only elements that cannot be added as children of other elements. The user will be prompted to define a name for the new element created when clicking the add element or add children button.

New links or connections between elements can be created by clicking the add link button in the link list visible on the right side of the editor (Figure 5.7, 2). There are three available link types: simple, arrow, and dashed arrow. Simple links are merely lines connecting elements with no specific orientation. Direction matters in arrow and dashed links, as near the target element, an arrowhead appears. The difference between these last two types of links is only whether the link line is solid or dashed, respectively.

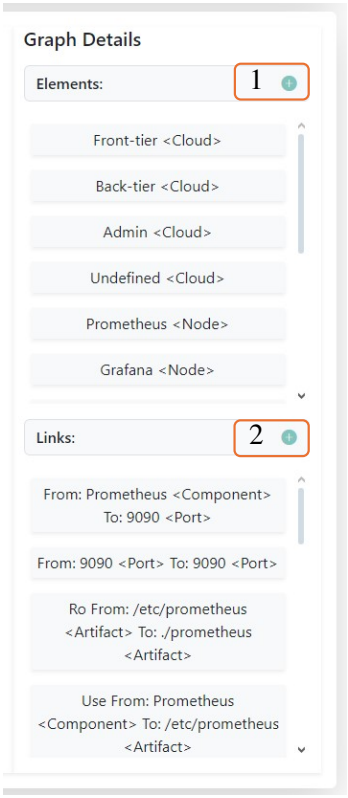
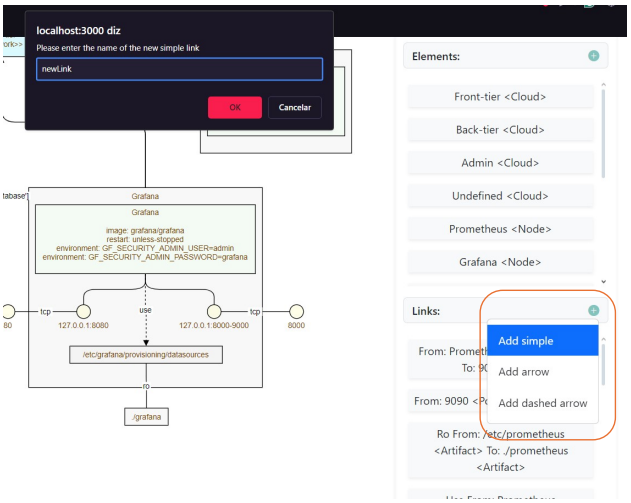


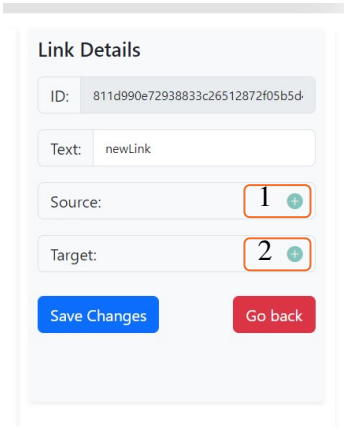
Figure 5.7: Graph Details - List of Elements and Links

The user is prompted to define a text for that link after clicking the 'add link' button (Figure 5.8a). This field can be left empty, resulting in a new link with no descriptive text defined. The created link does not appear in the diagram view right away because it doesn't have a defined source or target. To do so, one must scroll to the end of the link list and click on the new link

to view its properties. Once on this view, the user must click the 'add source' and 'add target' buttons (Figure 5.8b, 1 and 2) to select a source and a target for the link. Once the link terminals are defined, the visual representation of the connection is added to the diagram view.



(a) Adding a new link



(b) New Link - Add Source and Target

Figure 5.8: Adding a new link and editing it

New component properties are other elements that can be added to the diagram. To do so, the user must access the properties of the specific component and click the 'add property' button (Figure 5.6c, 2). Upon doing so, new key and value input fields are displayed, which need to be altered to complete this process successfully. When the user is satisfied with their modifications, the button 'save changes' must be pressed to save these edits to the change log.

These features might make adding new elements and links to the diagram easier for some users, as they provide more graphical support than the PlantUML editor features. The process of adding new links is expected to be more rewarding in this prototype, as users no longer need to copy and paste element IDs to create new connections between elements.

5.5.1.4 Delete elements from the diagram

Elements and links can be quickly deleted from the diagram by selecting them in the diagram view and clicking the 'DELETE' key on the computer keyboard.

This action might delete other elements besides the selected one if the initially deleted element is its parent. Links can also be deleted if the initially deleted element is their source or target element.

Component Properties can also be deleted from the diagram. To do so, the user must first access the specific component element property view and then click the 'delete property' button of the property they want to delete (Figure 5.6c, 3). As previously referred, all changes made in this window must be saved by clicking the 'save changes' button, or they will be lost.

Using the 'DELETE' key to delete elements from the diagram might make it faster and more intuitive to delete diagram elements. In this visual editor, the user can delete items as they are used

to instead of needing to scroll down a PlantUML textual representation of the diagram and access a specific dropdown to do so.

5.5.1.5 Submit changes to the repository

Once the user is satisfied with the result, the diagram can be submitted to the initial repository in a chosen branch from the dropdown at the end of the editor page (Figure 5.9, 1). A checkbox is available to define whether to add the new diagram to a README file (Figure 5.9, 2). After all the properties are as wished, the editing process can be finalized by clicking the 'Submit changes' button (Figure 5.9, 3). An additional 'View in PlantUML' button (Figure 5.9, 4) is available to switch between editors, which discards all changes made to the diagram.

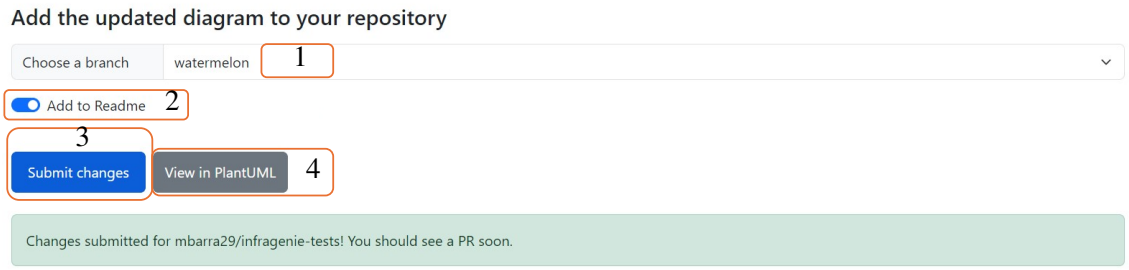


Figure 5.9: Editor Page Submission form

A simple message is displayed on the bottom of the form once the process is completed, informing us whether or not the request was successful. If it is successful, a new Pull Request should appear in the GitHub repository initially selected.

Chapter 6

Empirical Study

This chapter describes the study carried out to evaluate the previously described tool and answer the research questions defined in Chapter 4. An Experiment was conducted with a control and experimental group to obtain data that allows for comparing two different approaches for editing software architecture diagrams generated by Infragenie. This experiment was executed with twelve students of the Master in Informatics and Computing Engineering (MEIC) at the Faculty of Engineering of the University of Porto (FEUP) divided into two statistical equivalent groups.

The control group executed tasks on the PlantUML editor, with the additions made in the refactor stage, and the experimental group executed those same tasks in the visual editor prototype developed. The features available in each of the editors were carefully controlled to ensure that the two editors were as close as possible regarding functionality.

The research team defined codenames for each group to maintain secrecy before the execution of the experiment. The control group, which used the PlantUML editor, has the codename '*strawberry*', and the experimental group, using the visual editor, has the codename '*watermelon*'. The choice of fruit names reduces any connection with the original editor names, decreasing the possibility of participants knowing beforehand which editor they would be testing.

This chapter's last section describes the replication package details and instructions (cf. Section 6.8).

6.1 Goals

This study aims to collect the data necessary to reach conclusions on the usability and developer experience (DevEx) of the Visual Editor developed for Infragenie when compared to the updated PlantUML editor.

An evaluation form is given to participants in the context of the experiment, making relevant questions about the Technology Acceptance Model (TAM) and System Usability Scale (SUS) as well as prompting the participants to make comments and suggestions on the editors tested. Analyzing the data collected in this form makes it possible to answer the previously defined research questions (cf. Section 4.2) and to conclude about the hypothesis' truthfulness.

This experiment benefits significantly from being conducted with two separate groups, each testing a different editor, because it makes it easier to compare what was available before with the prototype developed. As described by Wholin et al. [39], 'Experiments are valuable tools for all software engineers who are involved in evaluating and choosing between different tools'. That direct comparison makes it possible to conclude about the changes in usability, usefulness, and acceptance of the tool with the prototype created.

6.2 Methodology

This section details the steps necessary for the successful execution of the experiment. Figure 6.1 visually represents the methodology followed for the empirical study.

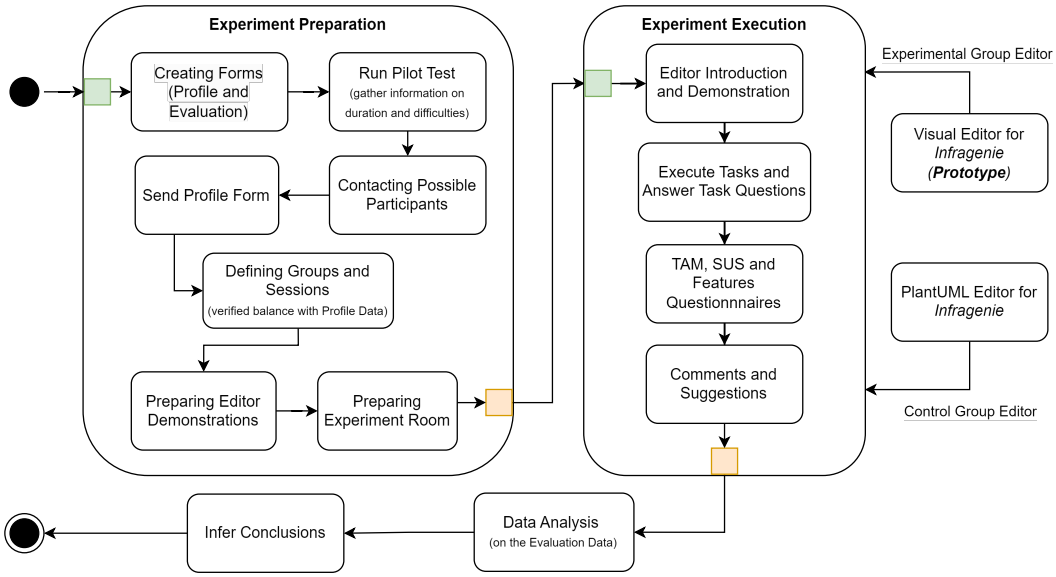


Figure 6.1: Empirical Study Methodology

To evaluate the prototype developed, we needed a hands-on approach, similar to user experience tests, to obtain the relevant information to answer the research questions defined in Section 4.2. With that in mind, the research team decided that an experiment was the best empirical study standard to evaluate the prototype.

Two Google Forms were created to share with possible participants. The first includes **Profile** Questions to collect relevant data about the participants' profiles and help verify the equivalence of the groups that were randomly defined. The Profile Form is thoroughly described in Section 6.2.4. The other is an **Evaluation** Form that is presented to the participants in the experiment and includes the tasks to complete and the evaluation questions for them to answer. This form is detailed in Section 6.2.5. The research team reviewed these forms multiple times until a final version was defined and made available to the participants.

Following the form creation, a **pilot test** on the experiment was done to determine the full duration of the experiment and to evaluate whether the texts available on the forms were self-explanatory or needed to be reformulated. The information gathered on this pilot test was important to ensure that the forms were ready to be sent and that the possible experiment participants had the most accurate information when contacted.

The research team held the experiment in person in a room at FEUP to allow for more control over possible external influences. The experiment sessions were scheduled according to the participants' and the experiment coordinator's availability.

6.2.1 Experiment Structure

The experiment was held in two separate one-hour sessions. The first session was for the control group, and the following was for the experimental group. The sessions were held independently to minimize the possible influences on each participant.

Both sessions followed a defined structure that included **four** main stages:

- **Introduction** and **demonstration** of the **visual editor** (circa **10** minutes)

  After introducing the Infragenie tool, we demonstrate the session editor. The live demo covered the main features of the editor: Opening the editor by selecting the repository and branch (each session had a separate branch to choose from), adding new elements, adding new connections or links between elements, editing properties of a specific element, deleting elements and links, and submitting the changes made to the repository.

- **Task execution** (circa **15** minutes)

  At this stage, the participants are instructed to complete six tasks as correctly and quickly as possible. In the first three tasks, participants were also asked to answer a multiple-choice question regarding their understanding of the actions executed. These tasks were designed to be simple and represent the most common and frequent actions to update architecture diagrams, e.g., adding new elements or changing properties.

- **Evaluation Questionnaire** (circa **25** minutes)

  The questionnaire presented at this stage contains 34 questions, of which 29 were mandatory. The full list of questions is available in Section 6.2.5.

  The questions can be divided into three groups. The first group includes 14 questions related to the Technology Acceptance Model (TAM); these questions allow the research team to calculate the TAM score and gather insights regarding each editor's perceived usefulness and ease of use. The second set contains 11 questions related to the System Usability Scale (SUS), which allows the team to collect information about the tool's usability. The last 4 questions collect information about the importance of specific features in this type of tool and UML compliance of the achieved diagram.

The five non-mandatory questions available in the form are all included in this last group. These questions are relevant to record additional remarks, comments, and suggestions about the tool tested in the experiment.

- **Demonstration** of the **other editor version** (circa **10** minutes)

  At the end of the experiment, a small demonstration of the editor version used by the other group was made. This demo allowed participants to obtain more general information about this thesis. At this stage, some informal comments can be obtained, which can further guide the analysis made.

6.2.2 Sample

The sample collected for this experiment is a **convenience sample** [40], given that participants were selected arbitrarily based on availability and expedience, i.e., how fast they could participate in the experiment.

To form the sample, the research team contacted 2nd-year students of the Master in Informatics and Computing Engineering (MEIC) at the Faculty of Engineering of the University of Porto (FEUP) to test the tool. This specific group was the fastest possible option since the experiment coordinator is also a second-year master's student at that institution.

Possible participants were contacted directly to collect availabilities for the week the experiment would be held (11th to 14th June 2024). The responses allowed for an estimate of the number of participants available for the experience. Most contacts were exchanged through the Discord app or WhatsApp, while a single participant was contacted in person.

In total, 18 people were contacted to participate in the experiment. As the experiment definition evolved, the research team had to remove some possible participants either because the experiment was in-person and not remote or because of their unavailability for the defined timeframe. In the end, 12 participants were available for the experiment.

A Discord server was created for all relevant experiment-related contacts, and all 12 participants were invited to join. At this stage, the participants were asked to complete a Profile questionnaire and sign their availabilities in a LettuceMeet [1] link. Section 6.2.4 shows the complete list of background questions.

The results obtained from the profile questions form were analyzed to test the statistical equivalence between the two randomly defined groups: control and experimental.

The **control group (CG)** executed the experiment in the PlantUML Editor. This group had the codename 'strawberry', and each member was assigned an ID 'SBX', where X is a number between 1 and 6.

The **experimental group (EG)** ran the experiment in the Visual Editor Prototype. This group had the codename 'watermelon'; their assigned IDs were 'WMX', where X is a number between 1 and 6.

[1] More information on **LettuceMeet** for scheduling group meetings available at https://lettucemeet.com/how-it-works.

The group formation process was made randomly by evenly separating the 12 participants. The equivalence between groups was verified by executing the Student t-test [41] and the Mann-Whitney U test [42]. The type of test run was defined by whether the question answers followed a normal distribution, which was verified with the Shapiro-Wilk test [43]. If both groups had answers that followed a normal distribution, the Student t-test was executed; if not, the Mann-Whitney U test was executed.

After completing these tests, the groups with 6 participants were accepted and attributed to the sessions. This process can be reviewed in the 'groups' Jupyter notebook in the experiment replication package.

6.2.3 Environment

An initial setup was needed to ensure that both sessions had the same knowledge and tools when executing the experiment.

First, a demonstration of each editor version was prepared, ensuring the features showcased were the same, minimizing possible differences between groups. This demonstration is held at the beginning of the experiment, as described in Section 6.2.1.

Then, on the day of the experiment, six of the FEUP room's computers were prepared with the evaluation form and the project corresponding to each session to minimize possible delays.

6.2.4 Profile Questions Form

The Profile Questions form starts by ensuring the participants that all data provided is kept anonymous and is only analyzed for the scope of the experiment.

The form contains a total of five questions. The first one tries to infer about the professional experience of the participants. The questions from two to five are related to the participants' experience with diagram-making software and languages. The questions present in the form are the following:

- **Q1** - Any professional experience in Software Development?

  Participants were asked to answer 'Yes' or 'No', and if their answer was affirmative, they were prompted to select one of the following options: Less than 1 year; 1 to 2 years; 2 to 4 years; 4 to 6 years; 6 or more years.

- **Q2** - I have experience with tools that allow the creation of diagrams from plain text or code (e.g., PlantUML, Mermaid.JS, others).

  This question tries to infer participants' knowledge of the PlantUML language.

- **Q3** - I have experience with tools that allow the creation of diagrams through a visual editor (e.g., draw.io, Lucidchart, others)

  This question is helpful to collect data on the participants' experience with similar visual editors as the prototype developed.

- **Q4** - I am familiar with the Unified Modelling Language (UML).

  The diagrams returned by Infragenie are in a notation similar to UML that aims to be ultimately UML compliant. Reviewing their answers, we can hypothesize about the participants' familiarity with this notation.

- **Q5** - I have experience creating architecture diagrams.

  The diagrams returned by Infragenie are software architecture diagrams. With this question, we verify the participants' experience with this type of diagram.

As previously discussed, these questions are important to understanding the background of the participants, making it easier to achieve a balanced distribution of the experiment groups.

The results obtained by sending this form are described in Section 6.5.1 of the Data Analysis. The PDF version of the complete profile questions form can be reviewed in Appendix B.

6.2.5 Evaluation Form

The Evaluation Form has two separate versions, one for the control group and the other for the experimental group. The contents of the form are mostly the same; the only differences are in the Introduction to the Visual Editor and the Experiment Setup sections.

This form starts by ensuring the participants that all data provided is kept anonymous and is only analyzed for the scope of the experiment to help evaluate the tool developed and find ways to improve it. The participants were also informed that the results would be published only in aggregate form and used to derive statistical data about the tool.

As this questionnaire serves as a textual support for the experiment to execute, it is divided into three main sections to reflect the first three stages of the experiment referred to in Section 6.2.1.

The **first section** of the form provides a small introduction to the visual editor in use and complements the introduction and demonstration performed by the experiment coordinator.

The **second section** asks the participants to complete six tasks correctly and quickly. This section has a different setup text for each session: the control group was instructed to open the editor on the *mbarra29/infragenie-tests* repository on branch *strawberry*, while the experimental group was instructed to open the editor in the same repository but on branch *watermelon*. This difference is needed because the selected branch was carefully prepared for the specific editor in each session.

The participants are requested to record the time (current hours and minutes) they started the set of tasks, in between each one, and at the end of all of them. This data is essential to calculate the duration of each participant when performing a specific task, which later serves as one of the variables for comparison between groups.

The correctness of the task execution is obtained from the data collected in the experiment. The first three tasks include a question with only one correct answer. The other three tasks are verified by reviewing the Pull Requests generated in the *mbarra29/infragenie-tests* repository when submitting the changes made to the diagram at the end of the experiment.

The list of tasks is the following:

- **Task 1** - Open the properties of the Prometheus Node. How many children does this node have?

  The available answers are 1, 2, 3, 4, or 5 children. The correct one is 4 children.

- **Task 2** - Open the Grafana Component. Add a new property to it. What is the total number of properties after your addition?

  The available answers are 1, 2, 3, 4, or 5 properties. The correct one is 5 properties.

- **Task 3** - Delete the Back-tier cloud/network. How many connections/links were deleted because of that initial deletion?

  The available answers are 1, 2, 3, 4, or 5 links. The correct one is 2 links.

- **Task 4** - Create a new element of a type of your choosing and add a new link connecting that new element to the Datadog Node.

  The correctness of this task is evaluated by analyzing the diagram screenshot available in the Pull Request created at the end of Task 6.

- **Task 5** - Open one element of your choosing and edit one of the editable properties (name, children, properties, ...) to your liking.

  This task is evaluated by analyzing the changelog of the changes file and also by observing the diagram SVG image.

- **Task 6** - Please submit the changes made, ensuring the whole diagram is visible in the preview window. In the dropdown, select the branch previously mentioned, check the “Add to Readme" checkbox, and click the "Submit Changes" button. After clicking the button, you will be prompted to insert your assigned ID.

  The correctness of this task is verified by checking whether the tool created the Pull Request in the repository. Tasks 4 and 5 are only verified if task 6 was completed.

The **third section** of the form contains thirty-four questions, of which twenty-nine are mandatory. The questions can be divided into three groups.

The first group includes fourteen questions related to the Technology Acceptance Model (TAM), which are essential to gather insights regarding each editor’s perceived usefulness and ease of use. The results are later analyzed to infer their relationship with the dimensions of Developer Experience.

The first six questions are related to ’Perceived Usefulness’ while the next asks participants to justify their previous answers. The following six questions refer to ’Perceived Ease-of-use’, and the last one justifies those previous answers.

All of this section’s questions, except the justifications, yield an answer on participants’ agreement with a statement on a 7-point Likert scale, with values from Strongly Disagree (1) to Strongly

Agree (7). The scale's negative and positive sides contain three steps, separated by a neutral 'Neither' answer.

The full list of **TAM-related questions**, considering the job of creating and maintaining architectural documentation, is the following:

- **1.** Using this tool in my job would enable me to accomplish tasks more quickly.
- **2.** Using this tool would improve my job performance.
- **3.** Using this tool in my job would increase my productivity.
- **4.** Using this tool would enhance the effectiveness on the job.
- **5.** Using this tool would make it easier to do my job.
- **6.** I would find this tool useful in my job.
- Please **justify** your answers regarding the **usefulness** of the tool.
- **7.** Learning to operate this tool would be easy for me.
- **8.** I would find it easy to get this tool to do what I want it to do.
- **9.** My interaction with this tool would be clear and understandable.
- **10.** I would find this tool would be clear and understandable.
- **11.** It would be easy for me to become skillful at using this tool.
- **12.** I would find this tool easy to use.
- Please **justify** your answers regarding the **ease-of-use** of the tool.

The second set contains eleven questions related to the System Usability Scale (SUS), collecting information about the tool's usability. These questions allow the research team to compute the SUS score later and reach conclusions about the connection between usability and DevEx.

The first ten of this section's questions yield an answer on participants' agreement with a statement on a 5-point Likert scale, with values from Strongly Disagree (1) to Strongly Agree (5). The scale's negative and positive sides contain two steps, separated by a neutral 'Neither' answer.

These questions also follow a specific pattern of positive questions followed by negative ones. That means odd questions are positive, while even questions are negative. The alternating positive and negative items in this scale ensure that participants have 'to read each statement and make an effort to think whether they agree or disagree with it' [37].

The last question of this group prompts the users to justify their answers to the previous ones. The full list of **SUS-related questions** is the following:

- **1.** I think that I would like to use this system frequently.

- **2.** I found the system unnecessarily complex.
- **3.** I thought the system was easy to use.
- **4.** I think that I would need the support of a technical person to be able to use this system.
- **5.** I found the various functions in this system were well integrated.
- **6.** I thought there was too much inconsistency in this system.
- **7.** I would imagine that most people would learn to use this system very quickly.
- **8.** I found the system very cumbersome to use.
- **9.** I felt very confident using the system.
- **10.** I needed to learn a lot of things before I could get going with this system.
- Please **justify** your answers regarding the **usability** of the tool.

The last nine questions prompt participants to reflect on the importance of specific features in this type of tool and UML compliance of the achieved diagram. The participants are also encouraged to leave additional remarks, comments, and suggestions about the tool tested in non-mandatory questions. Those questions are the following:

- Do you consider having a feature "undo/redo actions" important for this type of tool?

  Why? (not mandatory)
- Do you consider having a feature "move elements" important for this type of tool?

  Why? (not mandatory)
- Do you consider having a feature "use of keyboard keys" important for this type of tool?

  Why? (not mandatory)
- Do you believe the diagram achieved is compliant with UML standards?

  Why? (not mandatory)
- Do you have any comments or suggestions, or would like to suggest any new features? (not mandatory)

Section 6.5 contains a thorough analysis of the results obtained with this form. Once again, the PDF version of the complete evaluation form can be reviewed in Appendix C.

6.3 Experiment Execution

The experiment took place on the afternoon of the 14th of June 2024 in a room at FEUP. The control group session was held at 3 pm, while the experimental group was at 4:30 pm.

Each experiment session was approximately one hour, although some participants finished all tasks and questions several minutes before that time frame ended. Upon arriving at the room, the participants were asked to distribute themselves on the computers that were on, randomly assigning themselves an ID for that specific session.

Once all participants had arrived, the experiment began. First, the experiment coordinator thanked the participants for their availability and contribution. Then, the introduction to the Infragenie tool and a demonstration of the session editor were made.

Once the demonstration ended, the participants were asked if they had any questions and informed that their assigned ID was on their desks. The experiment coordinator also told them that questions about interpreting the form were allowed during the experiment. At this point, the participants started the experiment by following the instructions on the evaluation form.

The control group members were instructed to open the editor on the *mbarra29/infragenie-tests* repository on branch *strawberry*. The experimental group members were instructed to open the editor in the same repository but on branch *watermelon*.

All participants executed the tasks and answered the questions without any significant issues. Some participants had questions on the interpretation of the form, which were promptly answered with the care not to reveal any information that might influence their answers.

Once the experiment was complete, the data from the evaluation form and the execution of the tasks were collected and analyzed in a Jupyter notebook. The results are detailed in Section 6.5 and discussed in Section 6.6.

6.4 Variables

The evaluation form presented to participants in the experiment collects data that allows us to answer the research questions defined in Section 4.2.

The task execution correctness and comments gathered are relevant to answer **RQ1**: 'What are the positive effects of using a visual editor instead of a textual editor for editing diagrams that are automatically generated by Infragenie?'. Those variables that are dependent on each participant allow us to connect the participant's experience with the editor with the several dimensions of developer experience (DevEx), inferring possible conclusions on visual editors' influence in that area. The task execution times are independent variables that are also relevant to answer **RQ1**.

The Technology Acceptance Model (TAM) questions answers are used to infer conclusions for **RQ2**: 'Can a visual editor be more accepted by the community when compared to the previously available editor?'. These dependent variables can be used to compare the acceptance of the PlantUML editor and the prototype, making it possible to attempt to predict the community acceptance of the tool.

The System Usability Scale (SUS) questions answers are used to formulate an answer to **RQ3**: 'Does the developed editor achieve a good usability result (above 70%) in the System Usability Scale (SUS)?'. The values collected depend on each participant's opinion and allow us to compute the final SUS score of both editors, making it possible to conclude on the prototype's and PlantUML editor's usability and compare them.

6.5 Data analysis

This empirical study data analysis was executed at two different points in time using the data collected from both distributed forms.

First, the research team analyzed the participant data collected through the Profile Questions form to reflect on the equivalent of the randomly generated groups. The results of this analysis can be reviewed in Section 6.5.1, which verifies that the groups aren't statistically different.

Secondly, the data collected from both versions of the evaluation forms, the control and the experimental, are analyzed to reach conclusions on the prototype developed. The dimensions of this analysis span from the task execution performance (cf. Section 6.5.2) to the Technology Acceptance Model (cf. Section 6.5.3) and the System Usability Scale (cf. Section 6.5.4). Section 6.5.5 contains a review of the additional comments, suggestions, and feedback provided by the participants.

6.5.1 Participant Data

The profile questionnaire allowed the collection of relevant data about each group's participants. This section contains tables and graphs that help to understand the experiment samples. These results also allowed the research team to verify that the two groups were not statistically different, ensuring that the evaluation results did not depend on the group constitution.

The results regarding the statistical equivalence of the answers to each question are presented in Table 6.1. The table also displays the mean ($\bar{x}$) and standard deviation (σ) for the answers to each question divided by group.

For this analysis, we consider that the null hypothesis (H_0) is that the data is statistically different, and the level of significance (α) is equal to 0.05. The randomly generated groups can be considered equivalent if H_0 is rejected, which means that the p_value result of the test executed is higher than 0.05. The results allow us to conclude that the groups are equivalent because the answers to each question are not statistically different between groups.

The results of the professional experience in software engineering question, **Q1**, are available in Figure 6.2. Most participants have at least some professional experience, which can be verified because only two answers reveal that they didn't have any. In the histogram, the red bars represent answers from the control group, while the blue bars represent answers from the experimental group.

Table 6.1: Statistical difference test for profile questions

	Control		Experimental		Student t-test		Mann-Whitney U	
Question	$\bar{x}$	σ	$\bar{x}$	σ	p_value	Accept H_0?	p_value	Accept H_0?
Q1	1.83	1.57	2.33	1.37	-	-	0.51	No
Q2	1.83	0.69	2.33	1.25	0.45	No	-	-
Q3	4.50	0.50	4.83	0.37	-	-	0.28	No
Q4	4.50	0.50	4.67	0.47	-	-	0.64	No
Q5	3.67	0.94	4.17	0.69	0.36	No	-	-

Q1. Any professional experience in Software Development? (None: *0*; Less than 1 year: *1*; 1 to 2 years: *2*; 2 to 4 years: *3*; 4 to 6 years: *4*; 6 or more years: *5*)

Q2. I have experience with tools that allow the creation of diagrams from plain text or code (e.g., PlantUML, Mermaid.JS, others).

Q3. I have experience with tools that allow the creation of diagrams through a visual editor (e.g., draw.io, Lucidchart, others)

Q4. I am familiar with the Unified Modelling Language (UML).

Q5. I have experience creating architecture diagrams.

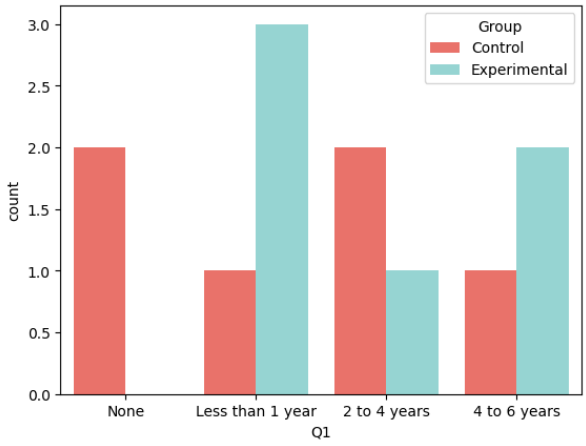


Figure 6.2: Professional experience in software development (Q1)

The following questions, from **Q2** to **Q5**, had their answers on the Likert Scale, with values from 1 to 5. The charts in Figure 6.3 contain the results obtained for the control and experimental groups, evidencing the equivalence between groups once more.

6.5.2 Performance on Tasks

The performance of participants executing the six tasks of the evaluation form is analyzed through the timestamps collected and the correctness of their answers. The tasks can be reviewed in Section 6.2.5.

First, the time duration of each task by each participant is calculated with the timestamps collected. The duration of Task 1 is equal to the timestamp between Tasks 1 and 2 minus the initial

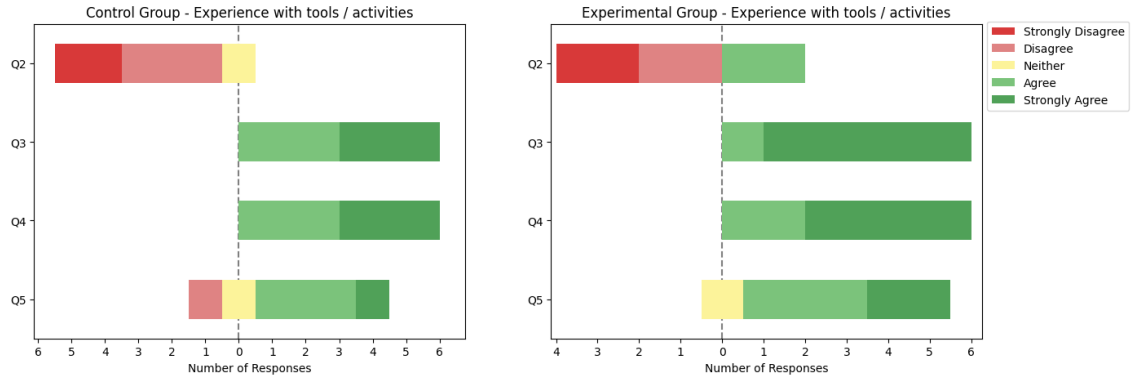


Figure 6.3: Likert Answers to Profile Questions

timestamp; the duration of Task 2 is the difference between the timestamp collected between Tasks 2 and 3 and the one between Tasks 1 and 2, and so on.

The means ($\bar{x}$) and standard deviation (σ) for these task durations were also calculated. Those values were used to conclude the statistical difference between the experimental and control groups, using a level of significance (α) equal to 0.05 (cf. Table 6.2).

Table 6.2: Statistical difference test for task durations

	Control		Experimental		Student t-test			Mann-Whitney U		
Task	$\bar{x}$	σ	$\bar{x}$	σ	Test	p_value	Accept H_0?	Test	p_value	Accept H_0?
Task 1	1.50	0.55	1.00	0.63	-	-	-	<	0.10	No
Task 2	1.50	0.55	1.33	0.52	-	-	-	<	0.32	No
Task 3	1.33	1.03	1.00	0.00	<	0.22	No	-	-	-
Task 4	1.83	1.17	2.83	1.47	<	0.89	No	-	-	-
Task 5	1.17	0.41	0.67	0.52	-	-	-	<	0.06	No
Task 6	1.33	0.82	1.17	0.41	-	-	-	<	0.50	No
Total	8.67	2.34	8.00	2.19	<	0.31	No	-	-	-

The tests executed for each task were 'lower' tests, with the Null Hypothesis (H_0) of the durations in the experimental groups being significantly lower than the durations of the control group. This H_0 was defined as we want to verify if, by using the prototype, the tasks can be completed faster than with the previous editor. The test executed for each task duration value set was determined by analyzing whether the samples from both groups followed a normal distribution. If both were normal, the Student t-test was executed; if not, the Mann-Whitney U test was run.

No H_0 for the task duration values was accepted, meaning that no task had significantly lower values in the experimental group than in the control group. Although there is no significant difference in the task duration of the two groups, we can observe a small difference in the values for $\bar{x}$.

The experimental group performed all tasks slightly faster except Task 4, where the control group was one minute faster on average. That difference might be explained because adding new links to the visual editor prototype requires more steps, i.e., finding the link at the bottom of the list

of links and defining both a source and target with a dropdown containing all selectable elements. Because those extra steps were not optimized, the experimental group took longer to complete that task.

The individual values for task duration can be reviewed visually in the box plot in Figure 6.4, which further evidences the conclusion drawn in the last paragraph. The values for the total duration can be seen in Figure 6.5. This boxplot visually presents the slight difference between groups, showing that the experimental group was generally faster than the control group. In both figures, the control group is represented in red, while the experimental group data is drawn in blue.

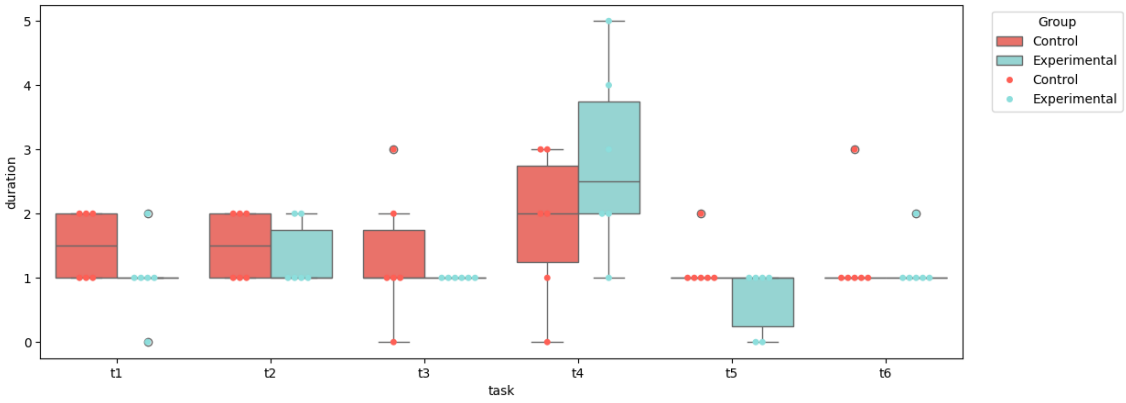


Figure 6.4: Individual Task duration distribution

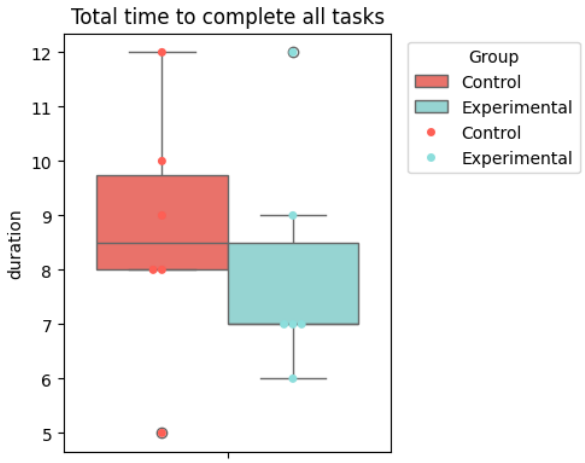


Figure 6.5: Total Tasks duration distribution

The correctness of the tasks can also be analyzed to conclude the differences between groups. The answers to Tasks 1 to 3 were considered correct if the answer selected in the form was the right one. The correctness of Tasks 4 to 6 is evaluated by analyzing the pull request (PR) created in the *mbarra29/infragenie-tests* repository. Tasks 4 and 5 are evaluated by reviewing the diagram screenshot and the changelog file present on the PR. Task 6 is correct as long as the PR is created. Because of that, Tasks 4 and 5 are only verified if Task 6 was completed.

The control group participant answers were 88.9% correct, while the answers of the experimental group were 100% correct. The distribution of correct answers out of the 6 participants can be viewed in Table 6.3. The only wrong answers were verified in Task 1, where the four control group participants answered incorrectly. These wrong answers could have happened because the control group editor doesn't have specific information about the number of elements easily inferred in the prototype by counting the children in the parent element properties view.

Table 6.3: Correct answers for each task

	Control	Experimental
Task	Correct	Correct
Task 1	2	6
Task 2	6	6
Task 3	6	6
Task 4	6	6
Task 5	6	6
Task 6	6	6

6.5.3 Technology Acceptance Model (TAM)

The fourteen questions in the evaluation form related to TAM yield results that allow us to form conclusions on the possible acceptance of the prototype compared to the PlantUML editor. The TAM questions defined in [36] are complemented by two additional questions allowing participants to comment on their previous choices regarding perceived usefulness and ease of use.

As previously referred in Section 6.2.5, the TAM questions expect a response representing the level of agreement on a 7-point Likert Scale. These questions allow us to obtain information on the editors' perceived usefulness and ease of use.

The means ($\bar{x}$) and standard deviation (σ) for each of these were calculated. These values start to indicate a slight difference between both groups.

The Student t-test and Mann-Whitney U tests were executed following the same process as the previous data samples collected, allowing us to conclude the statistical difference between the groups. As in the other analyses, the level of significance (α) used is 0.05.

The tests executed for each question were 'greater' tests because our Null Hypothesis (H_0) was that the TAM values of the experimental group would be significantly higher than the control group. This H_0 is connected with our second research question (RQ2) defined in Section 4.2. RQ2 questioned if a visual editor (our prototype) could be more accepted by the community than previously available editors. If the visual editor reached a higher TAM score than the PlantUML editor, we could safely conclude that it would be more accepted. The results of this analysis are visible in Table 6.4.

No H_0 for the TAM responses was accepted, meaning no questions had significantly greater values in the experimental group than in the control group. Although there is no significant difference in the answers of both groups, we can find meaning in the values for $\bar{x}$.

Table 6.4: Statistical difference test for TAM Questions

	Control		Experimental		Student t-test			Mann-Whitney U		
Question	$\bar{x}$	σ	$\bar{x}$	σ	Test	p_value	Accept H_0?	Test	p_value	Accept H_0?
1	6.00	0.89	5.33	1.86	>	0.78	No	-	-	-
2	6.17	1.17	4.67	2.07	-	-	-	>	0.91	No
3	6.00	1.10	5.50	1.87	>	0.71	No	-	-	-
4	5.50	1.22	5.17	1.94	>	0.64	No	-	-	-
5	6.33	0.52	5.50	1.87	-	-	-	>	0.78	No
6	6.83	0.41	5.33	1.86	-	-	-	>	0.98	No
7	6.50	0.84	6.67	0.52	-	-	-	>	0.46	No
8	6.33	0.82	6.33	0.52	-	-	-	>	0.61	No
9	6.50	0.84	6.67	0.52	-	-	-	>	0.46	No
10	6.67	0.52	6.00	1.26	-	-	-	>	0.84	No
11	6.67	0.52	6.50	0.84	-	-	-	>	0.61	No
12	6.50	0.55	6.17	1.60	-	-	-	>	0.43	No

- **1.** Using this tool in my job would enable me to accomplish tasks more quickly.
- **2.** Using this tool would improve my job performance.
- **3.** Using this tool in my job would increase my productivity.
- **4.** Using this tool would enhance the effectiveness on the job.
- **5.** Using this tool would make it easier to do my job.
- **6.** I would find this tool useful in my job.
- **7.** Learning to operate this tool would be easy for me.
- **8.** I would find it easy to get this tool to do what I want it to do.
- **9.** My interaction with this tool would be clear and understandable.
- **10.** I would find this tool would be clear and understandable.
- **11.** It would be easy for me to become skillful at using this tool.
- **12.** I would find this tool easy to use.

The means of the answers to the questions about the perceived usability (tam_pu questions) tell us that the control group had slightly higher values on the Likert scale than the experimental group. The means of the answers in the perceived ease of use questions (tam_peu questions) are more similar between groups, given that questions 7 and 9 have higher values in the experimental group, while questions 10 to 12 have lower values in the experimental group. Question 8 has the same mean for both groups.

Additional 'lower' tests, with the same parameters, were performed on the questions with a result p_value higher than 0.90. These tests allowed us to verify whether the visual editor had a significantly lower answer on the TAM questions with a high p_value. In this case, the H_0 was if the experimental answers were significantly lower than the control answers. Those additional tests were executed for questions 2 and 6, as their p_values were 0.91 and 0.98, respectively, and can be seen in Table 6.5.

These additional tests allowed us to conclude that H_0 was verified for Question 6. The answers to that question were significantly lower in the experimental group than in the control group. This

Table 6.5: Statistical lower test for TAM Questions 2 and 6

	Control		Experimental		Mann-Whitney U		
Question	$\bar{x}$	σ	$\bar{x}$	σ	Test	p_value	Accept H_0?
2	6.17	1.17	4.67	2.07	<	0.12	No
6	6.83	0.41	5.33	1.86	<	0.04	Yes

Please refer to Table 6.4 to review the TAM Questions.

might happen because one experimental participant said they 'don't use docker nor diagrams in my job' and answered 2 (Quite Disagree) to all tam_pu questions. Although the test yielded this result, we believed that if additional participants had been used, the results between the experimental and control groups would have been more similar.

The Likert charts present in Figure 6.6 visually verify the conclusions drawn before. With this visualization, we can understand that the prototype had more negative answers in the tam_pu questions, while the tam_peu questions have a more even distributed agreeance between groups.

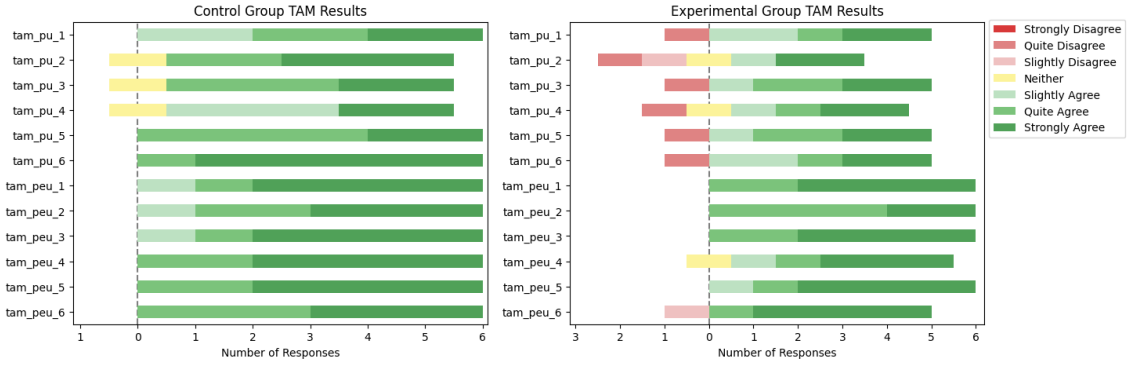


Figure 6.6: Likert Answers to TAM Questions

For furthering the analysis, the TAM scores given by each participant were also calculated using the following formulas [36] :

$$PU = (AVERAGE(TAM01, TAM02, TAM03, TAM04, TAM05, TAM06) - 1) \times (100/6)$$

$$PEU = (AVERAGE(TAM07, TAM08, TAM09, TAM10, TAM11, TAM12) - 1) \times (100/6)$$

$$TAM = AVERAGE(PU, PEU)$$

Following those formulas, first, we calculate the tam_pu and the tam_peu score for each participant (P). After that, we use those values to compute the individual TAM scores, presented in Table 6.6.

To conclude on the final TAM score of the tool, we calculated the mean of all individual scores calculated. The results show that the control group editor seems to have more acceptance than the experimental group editor, even if that difference is very small.

Table 6.6: TAM Scores

	Control	Experimental
	Score	Score
P1	79.17	97.22
P2	97.22	100.0
P3	81.94	52.78
P4	91.67	87.50
P5	91.67	69.44
P6	91.67	75.00
Mean	88.89	80.32

The justifications given by participants for their answers can also provide us with some additional information about the acceptance of the tool. Those comments are analyzed in the next paragraphs. The full comments can be reviewed in Table D.1 in Appendix D.

The perceived usefulness comments of the control group describe the editor tested as a useful and powerful tool that helps to quickly create diagrams, effectively decreasing the time spent documenting projects. A participant said that worker's performance can be improved with this tool as it helps them to avoid mistakes because they are easily and quickly identifiable in the diagram preview window.

The justifications for the ease of use of the control group refer to the fact that the PlantUML editor is simple, intuitive, and straightforward to use. The participants value the feedback on the preview diagram window, as it helps clarify how some actions affect the diagram, making it easier to understand the tool's functionality.

The experimental group participants found the visual editor prototype a useful and powerful tool that might help them take less time to make simple, accurate, and understandable architecture diagrams. The tool was found to be great for quickly finding actions to make and seeing properties of an element (like their children). One participant wrote that editing the layout of some elements, like the links between nodes, was a bit difficult sometimes.

The ease of use comments collected in the experimental group describe the prototype as an intuitive and simple tool that is easy to learn how to use. The participants reported that some improvements could be made by adding the action to copy and paste elements and to select link terminals visually. Adding more support to other UML elements and improving the accuracy of selecting elements in the diagram view was also pointed out.

These comments reinforce the idea that the prototype and the previous version have similar acceptability. The suggestions for improvement added in the experimental comments can be justified because the PlantUML editor already received improvements while the prototype was in its first iteration.

6.5.4 System Usability Scale (SUS)

The eleven questions in the evaluation form related to SUS allow us to compare the prototype's usability to the PlantUML editor. The SUS questions defined in [37] are complemented by an additional question allowing participants to comment on their previous choices.

As previously described in Section 6.2.5, the SUS questions request a response representing the level of agreement on a 5-point Likert Scale.

The Student t-test and Mann-Whitney U tests followed the same guidelines as previous analyses, allowing us to form conclusions about the statistical difference between the groups. As in the previous analyses, the level of significance (α) used is 0.05. The means ($\bar{x}$) and standard deviation (σ) for each set of answers to these questions were also calculated.

In SUS, odd questions are positive, while even questions are negative. Given that duality, the tests executed for each question were 'greater' tests if the question was positive and 'lower' tests if the question was negative. The Null Hypotheses (H_0) defined were that the SUS values of the experimental group would be significantly higher than the control group in positive questions and significantly lower in negative ones. The results of this analysis are visible in Table 6.7.

Table 6.7: Statistical difference test for SUS Questions

	Control		Experimental		Student t-test			Mann-Whitney U		
Question	$\bar{x}$	σ	$\bar{x}$	σ	Test	p_value	Accept H_0?	Test	p_value	Accept H_0?
1	3.83	0.75	3.67	1.03	>	0.62	No	-	-	-
2	1.67	0.52	2.33	1.03	-	-	-	<	0.91	No
3	4.83	0.41	4.33	0.52	-	-	-	>	0.96	No
4	2.00	1.26	1.33	0.52	-	-	-	<	0.21	No
5	4.50	0.55	4.33	0.52	-	-	-	>	0.74	No
6	1.17	0.41	1.17	0.41	-	-	-	<	0.55	No
7	4.50	0.55	4.50	0.84	-	-	-	>	0.43	No
8	1.50	0.84	1.33	0.52	-	-	-	<	0.46	No
9	4.17	1.17	4.17	0.75	-	-	-	>	0.67	No
10	1.67	0.82	1.33	0.52	-	-	-	<	0.26	No

- **1.** I think that I would like to use this system frequently.
- **2.** I found the system unnecessarily complex.
- **3.** I thought the system was easy to use.
- **4.** I think that I would need the support of a technical person to be able to use this system.
- **5.** I found the various functions in this system were well integrated.
- **6.** I thought there was too much inconsistency in this system.
- **7.** I would imagine that most people would learn to use this system very quickly.
- **8.** I found the system very cumbersome to use.
- **9.** I felt very confident using the system.
- **10.** I needed to learn a lot of things before I could get going with this system.

No H_0 for the SUS responses was accepted, meaning that no positive questions had significantly greater values in the experimental group than in the control group, and no negative questions

had significantly lower values when comparing the groups. Although there is no significant difference in the answers of both groups, we can formulate conclusions by analyzing the values for $\bar{x}$.

The means of the positive questions (1, 3, 5, 7, and 9) tell us that the experimental group mostly had slightly lower values than the control group, which contradicts what was expected. The values for $\bar{x}$ on the negative questions (2, 4, 6, 8, and 10) were slightly higher in the experimental group for questions 2 and 4, while in questions 8 and 10 they were lower on the experimental group. The answers for question 6 were curiously the same for both groups.

Additional tests were conducted on the questions with resulting p_values higher than 0.90, using the same parameters. Those tests were the opposite of the ones executed before, i.e., if it was a positive question, the new test was a 'lower' one; if it was a negative question, the new test was a 'greater' one. Those additional tests were executed for questions 2 and 3, as their p_values were 0.91 and 0.96, respectively, and can be reviewed in Table 6.8.

Table 6.8: Extra statistical difference test for SUS Questions 2 and 3

	Control		Experimental		Mann-Whitney U		
Question	$\bar{x}$	σ	$\bar{x}$	σ	Test	p_value	Accept H_0?
2	1.67	0.52	2.33	1.03	>	0.12	No
3	4.83	0.41	4.33	0.52	<	0.06	No

Please refer to Table 6.7 to review the SUS Questions.

These additional tests allowed us to conclude that the new H_0 for questions 2 and 3 were also not accepted. That means that even though there's a difference between those groups, it is not statistically significant given both the 'lower' and 'greater' tests. These two questions specifically refer to the fact that the visual editor can be considered slightly more complex than the PlantUML editor (Question 2) and that the prototype might also be slightly less easy to use than the editor used in the control group (Question 3).

The Likert charts in Figure 6.7 visually reinforce the conclusions made in the previous paragraph. With these plots, we can understand that the answers about the usability of the editors used in both groups are indeed not very different.

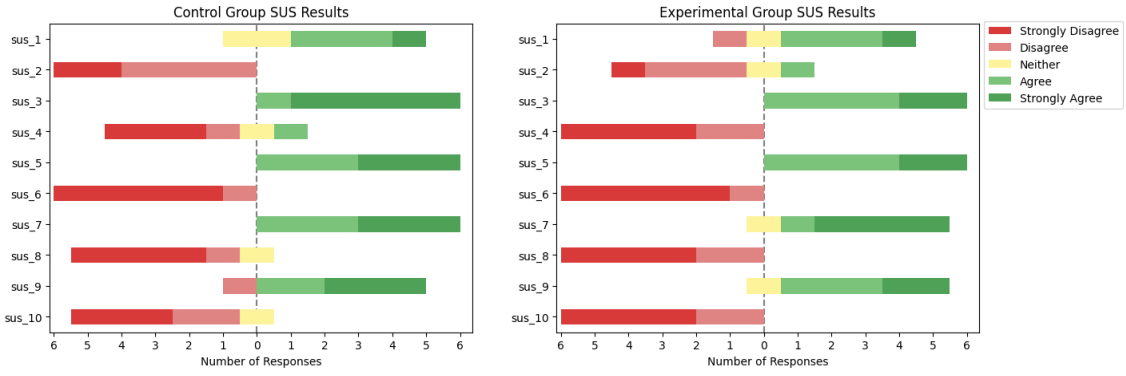


Figure 6.7: Likert answers to SUS Questions

In addition to the previous analyses, the SUS scores given by each participant, which are in a range from 0 to 100, were also calculated using the following formulas:

$$PositiveQuestions(PQ) = (SUS01 + SUS03 + SUS05 + SUS07 + SUS09) - 5$$

$$NegativeQuestions(NQ) = 25 - (SUS02 + SUS04 + SUS06 + SUS08 + SUS10)$$

$$SUS = (PQ + NQ) \times 2.5$$

Following those formulas, first, we calculate the positive and negative question scores for each participant (P). After that, we use those values to compute the individual SUS scores, presented in Table 6.9.

At last, we calculate the final SUS score of the editors tested. The visual editor prototype obtained a slightly lower score, but that difference is not statistically significant according to the tests made before.

The computing of this score is connected with our third research question (RQ3) defined in Section 4.2. RQ3 questioned if the prototype developed achieved a good usability result, i.e. a score of above 70% in the System Usability Scale. If the visual editor reaches an SUS score above that value, we can safely conclude that it provides good usability to users.

Table 6.9: SUS Scores

	Control	Experimental
	Score	Score
P1	80.0	95.0
P2	100.0	95.0
P3	72.50	75.0
P4	82.5	87.5
P5	82.5	77.5
P6	90.0	72.5
Final	84.58	83.75

The justifications written by participants provide additional information about the tool's usability. The main points from those comments are described in the following paragraphs. The full comments can be reviewed in Table D.2 in Appendix D.

The comments made by the control group regarding the usability of the editor they tested revealed that they considered it a usable, intuitive, minimalistic, and straightforward tool. The addition, editing, and deleting of elements was an easy process to complete. The available features are considered to be well-integrated with each other. Some participants revealed that they felt that some improvements could be made to increase the tool's engagement. Those improvements included additions like descriptive icons for each element to make them faster to find and more warnings to confirm actions performed. The participants also referred that the element IDs could be occluded from the interface, and the creation process of connectors could be improved by enabling the creation of new links via each element dropdown.

The experimental group considered the prototype an easy-to-use, intuitive, and not overwhelming tool. The participants considered this editor simpler than other alternatives, doing what is expected and simply what is necessary. One participant wrote that it could be seen as a worthy substitute for other tools because of its low complexity in making architecture diagrams. The frontend of the prototype was thought to be well organized, allowing users to find functionalities and start working as fast as needed easily. One participant revealed that they felt the links could be a little too complex, as their terminals needed to be defined through a dropdown, which is different from most editor alternatives that allow for visually defining these connections.

Both sets of comments depict intuitive, usable, and uncomplicated tools that could improve usability. Given the common points between comments, the prototype and the PlantUML editor can be considered to have similar usability.

6.5.5 Feedback

The last nine questions of the evaluation form made participants reflect on the tool used in the experiment. Participants were asked about the importance of specific features not implemented in at least one of the editors and the UML compliance of the achieved diagram. They are also asked to leave additional remarks, comments, and suggestions about the tool tested.

The answers to these questions allow us to gain insights into whether some specific features are important for the prototype. If the participants consider them indispensable and were not yet available on the prototype, they will be deemed relevant and labeled as future work (cf. Section 7.2).

The first four questions were simple 'Yes'/'No' questions. Figure 6.8 visually represents the 'Yes' votes received for each question. The undo_redo and move_elements questions are self-explanatory, while the keyboard_keys and uml_compliant need more explanation. The keyboard keys are relative to the use of keyboard shortcuts or keys to, for instance, move elements of the diagram or quick access to some specific features. The UML-compliant question asks participants if the final diagram is understandable according to UML standards. The full questions can be reviewed in Section 6.2.5.

The possibility to undo and redo actions was considered indispensable for diagram editors, having all group participants vote 'Yes.' Moving elements was considered a nice-to-have feature, especially in the visual editor prototype. The use of keyboard keys was deemed not as important in the PlantUML editor, but experimental group participants thought it was a good idea for the type of editor they tried. Regarding the UML compliance of the achieved diagram, participants considered the diagram understandable and compliant, even though some might not consider the final result of the prototype to be completely up to UML standards.

The participants were encouraged to explain why they felt each feature was or not important. The main points from those comments are described in the next paragraphs. The comments made by both groups can be viewed in Table D.3 available in Appendix D. Each comment text starts with the answer given by that participant to the corresponding question. The comments of participants

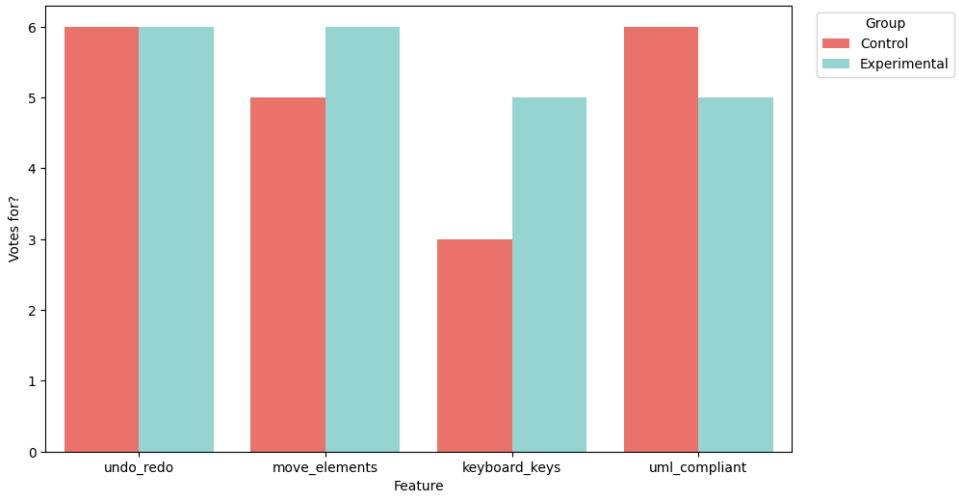


Figure 6.8: Votes for editor features

with an ID starting with 'SB' are from the control group, while the ones starting with 'WM' are from the experimental group.

The control group found that a feature to undo or redo actions was very important to save time and fix unintentional mistakes that could be made. It is useful to redo changes without returning to the original state.

When questioned on the same matter, the experimental group described this feature as a common and useful usability feature that could be considered a standard practice. Participants reported that mistakes are common, and tools should provide the opportunity to quickly fix them, ensuring that the user doesn't waste precious time correcting those actions. This feature is also depicted as a way to increase the tool's intuitiveness and the possibility of experimenting with diagram creation, as it makes it faster to try different options.

Most control group members found that moving elements in the textual editor or the diagram preview would be beneficial. That feature would allow for a better diagram organization, offering users the freedom to pair related elements together. One participant wrote that this feature could improve the users' performance when making diagrams as it helped them organize the elements in a more understanding way.

The prototype testers found this a useful and important feature to ensure the readability and comprehensibility of the diagrams generated. Participants believe that the appearance of the diagrams is very important for their comprehension. For example, correlations can be shown by placing items close to each other. A participant said that having the possibility to readjust the automatic layout helps to improve the visual semantics of the diagram.

Regarding keyboard keys, the participants who tested the PlantUML editor found them useful for shortcuts, even though the tool works better with mouse controls. Participants referred that keyboard keys should be used to increase the tool's accessibility while providing the opportunity to implement shortcuts that increase the performance of frequent and skillful system users.

The experimental group believes keyboard keys are useful for small adjustments on the diagram element's positions. Shortcuts like CTRL+C & CTRL+V or CTRL+Z increase usability, productivity, and intuitiveness for more experienced users, allowing them to execute simple and repetitive tasks faster. Even though it might not be absolutely necessary, the possibility of using the keyboard also increases the tool's accessibility for users who might not use a mouse.

When asked about the UML compliance of the generated diagrams, the control group believes the diagrams are completely understandable, even if they might not completely follow the standards. When using this editor, participants assume the resulting diagram is compliant because it is drawn using the PlantUML language, which was developed to follow those guidelines.

Most experimental group participants found the diagrams generated on the visual editor prototype to be readable and understandable, having all the necessary components to build an architecture diagram, even if it might not be 100% up to the UML standard. One participant wrote that some functionalities of UML, like composition, generalization, and multiplicity, weren't available in the tool, but because they might not be needed for this type of diagram, the tool shouldn't need them to be useful.

The last question of the form prompted participants to leave comments or suggestions, including possible new features they wished to see. The main insights collected are described in the following paragraphs. Table D.4 available in Appendix D shows the full comments received for both groups.

The control group's comments and suggestions mainly cover features or fixes they want to see in the tool. The possibility of analyzing private repositories and other file types from docker-compose is considered to be a good addition. One participant stated that creating GitHub actions and pipelines would greatly increase the possible use cases for the tool. Regarding the interface, participants believed that element hash IDs could be hidden, while small explanations on the types of links available could be added. Participants found the possibility to view a list of the element connections in the element dropdown and the feature of clicking on the view of elements on the diagram to focus them on the textual editor, which also made good improvements.

The experimental group described various features they wished to see implemented in the prototype. Some requests made by multiple participants were Copy & Paste and Undo & Redo features, as well as more customization and a dark theme. Other improvements discussed include an easier way to create and edit new links, a fullscreen mode, a dropdown with search functionality for associating sources and targets to links, a filter on the elements and link lists, and a more controlled delete feature to prevent mistakes and repeating actions. Two participants found small bugs in the prototype, which will be labeled for fixing in future work (cf. Section 7.2). The prototype was also described as a pleasing-to-use and intuitive tool. One participant wrote that it could be a good alternative to paid tools or free alternatives that might be too disconnected from UML or not very optimized.

6.6 Discussion

This experiment was an important tool to gather data that allows us to answer the research questions and analyze the correctness of the hypothesis in the next chapter (cf. Chapter 7). The main conclusions made from the data analysis can be found in the next paragraphs.

The randomly generated experiment groups were considered not statistically different by the analysis conducted on Section 6.5.1. That means the random distribution of participants in the control (CG) and experimental (EG) groups reached a balanced result, which allowed us to execute the experimental sessions confidently.

The data collected on the performance of task execution allows us to compare the editors in terms of how fast and accurate they allowed users to complete common actions, like adding new elements or viewing properties. These values allow us to draw relative conclusions regarding the flow state and cognitive load dimensions of DevEx, as when a task is executed faster, it possibly means that the tool was designed in a way that allows for intuitive and not cumbersome work. Flow state is maintained when the users can easily perform their desired actions in a way that presents them with no setbacks or delays, which can be found in intuitive tools. Cognitive Load is related to the amount of mental effort needed to conclude a task, which is reduced in simple and easy-to-understand tools.

The initial statistical analysis revealed that no experimental task duration was statistically lower than the control ones. Analyzing the mean values (cf. Table 6.4) and box plots, available in Figures 6.4 and 6.5), made it possible to conclude that the experimental group performed all tasks **slightly faster** except for **Task 4**. That difference might be explained by the process of adding new links to the diagram, which is one of the actions requested in that task, taking more steps in the visual editor than in the PlantUML editor. This feature is one of the ones flagged for optimization in future work.

When considering the correctness of task execution, the prototype allowed the participants to complete all tasks correctly, while four of the six CG participants had **incorrect answers** to **Task 1**. These errors might have happened because the PlantUML editor doesn't offer specific information about the number of children elements of a parent, which is easier to obtain in the visual editor prototype.

The questions regarding the Technology Acceptance Model (TAM) allowed the research team to conclude that **the prototype cannot be considered to be more acceptable than the PlantUML editor**. That conclusion was possible by analyzing the scores obtained for each question, which were not significantly greater in the experimental group than in the control group.

In fact, most answers reveal **higher scores on the CG than the EG**, except for questions 7, 8, and 9 regarding perceived ease-of-use (see Table 6.4 to review questions). With further tests, we could conclude that the answers to **question 6 were significantly lower** in the **EG than in the CG**. We hypothesize that might have happened because one EG participant answered 2 ('Quite Disagree' on the 7-point Likert Scale) to all perceived usefulness questions, for considering the

tool not to be useful in their job because they didn't use diagrams on their work. The Likert charts of Figure 6.6 are useful to justify these conclusions further.

The **final TAM scores** obtained are **80.3% for the prototype** and **88.89% for the CG editor**. The EG score is slightly lower than the CG score, but that difference is not statistically significant. Both values predict good acceptance of the tools, given that they are at least 30 percentage points above 50%.

The justifications made by the CG describe the PlantUML editor as a **useful and powerful tool that improves performance by helping people avoid mistakes**. It is also seen as a simple, intuitive, and straightforward tool to use. On the other hand, the prototype is depicted also as a useful and powerful tool that allows users to make simple, accurate, and understandable diagrams in less time. The visual editor is viewed as an **intuitive and simple tool that is easy to learn how to use**. Some participants referenced possible additions like improving the layout editing, adding shortcuts, and including more support for other UML elements.

The answers collected for the System Usability Scale (SUS) questions allowed us to conclude that **the prototype developed cannot be considered to have better usability than the control group editor**. The scores obtained for each question and final SUS scores are not significantly different between groups and, therefore, not significantly better in the prototype.

The mean scores for the questions were **mostly slightly worse for the experimental group**, meaning they were lower in positive and greater in negative questions. The questions and those values can be reviewed in Table 6.7. The answers to questions 8 and 10 had better scores in the EG than in the CG, i.e. had lower scores. Curiously, the answers to question 6 were exactly the same for both groups. These conclusions can also be reviewed in Figure 6.7 Likert charts. The results obtained are evidence of the need for future improvements in the prototype's usability.

The **final SUS scores** obtained are **83.75% for the prototype** and **84.58% for the PlantUML editor**. The EG and CG scores are quite similar; therefore, we can conclude that both editors provide good usability to their users. Actually, because both scores are above 80 and below 85, the grade attributed is A, which means Excellent usability, according to Sauro's SUS tiers [38]. The SUS score obtained for the prototype is even greater than predicted, which was a 70% minimum SUS score.

The justifications made by the CG depict the PlantUML editor as a **usable, intuitive, minimalistic, and straightforward tool**, with some room for improvements. The comments written by EG participants describe the prototype as an **easy-to-use, intuitive, organized, and not overwhelming tool**, which can be improved with, for instance, a better system for creating links.

The last group of questions allowed for gathering insights on extra functionalities and UML compliance of the generated diagrams. A histogram with the votes in favor of those questions can be reviewed in Figure 6.8. Both groups consider the **undo & redo an important feature to save time and quickly fix mistakes**, and the **move elements feature as desirable to better organize and improve the visual semantics of the diagram**. The use of **keyboard keys** is deemed relevant for **improving accessibility** and making it possible to **use shortcuts**, which were referred to be interesting to both editors. The EG found the **use of keyboard keys more relevant than the CG**,

as it allows for **quick and simple improvements to the diagram** that are harder to make with a mouse.

When considering the UML compliance of the diagrams, some EG participants said that the resulting diagram might not be 100% compliant with UML, but the fact that they are simple and quite readable makes it not that important in their opinion. Both groups found the resulting diagrams to be quite understandable and clear.

The final question prompted participants to add more comments and suggestions, which both groups used to detail improvements they would like to see in the editors. The members of the CG referred to additions that could be made to the analyzer, like **allowing to view diagrams from private repositories** and **supporting more file types**. Some participants also requested possible interface improvements, like the removal of hashed IDs and a better way to add and edit connections.

The EG suggested new features for the prototype, like the possibility to **copy & paste** elements and **undo & redo** actions. Other suggestions were **more customization of colors and fonts**, easier ways to create and edit links visually, **filters for the lists** and **search features for link terminal dropdowns**, and an **improved delete feature that reduced the need to redo conditionally deleted elements**. Overall, **the participants found the prototype to be a pleasing and intuitive tool**, which could be a **good alternative** to other diagram editors available.

All data collected depicts the prototype as a **useful, intuitive, and not overwhelming tool**, even if it might not be as acceptable and with as high usability as the PlantUML editor. The comments and suggestions collected allow us to formulate some additions that can be made in future work to improve this visual editor further, which can be reviewed in Section 7.2 of the next chapter.

The results obtained were different than our initial prediction. We believe the results might have been influenced because participants unconsciously compared the prototype with other similar tools that they knew. The polishment level of the visual editor prototype was also lower than that of the PlantUML editor, which had more development iterations on top of it. Therefore, some actions or tasks might be easier to execute in the textual editor.

6.7 Threats to validity

Across all stages of the experiment, we were mindful of possible threats to the validity of our evaluation. Those threats and their proposed management approaches are detailed below:

- **Different skill sets**

  The experiment planning assumes that both groups are statistically equivalent, i.e., participants have similar skill sets. While the groups were homogeneous regarding academic knowledge, because all participants were 2nd-year MIEC at FEUP students, some participants could possess additional skills that could benefit the experiment's performance. The Profile Form questions (available in Section 6.2.4) were designed to capture these possible

differences, but the question set was relatively small (i.e., 5 questions). Considering that information, it is possible that some information that could influence the balance of the groups and the final results was not captured by that questionnaire.

A possible solution to mitigate that threat is to add more questions to collect additional information on participant background to further ensure the balance of skill sets and knowledge of the control (CG) and experimental groups (EG).

- **Features available**

  An experiment version of the PlantUML editor and visual editor prototype was prepared to offer participants the same features. Some features were hidden or altered in the editor to decrease the probability of participants making mistakes. Even with this careful preparation, the editor used on the CG had two additional features (undo action and restore to original state) not referred to in the initial demonstration that could increase participants' performance compared to the EG in case they were found.

  The proposed management for this approach would be to either disable those features on the PlantUML editor or add them to the visual editor. Because they are features deemed relevant by participants, the most viable solution will be adding them to the prototype.

- **Participants' mindset**

  The answers given by participants might have been a bit altered because they all were colleagues of the experiment coordinator. That being said, it is possible that results for the TAM and SUS questions might have been inflated to achieve better results, influencing the final outcome.

  To tackle this threat, the possibility of organizing a new study with a more variable set of students is a good idea. Those possible participants would possibly not be contacted as directly as this study participant, which means a new way of communication needed to be defined. By contacting possible interested parties via academic email, for example, we could ensure that those specific influences are minimized.

- **Generalization**

  The experiment and control groups can be considered too small and homogeneous, making them possibly unusable for an accurate generalization to the community. In addition, because the participants are students with less than 6 years of professional experience, it is harder to generalize the conclusions drawn to the software engineering community. Considering the number of participants, it can also be difficult to generalize these results to other diagramming tools currently available.

  A possible solution to this problem is to increase the sample size by inviting people from different backgrounds and career experiences to participate in a new experiment. By doing so, the conclusions drawn would be a more accurate and valuable representation of the software engineering community.

6.8 Empirical Package

The replication package for this empirical study is available in a GitHub repository [2], which includes the evaluation experiment protocol, the profile and evaluation form pdfs, the Jupyter notebooks created to analyze the experiment groups, and the results obtained from the evaluation form. All the data collected with the forms made available to participants is also available there.

This repository contains a *README.md*, the experiment protocol PDF, and three folders with contents described below:

- **data** - This folder contains two subfolders (**control** and **experimental**) that store the answers collected on both versions of the evaluation forms and the data available in the pull requests created by each participant in their **repo_files** folder. These data are used to execute the statistical analysis of the participant's responses and evaluate the correctness of the tasks they executed.
- **forms** - This folder contains the **three PDF files** of the **Google Forms** created in the context for this experiment, which is the profile questions form and both versions of the evaluation forms.
- **notebooks** - This folder contains the **groups** and **evaluation** Jupyter notebooks. The **groups** notebook is where the **the profile questions** form is analyzed, while the **evaluation** notebook stores the **analysis of the data collected on both evaluation forms**.

The README file contains a comprehensible description of the contents of the empirical package and how to use them to replicate the experiment. The Evaluation Experiment Protocol PDF is a document used to plan the experiment and contains a detailed description of the several stages of planning and execution.

The forms can be used to obtain more data for a similar experiment, while the notebooks can be executed with possible new data collected to reach updated versions of the tables and plots generated.

[2] Replication Package GitHub Repository available at https://github.com/mbarra29/infragenie-visual-rep-pckg.

Chapter 7

Conclusion

The work developed in the context of this dissertation started with a systematic literature review (Chapter 3) conducted on practices and improvements in developer experience and usability for visual editors. The literature review questions we hoped to find answers to were the following:

- **Q1**: What improves the Developer Experience in a visual model editor?
  - **Q1.1**: What improves the several dimensions of DevEx in a visual editor?
  - **Q1.2**: What are good practices for designing visual editors?

This State-of-the-art review allowed the discovery of relevant methods and practices that are important for visual diagram editors. With those results in mind, the development of a visual editor prototype was proposed to improve the developer experience of software architecture diagram-making tools, and the following hypothesis was defined (Chapter 4):

> "A visual model editor that allows adding manual changes to automatically generated diagrams would improve the developer experience of creating and maintaining software architecture documentation."

To gather evidence to attempt to validate the hypothesis defined a visual editor prototype was proposed (Chapter 5). This tool was developed on top of the previous version of Infragenie, and its main focus was on acceptance by the community, usability, and DevEx. Because of the visual nature of this editor, it was expected to have positive effects on the DevEx of users when compared to the PlantUML editor, as well as being more accepted by the community and achieving a good System Usability Scale score.

Considering the high focus on user experience and usability, an empirical study with an Experiment (Chapter 6) was conducted to gather the information needed to evaluate the prototype accurately. The experiment was executed with twelve students of the Master in Informatics and Computing Engineering (MEIC) at the Faculty of Engineering of the University of Porto (FEUP) divided into control and experimental statistically equivalent groups. The results obtained with the evaluation questionnaire made available during the experiment were analyzed to form concluded that allows us to answer the research questions defined in Section 4.2.

- **RQ1 - What are the positive effects of using a visual editor instead of a textual editor for editing diagrams that are automatically generated by Infragenie?**

  The visual editor prototype was slightly faster on average than the PlantUML textual editor. It allowed users to understand the presented information more accurately, resulting in a higher percentage of overall correct answers than the textual editor.

  With those conclusions, we can hypothesize that the **cognitive load of participants of the experimental group** (EG), which used the visual editor, **can be slightly lower than the ones from the control group** (CG). Participants from both groups described the editors they tested as intuitive, straightforward, easy to use, organized, and not overwhelming tools. This can mean that there is **no significant difference in the flow state and feedback loops dimensions of DevEx when comparing results from both editors**.

- **RQ2 - Can a visual editor be more accepted by the community when compared to the previously available editor?**

  The Technology Acceptance Model Score for the visual editor (80.3%) is lower than the PlantUML editor (88.89%); however, that difference is not statistically significant.

  With this information, we can **hypothesize that the two types of editors have similar acceptability**. The visual editor prototype developed might have a lower acceptability because it is in a more initial state, while the PlantUML editor already received some improvements from its first version.

- **RQ3 - Does the developed editor achieve a good usability result (above 70%) in the System Usability Scale (SUS)?**

  The SUS score obtained by the **visual editor (83.75%) is a value above 70%**, even if it is slightly lower than the one obtained by the PlantUML Editor (84.58%).

  The score obtained by the prototype is considered an **'A' grade**, which means **'Excellent' usability** according to Sauro's tiers [38]. This score is a good outcome of the experiment because it establishes the prototype as a tool with good usability which can be translated to a good user experience.

Unfortunately, the hypothesis referred to before **cannot be fully accepted**, as the results indicate that the **visual editor is not an improvement on the previous version**. However, the data collected sustains that **visual editors can be seen as a worthy alternative to textual editors**, which can be further improved with more visually dependent features.

The improvements were not as clear as expected but taking into account the different polishment levels and features available, they can be considered relevant results. We believe that with additional work on top of the visual editor prototype, some benefits can become clearer.

Developers can be more visual or textual-oriented. Therefore, each person can find different advantages and disadvantages in the PlantUML textual editor and the visual editor. With that in mind, we believe that the best possible approach for these types of tools is to offer users the

option to choose which editor they prefer at any given point in time while providing a seamless experience between them.

The feedback collected from the experiment allowed the research team to find additional relevant future work, described in Section 7.2.

7.1 Contributions

The contributions of this thesis to the software engineering community are the following:

- **State-of-the-Art review**

  An exploration and description of methods and practices to enhance DevEx and usability of diagram-based software architecture documentation tools, including visual model editors.

- **Visual Editor for Infragenie**

  The prototype developed on top of Infragenie to improve the usability and DevEx of adding changes to automatically generated architecture diagrams.

- **Empirical study**

  An in-person experiment executed with 12 MIEC students from FEUP, which allowed us to evaluate the prototype developed. This study can be replicated using the empirical package available in GitHub (more information on Section 6.8).

7.2 Future Work

Several possible improvements to the work conducted in this thesis were identified. They can be mostly divided between Infragenie improvements, visual editor prototype improvements, and empirical study improvements.

Infragenie improvements

- **Tool enhancements**:

  Members of the control group refer that it would be interesting to have the possibility to analyze private repositories. The support for analysis of other infrastructure file types and the creation of a GitHub Action or pipeline for automatic diagram generation after changes are made to the code were also suggestions given by the CG.

- **User Interface**:

  - Adding descriptive icons to find elements faster in the editor, more warnings to give feedback to the users on actions executed, and small descriptions of the available link types.

  - Add keyboard shortcuts and improve the system to create links (using the element dropdowns).

- Add information on the element connections closer to the element textual representation.

- Make it possible to reorganize the text in the editor.

- Focus on the element text by clicking on its preview on the diagram.

- Remove the Hashed IDs from the frontend.

Visual Editor Prototype Improvements

- **Features to add** (the ones that were referred by participants contain a + symbol)

  - Perform a better automatic layout upon loading the graph.

  - Undo and Redo changes. +

  - Revert to the original diagram state.

  - Copy and Paste diagram elements. +

  - Save the image of the diagram to the computer.

  - Improve the modal for adding new elements defining all relevant properties when creating (name, source, target, ...) +(bug fixing).

  - Zoom-in and Zoom-out.

  - Context menu when clicking elements to edit their properties.

  - Keyboard keys to make small changes in the element's position. +

  - Keyboard shortcuts for frequent actions. +

  - Highlight authorship of diagram elements (whether they were automatically generated or changed by users).

  - Seamless experience between editors with compatibility of changes made (to provide a better experience for all types of users).

  - Select link terminals visually instead of via dropdown. +

  - Improve the accuracy of the element selection in the diagram view. +

  - Add search functionality to the link source and target definition dropdown. +

  - Filter by element type in the elements and links lists. +

  - Improve the delete functionality to delete only what is selected, not the dependent links and children. Add a prompt to ask if users want to delete those. +

  - Fix issues with opening properties views. +

  - Fix issues with moving link layouts. +

  - Further UML support for other types of diagrams. +

  - More customization by enabling different fonts and colors. +

  - Dark theme +

- Fullscreen mode for editor window. +

Empirical Study Improvements

- **Increase sample size and diversity**

  Increasing the sample size and diversity by expanding the experience to more participants with different professional and academic backgrounds can benefit the validity of the results. With those improvements, we can have more confidence in our conclusions as they become more generalizable for the software engineering community and other diagramming tools.

- **Decrease difference between tested editor**

  Improving the visual editor so that it includes the undo & redo and return to original features allows for more control over the results obtained. Those features were only available for the PlantUML editor in the experiment and were not described in that demonstration. Even if they weren't referred, those features could have influenced the results simply because they were available, while in the visual editor, they were not.

- **Develop theories on what impacts DevEx**

  We believe that new theories regarding what impacts the Developer Experience of diagramming tools can also be developed by using the results collected with our study as well as possible future results.

References

[1] C. Martraire, *Living documentation: Continuous knowledge sharing by design*. Addison-Wesley Professional, 2019. [Online]. Available: https://books.google.com/books?hl=pt-PT&lr=&id=8_6ZDwAAQBAJ&oi=fnd&pg=PT28&dq=Cyrille-Martraire+%22Living+Documentation%22&ots=SrBu62YSXi&sig=ULeE3Vg_5boE4mKoCyeLnOL2hPQ

[2] F. F. Correia, A. Aguiar, H. S. Ferreira, and N. Flores, "Patterns for consistent software documentation," in *Proceedings of the 16th Conference on Pattern Languages of Programs*, 2009, pp. 1–7.

[3] J. Santos and F. F. Correia, "A review of pattern languages for software documentation," in *Proceedings of the European Conference on Pattern Languages of Programs 2020*, 2020, pp. 1–14.

[4] J. Santos and F. Correia, "Patterns for documenting open source frameworks," in *Proceedings of the Conference on Pattern Languages of Programs 2021*, 2021.

[5] D. E. Knuth and S. Levy, *The WEB system of structured documentation*. Department of Computer Science, Stanford University, 1983.

[6] L. Friendly, "The design of distributed hyperlinked programming documentation," in *Hypermedia Design: Proceedings of the International Workshop on Hypermedia Design (IWHD'95), Montpellier, France, 1–2 June 1995*. Springer, 1996, pp. 151–173.

[7] A. Aguiar, G. David, and M. Padilha, "Xsdoc: an extensible wiki-based infrastructure for framework documentation." in *JISBD*, 2003, pp. 11–24.

[8] F. F. Correia, H. S. Ferreira, N. Flores, and A. Aguiar, "Incremental knowledge acquisition in software development using a weakly-typed wiki," in *Proceedings of the 5th International Symposium on Wikis and Open Collaboration*, 2009, pp. 1–2.

[9] F. A. P. de Figueiredo Correia, "Documenting software with adaptive software artifacts," Ph.D. dissertation, Universidade do Porto (Portugal), 2015.

[10] A. Tang, M. Razavian, B. Paech, and T.-M. Hesse, "Human aspects in software architecture decision making: a literature review," in *2017 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2017, pp. 107–116.

[11] N. A. Ernst and M. P. Robillard, "A Study of Documentation for Software Architecture," May 2023, arXiv:2305.17286 [cs]. [Online]. Available: http://arxiv.org/abs/2305.17286

[12] F. F. Correia, A. Aguiar, H. S. Ferreira, and N. Flores, "Patterns for consistent software documentation," in *Proceedings of the 16th Conference on Pattern Languages of Programs*. Chicago Illinois USA: ACM, Aug. 2009, pp. 1–7. [Online]. Available: https://dl.acm.org/doi/10.1145/1943226.1943241

[13] M. Wybrow, "Using semi-automatic layout to improve the usability of diagramming software," PhD Thesis, Citeseer, 2008. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=87ec44478cc75f6bf6d6b4fd18c2b5c90eff955b

[14] R. Kalantari and T. C. Lethbridge, "Characterizing ux evaluation in software modeling tools: A literature review," *IEEE Access*, 2022, publisher: IEEE. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9973242/

[15] Object Management Group, Dec 2017. [Online]. Available: https://www.omg.org/spec/UML/2.5.1/About-UML/

[16] "Lucidchart - what is uml," Oct 2016. [Online]. Available: https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language

[17] J. Mayr, "Application of paradigms of HCI and graphical modelling to improve usability of graphical UML modelling editors: Assessing changes to hierarchy, diagram composition, and navigation compared to an existing editor," PhD Thesis, FH Vorarlberg (Fachhochschule Vorarlberg), Jul 2022. [Online]. Available: https://opus.fhv.at/frontdoor/index/index/docId/4541

[18] R. J. de Araújo Ferreira, "Recovery of Software Architecture from Code Repositories," Master's thesis, Faculty of Engineering of the University of Porto, 2022. [Online]. Available: https://repositorio-aberto.up.pt/handle/10216/145130

[19] F. F. Correia, R. Ferreira, P. G. G. Queiroz, H. Nunes, M. Barra, and D. Figueiredo, "Towards living software architecture diagrams," 2024. [Online]. Available: https://arxiv.org/abs/2407.17990

[20] A. Terekhov, T. Bryksin, and Y. Litvinov, "How to Make Visual Modeling More Attractive to Software Developers," in *Present and Ulterior Software Engineering*, M. Mazzara and B. Meyer, Eds. Cham: Springer International Publishing, 2017, pp. 139–152. [Online]. Available: http://link.springer.com/10.1007/978-3-319-67425-4_9

[21] J. R. Rasure and C. S. Williams, "An integrated data flow visual language and software development environment," *Journal of Visual Languages & Computing*, vol. 2, no. 3, pp. 217–246, 1991.

[22] P. Caire, N. Genon, P. Heymans, and D. L. Moody, "Visual notation design 2.0: Towards user comprehensible requirements engineering notations," in *2013 21st IEEE International Requirements Engineering Conference (RE)*. IEEE, 2013, pp. 115–124.

[23] B. Piedade, J. P. Dias, and F. F. Correia, "An empirical study on visual programming docker compose configurations," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2020, pp. 1–10.

[24] ——, "Visual notations in container orchestrations: an empirical study with docker compose," *Software and Systems Modeling*, vol. 21, no. 5, pp. 1983–2005, 2022.

[25] "Plantuml at a glance." [Online]. Available: https://plantuml.com/

[26] M. A. Teruel, E. Navarro, V. López-Jaquero, F. Montero, and P. González, "A CSCW requirements engineering CASE tool: development and usability evaluation," *Information and Software Technology*, vol. 56, no. 8, pp. 922–949, 2014, publisher: Elsevier. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584914000512?casa_token=fB3G5Nt9XlsAAAAA:a9pJPGhWp_zbmEWCmx9gunAlpXRDdtGh6v6M48oBNN86Azp6_Km0rVc0va_QWFm6dQLZozJ85w

[27] A. Noda, M.-A. Storey, N. Forsgren, and M. Greiler, "DevEx: What Actually Drives Productivity: The developer-centric approach to measuring and improving productivity," *Queue*, vol. 21, no. 2, pp. 35–53, Apr. 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3595878

[28] A. Nylund, "A multivocal literature review on developer experience," Master's thesis, Aalto University - School of Science, 2020. [Online]. Available: https://aaltodoc.aalto.fi/handle/123456789/43567

[29] B. A. Myers, A. J. Ko, T. D. LaToza, and Y. Yoon, "Programmers Are Users Too: Human-Centered Methods for Improving Programming Tools," *Computer*, vol. 49, no. 7, pp. 44–52, Jul. 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7503516/

[30] N. Condori-Fernández, J. I. Panach, A. I. Baars, T. Vos, and O. Pastor, "An empirical approach for evaluating the usability of model-driven tools," *Science of computer programming*, vol. 78, no. 11, pp. 2245–2258, 2013, publisher: Elsevier. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642312001451

[31] H. Suryawirawan, "Tech lead journal: [best of 2023] #134 - a developer-centric approach to measuring and improving productivity - margaret-anne storey & abi noda em apple podcasts," Jan 2024. [Online]. Available: https://podcasts.apple.com/pt/podcast/tech-lead-journal/id1523421550?i=1000641302071

[32] A. E. Bobkowska and K. Reszke, "Usability of UML Modeling Tools." *Software engineering: evolution and emerging technologies*, vol. 5, 2005. [Online]. Available: https://books.google.com/books?hl=pt-PT&lr=&id=dv7uAgAAQBAJ&oi=fnd&pg=PA75&dq=(UX+OR+%22user+experience%22+OR+DevEx+OR+%22developer+experience%22+OR+usability)+AND+(%22model+editor%22+OR+%22model+editors%22+OR+%22diagram+editor%22+OR+%22diagram+editors%22)&ots=HwxawG1hLo&sig=GQZSjiA-M05WttHoKtMN2-BV4Jw

[33] ACM SigSOFT. [Online]. Available: https://www2.sigsoft.org/EmpiricalStandards/docs/?standard=SystematicReviews

[34] F. D. Giraldo, S. España, R. Y. Rivera, J. A. Onofre, W. J. Giraldo, and O. Pastor, "Exploratory usability evaluation of the capability-design tool," in *2017 36th International Conference of the Chilean Computer Science Society (SCCC)*. IEEE, 2017, pp. 1–9. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8405118/

[35] S. N. Orni, "Development and Usability Evaluation of a Domain-Specific Modeling Tool for Cybersecurity-related User Journeys," Master's thesis, University of Oslo - Faculty of Mathematics and Natural Sciences - Department of Informatics, 2022. [Online]. Available: https://www.duo.uio.no/handle/10852/100305

[36] J. R. Lewis, "Comparison of four tam item formats: Effect of response option labels and order - jux," Aug 2019. [Online]. Available: https://uxpajournal.org/tam-formats-effect-response-labels-order/

[37] J. Brooke, "SUS: A quick and dirty usability scale," *Usability Eval. Ind.*, vol. 189, Nov 1995. [Online]. Available: https://www.researchgate.net/publication/228593520_SUS_A_quick_and_dirty_usability_scale

[38] J. Sauro, "5 ways to interpret a sus score," Sep 2018. [Online]. Available: https://measuringu.com/interpret-sus-score/

[39] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. [Online]. Available: http://link.springer.com/10.1007/978-3-642-29044-2

[40] S. Baltes and P. Ralph, "Sampling in software engineering research: a critical review and guidelines," *Empirical Software Engineering*, vol. 27, no. 4, p. 94, Jul 2022. [Online]. Available: https://link.springer.com/10.1007/s10664-021-10072-8

[41] Student, "The probable error of a mean," *Biometrika*, pp. 1–25, 1908.

[42] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[43] S. Shaphiro and M. Wilk, “An analysis of variance test for normality,” *Biometrika*, vol. 52, no. 3, pp. 591–611, 1965.

Appendix A

Infragenie Mockups

The mockups for the Infragenie Interface were developed using Figma. These designs were useful for updating the web app after the refactoring and developing the visual editor prototype. All pages were designed with a full view and side-by-side view in mind for developers multitasking and having multiple apps open on their screens.

First, an analysis was made of the identity of Infragenie. That process allowed for identifying the web application's main colors and for defining an updated logo and web app icon. This step saved time in the next developments, as it was easier to find the colors to use, given that they were identified here. Figure A.1 displays this small identity study and the logos developed.

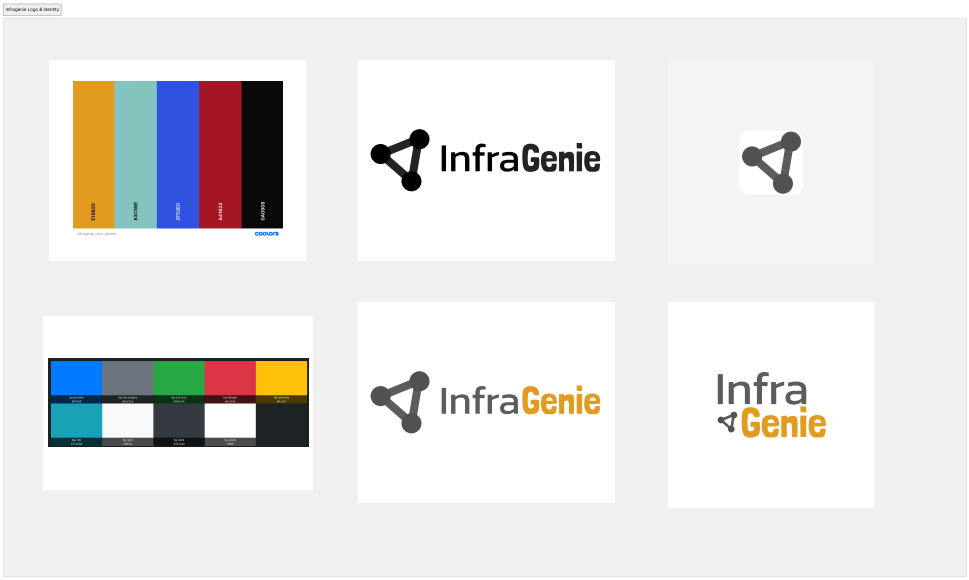


Figure A.1: Infragenie Logos and Identity

After that, the 'Bad Repo' and 'Not Found' pages were idealized. The main goal was to simplify these pages while ensuring their interface was as close to the other web app pages as possible. The mockups for these pages can be viewed in Figure A.2.

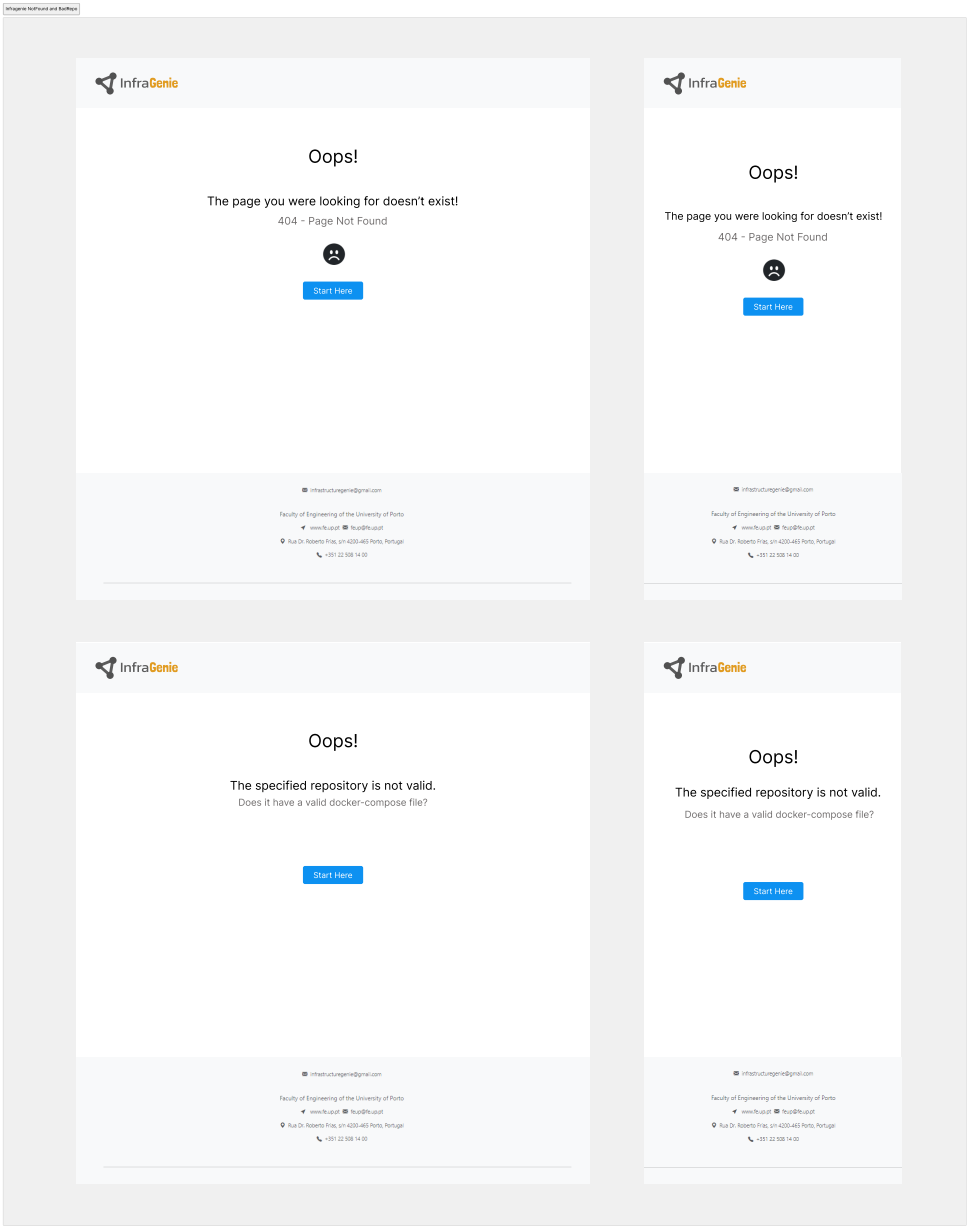


Figure A.2: Infragenie New Pages Mockups

The Infragenie Homepage was also reviewed, mainly in terms of CSS. The mockup served as a helpful guide to rearranging the buttons and textual elements of this page. Figure A.3 contains the mockups designed for the homepage.

A general mockup was developed for the PlantUML Editor so it could become easier to visualize the planned updates. Using the Figma app, the best way to display the side-by-side view of the textual editor and the diagram preview was found after multiple reviews by the research team. This page also allows for a 'minimized' view, as visible in Figure A.4.

At last, the visual editor mockup was designed. This visualization allowed us to easily plan the positioning of planned features while predicting the workflow of the prototype. The mockup

displayed in Figure A.5 is not extensive, but it was still quite helpful for the development process.

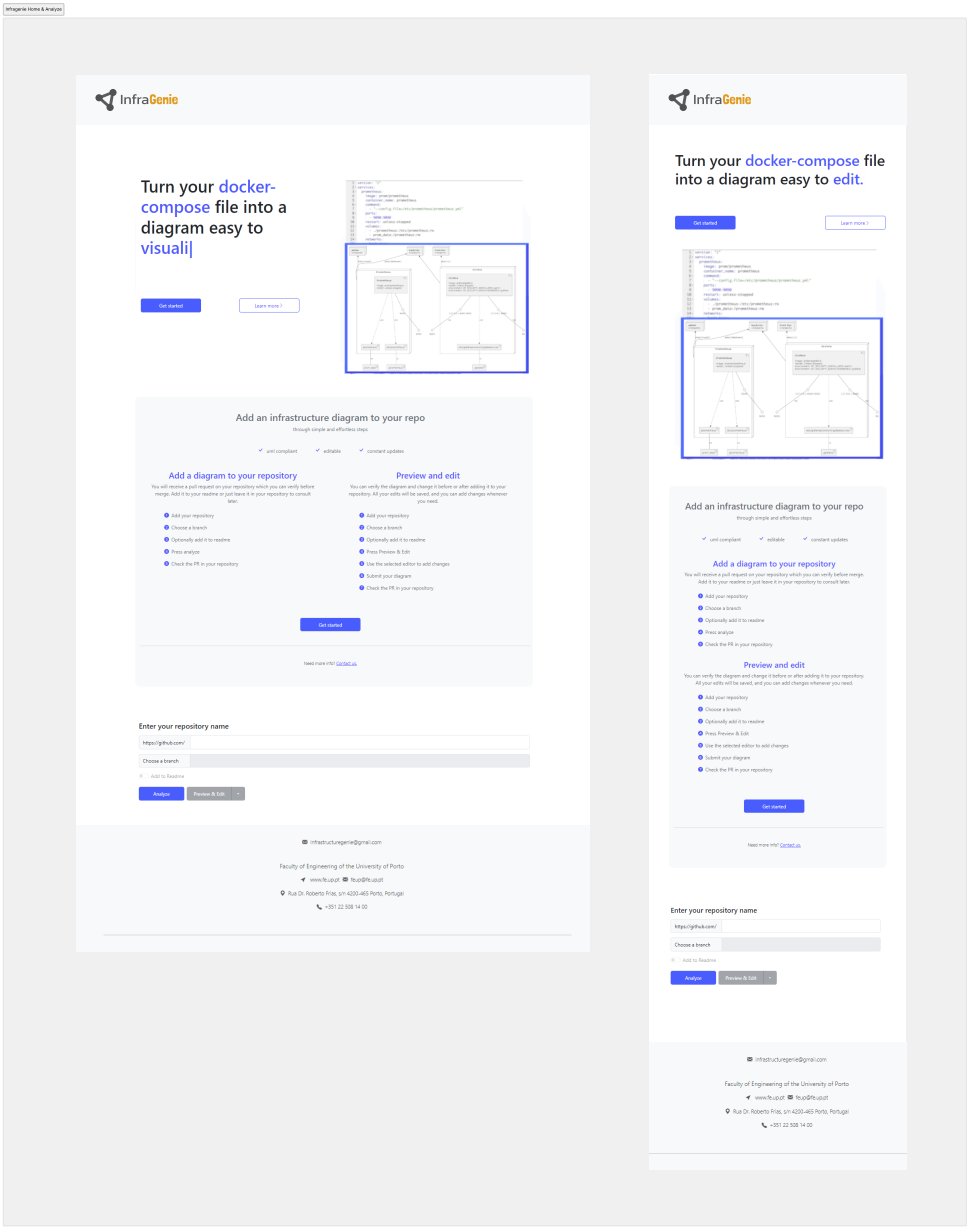


Figure A.3: Infragenie Homepage Mockups

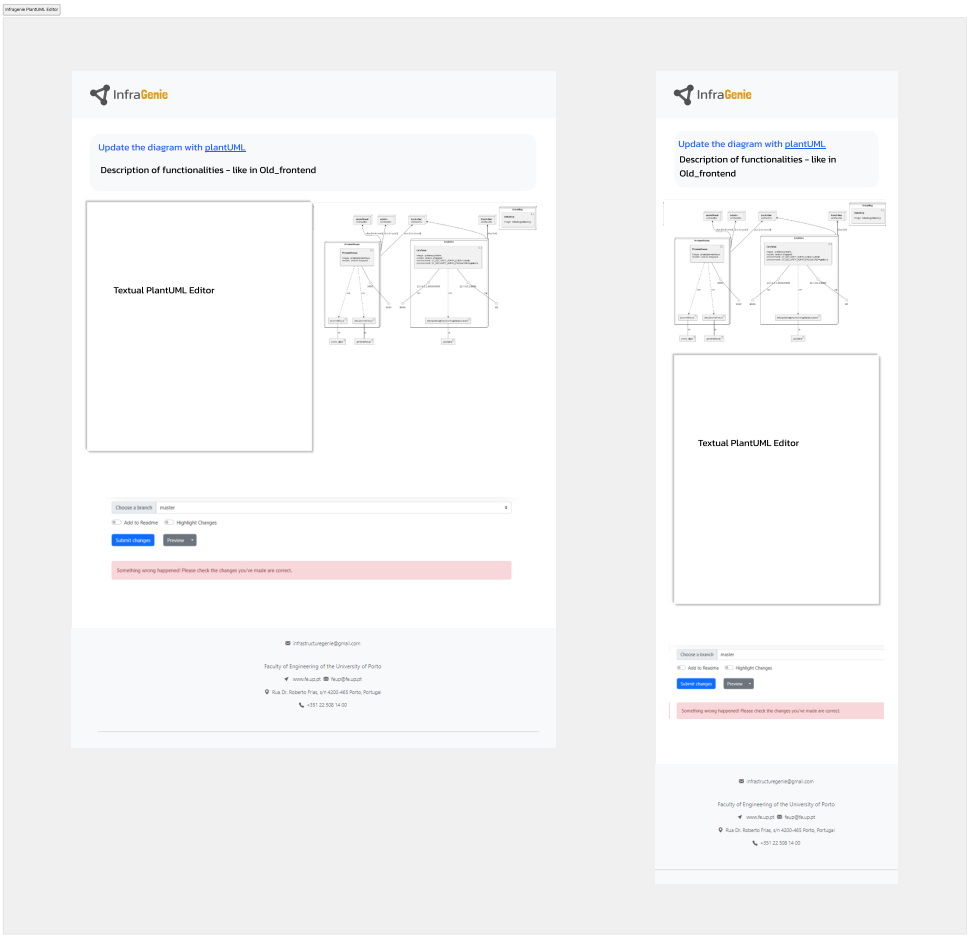


Figure A.4: Infragenie PlantUML Editor Mockups

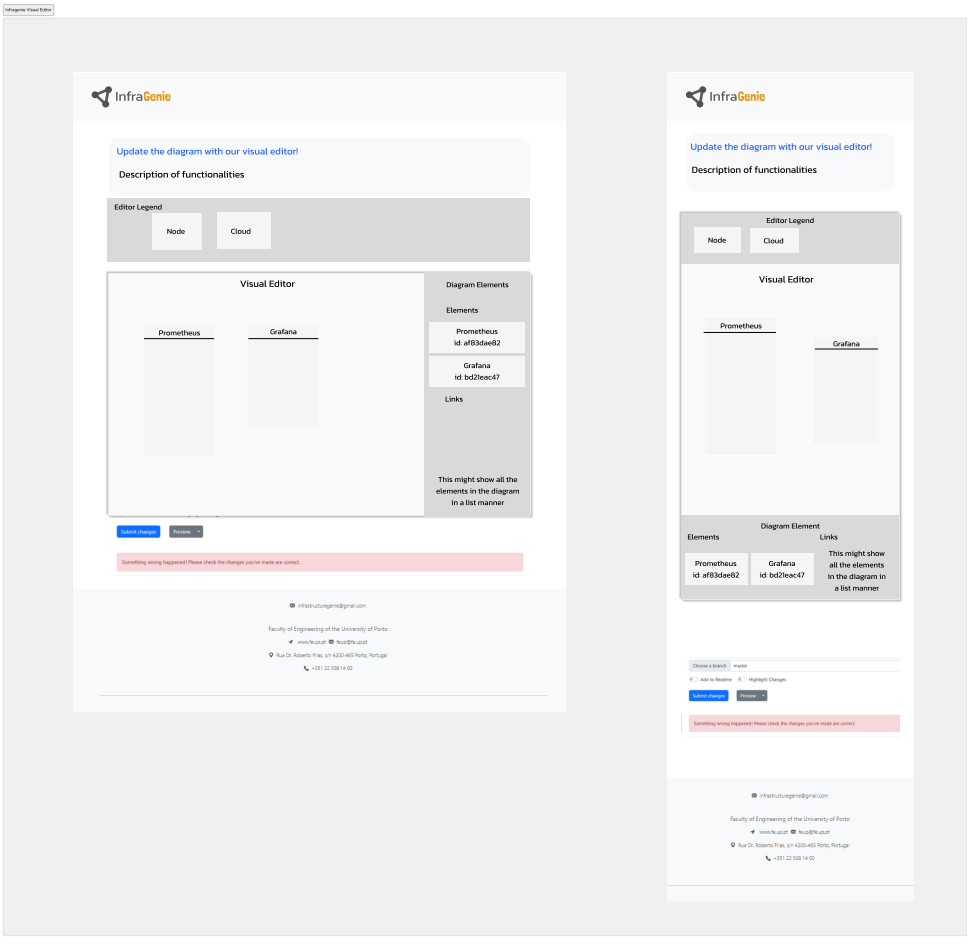


Figure A.5: Infragenie Visual Editor Mockups

Appendix B

Profile Questions Form

Infragenie Profile Questions

Thank you for agreeing to participate in this experiment.
All data provided in the form is anonymous and will only be used for the scope of this experiment to help evaluate the tool developed and find ways to improve it.
The results will be published only in aggregate form and used to derive statistical data about the tool.

This form contains questions to help characterize and better understand our study groups.

* Indicates required question

1. Do you have any **professional experience in Software Development**? *

   Mark only one oval.

   - ◯ Yes
   - ◯ No

2. If you answered "**Yes**" to the previous question, for how long?

   Mark only one oval.

   - ◯ Less than 1 year
   - ◯ 1 to 2 years
   - ◯ 2 to 4 years
   - ◯ 4 to 6 years
   - ◯ 6 or more years

3. Experience with **diagram-making software and languages** *

Rate your level of agreement with the following statements.

Mark only one oval per row.

	Strongly Disagree	Disagree	Neither	Agree	Strongly Agree
I have experience with tools that allow the creation of diagrams from plain text or code (e.g. PlantUML, Mermaid.JS, others).					
I have experience with tools that allow the creation of diagrams through a visual editor (e.g. draw.io, Lucidchart, others)					
I am familiar with with the Unified Modelling Language (UML).					
I have experience creating architecture diagrams.					

This content is neither created nor endorsed by Google.

Google Forms

Appendix C

Evaluation Form

C.1 Control Group (PlantUML Editor)

Infragenie Evaluation Form

Thank you for agreeing to participate in this experiment.

All data provided in the form is anonymous and will only be used for the scope of this experiment to help evaluate the tool developed and find ways to improve it.

The results will be published only in aggregate form and used to derive statistical data about the tool.

* Indicates required question

Introduction to the visual editor

The visual editor you will use allows you to edit a diagram of the architecture of a software system using the PlantUML language. The diagram is automatically generated from a docker-compose file in the repository analyzed by the tool. You can also preview the resulting diagram on the plantUML diagram page.

You can move on to the next section now.

Good Luck!

Tasks to complete

This section contains several tasks for you to complete as **correctly** and **quickly** as possible.
Please read the instructions **carefully**.

We will ask you to **record the specific time** of the **start of each task** and the **end time** of the **last task**.

After completing each task, **return to this form** to continue the experiment.

1. Please **insert your assigned ID**. *

Experiment Setup
Please open the editor. To do so, enter the **repository name** and corresponding **branch** (see below) on the **form** of the tool's **homepage** and click on the "**Preview & Edit**" button.

Below, we present the relevant data to input:

Repository: mbarra29/infragenie-tests
Branch: strawberry

2. **Start of Task 1** *

Please record the time you reached this stage.

Example: 8:30 AM

3. **Task 1** *

Open the properties of the **Prometheus Node**.
How many children does this node have?

Mark only one oval.

- 1
- 2
- 3
- 4
- 5

4. **Start of Task 2** *

Please record the time you reached this stage.

Example: 8:30 AM

5. **Task 2** *

Open the **Grafana Component** and **add a new property** to it.
What is the total number of properties **after** your addition?

Mark only one oval.

- 1
- 2
- 3
- 4
- 5

6. **Start of Task 3** *

Please record the time you reached this stage.

_____
Example: 8:30 AM

7. **Task 3** *

Delete the "**Back-tier**" **cloud/network**.
How many **connections/links** were **deleted** because of that initial deletion?

Mark only one oval.

- 1
- 2
- 3
- 4
- 5

8. **Start of Task 4** *

Please record the time you reached this stage.

_____
Example: 8:30 AM

Task 4

Create a new element of a type of your choosing and **add a new link** connecting that **new element to the Datadog Node**.

9. **Start of Task 5** *

   Please record the time you reached this stage.

   _____

   Example: 8:30 AM

Task 5

Open **one element of your choosing** and edit **one of the editable properties** (name, children, properties, ...) to your liking.

10. **Start of Task 6** *

    Please record the time you reached this stage.

    _____

    Example: 8:30 AM

Task 6

Please submit the changes made, **ensuring the whole diagram is visible in the preview window**. In the dropdown, select the branch previously mentioned, check the “Add to Readme” checkbox, and click the “Submit Changes” button.

After clicking the button, you will be prompted to **insert your assigned ID**.

11. **End of the experimental tasks** *

    Please record when you reached the end of the tasks here.

    _____

    Example: 8:30 AM

Evaluation Questions

In this section, you will be asked to **answer several questions** that will allow us to evaluate the tool thoroughly.

Please **answer carefully and truthfully**.

Technology Acceptance Model (TAM)

Express your agreement with the following items, considering t**he job of creating and maintaining architectural documentation**.

12. Perceived Usefulness *

Mark only one oval per row.

	Strongly Disagree	Quite Disagree	Slightly Disagree	Neither	Slightly Agree	Quite Agree	Strongly Agree
Using this tool in my job would enable me to accomplish tasks more quickly.	⬭	⬭	⬭	⬭	⬭	⬭	⬭
Using this tool would improve my job performance.	⬭	⬭	⬭	⬭	⬭	⬭	⬭
Using this tool in my job would increase my productivity.	⬭	⬭	⬭	⬭	⬭	⬭	⬭
Using this tool would enhance the effectiveness on the job.	⬭	⬭	⬭	⬭	⬭	⬭	⬭
Using this tool would make it easier to do my job.	⬭	⬭	⬭	⬭	⬭	⬭	⬭
I would find this tool useful in my job.	⬭	⬭	⬭	⬭	⬭	⬭	⬭

13. Please justify your answers regarding the **usefulness** of the tool. *

14. Perceived Ease-of-use *

Mark only one oval per row.

	Strongly Disagree	Quite Disagree	Slightly Disagree	Neither	Slightly Agree	Quite Agree	Strongly Agree
Learning to operate this tool would be easy for me.	◯	◯	◯	◯	◯	◯	◯
I would find it easy to get this tool to do what I want it to do.	◯	◯	◯	◯	◯	◯	◯
My interaction with this tool would be clear and understandable.	◯	◯	◯	◯	◯	◯	◯
I would find this tool would be clear and understandable.	◯	◯	◯	◯	◯	◯	◯
It would be easy for me to become skillful at using this tool.	◯	◯	◯	◯	◯	◯	◯
I would find this tool easy to use.	◯	◯	◯	◯	◯	◯	◯

15. Please justify your answers regarding the **ease-of-use** of the tool. *

16. **System Usability Score** (SUS) *

Express your agreement with the following items.

Mark only one oval per row.

	Strongly Disagree	Disagree	Neither	Agree	Strongly Agree
I think that I would like to use this system frequently.	◯	◯	◯	◯	◯
I found the system unnecessarily complex.	◯	◯	◯	◯	◯
I thought the system was easy to use.	◯	◯	◯	◯	◯
I think that I would need the support of a technical person to be able to use this system.	◯	◯	◯	◯	◯
I found the various functions in this system were well integrated.	◯	◯	◯	◯	◯
I thought there was too much inconsistency in this system.	◯	◯	◯	◯	◯
I would imagine that most people would learn	◯	◯	◯	◯	◯

to use this system very quickly.					
I found the system very cumbersome to use.					
I felt very confident using the system.					
I needed to learn a lot of things before I could get going with this system.					

17. Please justify your answers regarding the **usability** of the tool. *

Additional Remarks

Express your opinion with these questions.
When you find it relevant, please add additional thoughts.

18. Do you consider having a feature "**undo/redo actions**" important for this type of tool? *

Mark only one oval.

- Yes
- No

19. **Why?**

20. Do you consider having a feature "**move elements**" important for this type of tool? *

Mark only one oval.

- Yes
- No

21. **Why?**

22. Do you consider having a feature "**use of keyboard keys**" important for this type of tool? *

Mark only one oval.

- Yes
- No

23. **Why?**

24. Do you believe the diagram achieved is **compliant with UML standards**? *

Mark only one oval.

- Yes
- No

25. **Why?**

26. Do you have any **comments** or **suggestions**, or would like to **suggest** any **new features**?

This content is neither created nor endorsed by Google.

Google Forms

C.2 Experimental Group (Visual Editor)

Infragenie Evaluation Form

Thank you for agreeing to participate in this experiment.

All data provided in the form is anonymous and will only be used for the scope of this experiment to help evaluate the tool developed and find ways to improve it.

The results will be published only in aggregate form and used to derive statistical data about the tool.

* Indicates required question

Introduction to the visual editor

The visual editor you will use allows you to edit a diagram of the architecture of a software system. The diagram is automatically generated from a docker-compose file in the repository analyzed by the tool.

You can move on to the next section now.

Good Luck!

Tasks to complete

This section contains several tasks for you to complete as **correctly** and **quickly** as possible.
Please read the instructions **carefully**.

We will ask you to **record the specific time** of the **start of each task** and the **end time** of the **last task**.

After completing each task, **return to this form** to continue the experiment.

1. Please **insert your assigned ID**. *

Experiment Setup

Please open the editor. To do so, enter the **repository name** and corresponding **branch** (see below) on the **form** of the tool's **homepage** and click on the "**Preview & Edit**" button.

Below, we present the relevant data to input:

Repository: mbarra29/infragenie-tests
Branch: watermelon

2. **Start of Task 1** *

Please record the time you reached this stage.

Example: 8:30 AM

3. **Task 1** *

Open the properties of the **Prometheus Node**.
How many children does this node have?

Mark only one oval.

- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5

4. **Start of Task 2** *

Please record the time you reached this stage.

Example: 8:30 AM

5. **Task 2** *

Open the **Grafana Component** and **add a new property** to it.
What is the total number of properties **after** your addition?

Mark only one oval.

- 1
- 2
- 3
- 4
- 5

6. **Start of Task 3** *

Please record the time you reached this stage.

Example: 8:30 AM

7. **Task 3** *

Delete the "**Back-tier**" **cloud/network**.
How many **connections/links** were **deleted** because of that initial deletion?

Mark only one oval.

- 1
- 2
- 3
- 4
- 5

8. **Start of Task 4** *

Please record the time you reached this stage.

Example: 8:30 AM

Task 4

Create a new element of a type of your choosing and **add a new link** connecting that **new element to the Datadog Node**.

9. **Start of Task 5** *

   Please record the time you reached this stage.

   _____

   Example: 8:30 AM

Task 5

Open **one element of your choosing** and edit **one of the editable properties** (name, children, properties, ...) to your liking.

10. **Start of Task 6** *

    Please record the time you reached this stage.

    _____

    Example: 8:30 AM

Task 6

Please submit the changes made, **ensuring the whole diagram is visible in the preview window**. In the dropdown, select the branch previously mentioned, check the “Add to Readme” checkbox, and click the “Submit Changes” button.

After clicking the button, you will be prompted to **insert your assigned ID**.

11. **End of the experimental tasks** *

    Please record when you reached the end of the tasks here.

    _____

    Example: 8:30 AM

Evaluation Questions

In this section, you will be asked to **answer several questions** that will allow us to evaluate the tool thoroughly.

Please **answer carefully and truthfully**.

Technology Acceptance Model (TAM)

Express your agreement with the following items, considering t**he job of creating and maintaining architectural documentation**.

12. Perceived Usefulness *

Mark only one oval per row.

	Strongly Disagree	Quite Disagree	Slightly Disagree	Neither	Slightly Agree	Quite Agree	Strongly Agree
Using this tool in my job would enable me to accomplish tasks more quickly.	⬭	⬭	⬭	⬭	⬭	⬭	⬭
Using this tool would improve my job performance.	⬭	⬭	⬭	⬭	⬭	⬭	⬭
Using this tool in my job would increase my productivity.	⬭	⬭	⬭	⬭	⬭	⬭	⬭
Using this tool would enhance the effectiveness on the job.	⬭	⬭	⬭	⬭	⬭	⬭	⬭
Using this tool would make it easier to do my job.	⬭	⬭	⬭	⬭	⬭	⬭	⬭
I would find this tool useful in my job.	⬭	⬭	⬭	⬭	⬭	⬭	⬭

13. Please justify your answers regarding the **usefulness** of the tool. *

14. Perceived Ease-of-use *

Mark only one oval per row.

	Strongly Disagree	Quite Disagree	Slightly Disagree	Neither	Slightly Agree	Quite Agree	Strongly Agree
Learning to operate this tool would be easy for me.	◯	◯	◯	◯	◯	◯	◯
I would find it easy to get this tool to do what I want it to do.	◯	◯	◯	◯	◯	◯	◯
My interaction with this tool would be clear and understandable.	◯	◯	◯	◯	◯	◯	◯
I would find this tool would be clear and understandable.	◯	◯	◯	◯	◯	◯	◯
It would be easy for me to become skillful at using this tool.	◯	◯	◯	◯	◯	◯	◯
I would find this tool easy to use.	◯	◯	◯	◯	◯	◯	◯

15. Please justify your answers regarding the **ease-of-use** of the tool. *

16. **System Usability Score** (SUS) *

Express your agreement with the following items.

Mark only one oval per row.

	Strongly Disagree	Disagree	Neither	Agree	Strongly Agree
I think that I would like to use this system frequently.					
I found the system unnecessarily complex.					
I thought the system was easy to use.					
I think that I would need the support of a technical person to be able to use this system.					
I found the various functions in this system were well integrated.					
I thought there was too much inconsistency in this system.					
I would imagine that most people would learn					

to use this system very quickly.					
I found the system very cumbersome to use.					
I felt very confident using the system.					
I needed to learn a lot of things before I could get going with this system.					

17. Please justify your answers regarding the **usability** of the tool. *

Additional Remarks

Express your opinion with these questions.
When you find it relevant, please add additional thoughts.

18. Do you consider having a feature "**undo/redo actions**" important for this type of tool? *

Mark only one oval.

- Yes
- No

19. **Why?**

20. Do you consider having a feature "**move elements**" important for this type of tool? *

Mark only one oval.

- Yes
- No

21. **Why?**

22. Do you consider having a feature "**use of keyboard keys**" important for this type of tool? *

Mark only one oval.

- Yes
- No

23. **Why?**

24. Do you believe the diagram achieved is **compliant with UML standards**? *

Mark only one oval.

- Yes
- No

25. **Why?**

26. Do you have any **comments** or **suggestions**, or would like to **suggest** any **new features**?

This content is neither created nor endorsed by Google.

Google Forms

Appendix D

Additional Evaluation Data

Table D.1: TAM Usefulness and Ease of use comments (Control)

ID	Perceived Usefulness Justification	Perceived Ease-of-use Justification
SB1	In many of the options I selected "Neither", because although I find this tool very useful, it seems to be restricted to some specific use cases. Thus, I cannot say it would drastically increase my productivity. Despite that, it appears to facilitate the creation of diagrams which can indeed make some parts of my job easier.	Overall, it is very simple and straightforward to use.
SB2	It is easy to use and simpler to work with than other more known drag and drop tools that quickly become visually disorganized. The drop down menus make it so that even someone like me who as never used the language can quickly understand how things can be added by exploring the available options without visiting the documentation. I feel like it would be very useful to make quick changes to an architecture diagram even after a long time without visiting a project.	It is easy to understand what types of boxes and connections can be added and how their properties can be changed. And the instant model feedback on the right quickly rearranges to make all parts visible.
SB3	The tool allows to quickly create a uml diagram without needing to drag and drop components	It is easy to edit and to add components , the preview also helps understand the diagram and makes it faster to detect any mistakes.
SB4	Having a tool that automates the process of documenting a given software architecture would decrease the time spent in that task, allowing to do more in less time. Therefore, it would be both powerful and useful.	Given that the underline language technology cannot be exchanged(in this little experiment this feature wasn't shown), even though I would find easy to operate this tool, it wouldn't allow the freedom to change the visual aspect of the diagram (e.g., font type, highlighting colors, etc). Nevertheless, I find that having standards to uniformize is important, but this feature could be implemented. It's easy to use for a software engineer, but there's a chance that the use of hashed IDs wouldn't be understood by someone not very tech savy. This backend behavior could be hidden in the frontend.
SB5	The tool can improve the worker's performance and make him avoid some mistakes. I believe that it has more effect on the time it takes to perform the job and a bit less (although some) impact on its effectiveness. Without the tool, workers perform the job correctly but it may take more time. The preview might help with identifying some mistakes and correct them in time.	The interface of the tool is clean and gives good feedback to the user, which makes it intuitive. The preview can clarify any doubt the user might have of how an action affects the diagram, which helps the user understand the tool and improve at using it.
SB6	For the job of creating and maintaining architectural documentation this tool would cut down on the time needed for the tasks of maintaining UML diagrams, and help in the consistency of the information.	The tool is very intuitive and simple to use. The interactions are simple and straight-forward.

Table D.1: TAM Usefulness and Ease of use comments (Experimental)

ID	Perceived Usefulness Justification	Perceived Ease-of-use Justification
WM1	The tool allow us to quickly set up something that is very visual. Usually this is very difficult do to the editors being to graphical. It would save a lot of time	Pretty intuitive but missing ctrl c/ctrl v. Also, it's not clear on to open the element and then jump to add new elements
WM2	I'm used to using tools like LucidChart and one of the things I think I waste most time there is searching for the components I want to work with; this tool shows all the existent components right away and it's much easier to select them. Besides that, it's easier to edit the components using this platform rather than those usual tools. The children feature is also great and very helpful.	I was able to do all the tasks from this experiment with ease and in a short amount of time, at least I believe it was short. It's true that the student gave a short explanation before the experiment but I believe the platform is simpler enough that one might have figured out everything without the demonstration. I also believe that the short explanatory text that contributes to the ease-of-use of this tool.
WM3	I don't use docker nor diagrams in my job	it's just an overall easy to use tool, it could use some improvements (eg: if I want to select an element inside another I need to do so through the menu, in my testing, double-clicking always set the selected element to the outside one) but the overall implementation is good.
WM4	I think the tool can be very useful to properly document existing open source software, which may be used in academic settings. While explaining what the software actually does could work, diagrams are more understandable because their interpretation is usually universal - and the fact that these diagrams can essentially be automated from docker-compose files is not only useful, but powerful.	I got a brief explanation before starting this form and I believe I picked up how to use it quite easily.
WM5	The tool would be good to rapidly draft the architecture of the software. It is automatic and would allow for changes in the codebase to be immediately seen in the UML.	The tool is simple. That allows for a user to quickly learn in and develop their tasks. On the other hand, it might be that not all of the functionalities of other UML tools is available which might make some tasks not feasible. Additionally, the creation and maintaining of links didn't seem the most intuitive, since selection of sources and targets can only be done through a drop down and not in a visual way
WM6	The tool is useful in the sense that it avoids worrying about the location and layout of the diagram elements and maintains uniformity in documentation, allowing more diagrams to be made in less time. However, I found the editing of some diagram elements to adjust the layout a bit difficult.	The way to manipulate the elements is simple and relatively intuitive. Because most of the interactions are based on clicks and lists of values, it becomes easy to understand how to use the tool and use it to build diagrams easily, but with the growth of the diagrams, some currently used mechanisms might not be that scalable, particularly the use of the dropdown of all elements to choose a link source or target.

Table D.2: SUS Usability comments

ID	Usability Justification
SB1	The usability is very good for a starting point, I would only add some minor improvements that would increase the engagement in the tool. For instances, associating each component with icons would increase the diagram edition speed, because one would not spend time searching in the dropdown menu for a given option. In addition to that, I would like to see more warnings (popups) just as a confirmation of what is being done to the diagram (e.g. Port added).
SB2	I think anyone, regardless of having the specific background in program architecture or UML could construct a model if given a bit of detail on what elements to add, given that the syntax seems straight forward and the dropdown menus make it easy to add. change or delete elements without much thought.
SB3	I found it quite intuitive to add/edit/delete components, the only downside is copying and pasting the ids
SB4	The tool was very usable, the hash IDs could be occluded in the frontend. As this product is directed to people who use Docker on the daily, there's only a need to know that technology well. The connectors could be improved, perhaps they should be added at the top of the editor instead. Or an option to add a connection to another component should be added on the selection menu of a component.
SB5	The system is easy to understand after a quick demonstration (although the demo is helpful). The features are naturally integrated with one another. In my opinion, to complete the sense of integration and give one final boost to the (already good) flow of the system I would include an option to create connections clicking on the elements (in addition to copying ids)
SB6	The system is very straight-forward and doesn't take a lot of time to get used to, especially if having a background on IT. It is minimalistic while still providing the needed functionality.
WM1	Easy to learn and discover stuff by myself
WM2	The tool is very easy to use and I believe that due to that simplicity it's a worthy substitute for other diagram making tools that most times are way too complicated for what one wants to do with them.
WM3	I believe some people will have issues with the tool, but in my opinion the tool does what it is intended to do with little to none 'bloat'
WM4	It's hard to justify - but I felt like controls were mostly intuitive after the brief explanation. Everything seemed to work exactly how I expected - sort of like it is said, when learning how to code with Python, that it is a great language to learn because "you think about something you should be able to do, and it simply works".
WM5	The system is simple and quite straight forward. It it easy to get to know all of the functionalities and start to work right away. It might be a little too complex in the way that links work since their implementation seems to differ from most UML alternatives and require you to select from a dropdown.
WM6	The tool is quite intuitive and not overwhelming. It has the necessary elements well organized on the page, and the display of the diagram and the information panel are well organized.

Table D.3: Comments on Feature Importance and UML compliance

ID	undo_redo	move_elements	keyboard_keys	uml_compliant
SB1	**Yes.** It is very important to don't lose time trying to mimic that feature ourselves.	**Yes.** This tool reminds me of draw.io and I would enjoy seeing a similar experience to that tool in this one. I consider a lot more engaging moving and "feeling" the elements in first hand.	**No.** I don't see people using the keyboard that much, it should support it anyway because of the accessibility	**Yes.** Although I am not sure, it seems that it follows all the UML standards.
SB2	**Yes.** Many times you delete or change things that you didn't mean to and need to roll back changes so it is useful to have some kind of CTRL-Z.	**Yes.** Moving elements that may be related together if they were initially written in sections apart is useful for having them both in view if I am trying to make some decisions on what information should be in each one.	**Yes.** As with every digital tool, the use of keyboard shortcuts makes for increased performance when we become accustomed to the program, so I say it is important for using it daily.	**Yes.** I am not sure of what are the correct UML standards but from what I know it looks correct and is certainly completely understandable.
SB3	**Yes.** It is easier to reset to previous state	**Yes.** It helps if I want to change positions in the diagram so I can organize it as I see fit	**No.**	**Yes.**
SB4	**Yes.** A delete action that wasn't planned, or one that was planned but after some changes there's a sudden realization that it made sense in that system. I think the feature of middle crossing the deleted element is welcome and well realized.	**Yes.** For people with ADHD that's a really important feature, I find it impossible to not group certain components together, for instance, networks, or change their order. Nevertheless, it's important to give liberty to the user, even though the majority of users might not even utilize it, others might find it really important.	**Yes.** For a tool whose purpose is efficiency, to maximize it some shortcuts would be really useful.	**Yes.** It's written in a "well documented" language (plantuml). The responsibility falls on the developers of the language.

Table D.3: Comments on Feature Importance and UML compliance (cont. 1

ID	undo_redo	move_elements	keyboard_keys	uml_compliant
SB5	**Yes.** As in many scenarios, you can perform an action by mistake that is hard to reverse	**Yes.** It allows people to organize the elements a little bit better and more to their liking, which could be a factor in improving performance even more, especially when there are a great number of elements	**Yes.** I think the system works best with pointer control, but more accessibility is always important and I imagine that some keyboard shortcuts can also improve performance for users with more skill operating the system (as we see in other systems like the text editors)	**Yes.** It seems that way, but I would need to review the standards to give a more informed answer
SB6	**Yes.** Although you can always revert it to the original state, it would be useful to be able to undo or redo certain actions that were not yet committed.	**No.** Since the elements are automatically rendered on the diagram, their order should not influence the final diagram.	**No.** Although it could be useful for short-cutting some actions, the mouse fits well for this very visual tool.	**Yes.** Since it's being generated automatically with plantUML, I would assume it is compliant.
WM1	**Yes.** As it is so common, it's comfortable to have. I would pay a premium version for it	**Yes.** Pretty useful when refactoring architecture	**Yes.** Better for small adjustments	**Yes.** Well readjusting even when trying to move boldly and trying do reformat
WM2	**Yes.** I didn't have the misfortune of making a mistake during this experiment but I can see how it could happen easily. And even though, it didn't happen now, I can see it happen to me in the future. Not even just for errors but even if one just thinks better of something and wants to change what they have done, it would be very useful.	**Yes.** This feature is important both for design purposes (a person might want the diagram to look a certain way: certain components next to each other) and for comprehensibility/visibility purposes (a component might be on top of others or the text in a link might not be readable due to components being on top of it).	**Yes.** I don't use keyboard that much, I prefer to use the mouse but sometimes you just want to move components a little little bit which is impossible with the mouse so for that I think it's useful.	**Yes.** It has all necessary components available and allow to the necessary connections so yes.

Table D.3: Comments on Feature Importance and UML compliance (cont. 2)

ID	undo_redo	move_elements	keyboard_keys	uml_compliant
WM3	**Yes.** People make mistakes, if you delete an element and want to revert that you'll need to recreate that entire element, it's children and connections.	**Yes.** It would help better visualizing the actual architecture.	**Yes.** Some keyboard key interactions would be very helpful, eg. if you need to duplicate an element you'll need to recreate it and all it's children, being able to do that with Ctrl+C & Ctrl+V would drastically improve performance	**Yes.** I'm not 100% on the UML standards of others, but the overall diagram was readable to me
WM4	**Yes.** It's really just a usability feature! Even if just in the scenario of a user experiencing a bug (not my case) and being able to revert whatever the bug caused could be helpful for users. In general, it is one of those "shortcut" actions most people know, so it would also give points for the intuitiveness of the tool!	**Yes.** I, for one, messed around with the positions to make the text as readable as possible. The appearance / visual side of the diagrams is very important for their comprehension.	**Yes.** Intuitiveness. It's almost expected - "delete" removes stuff, "Ctrl + C" copies, et cetera.	**Yes.** In my opinion, it simply is. I could clearly understand the diagram just by "reading" it.
WM5	**Yes.** "Oh no! I hit delete. My 20-properties component is gone!" Besides being standard practice, undo and redo actions help the user allowing them to experiment with their work and allow them to undo mistakes	**Yes.** For more complex architectures, the user might want to move elements not only so there is fewer overlap of links (making the readability of the UML simpler), but also to show correlation of objects more or less based on their location.	**Yes.** While not absolutely necessary, it increases usability and productivity for more experienced users. They can execute simpler tasks in less time. It also provides alternatives for machines without a mouse.	**No.** I found lacking support for some of the functionalities of UML such as composition, generalization and multiplicity. Please disregard this opinion if such properties and characteristics are not supposed to exist in architecture UMLs for which the software appears to be tailored to.

Table D.3: Comments on Feature Importance and UML compliance (cont. 3)

ID	undo_redo	move_elements	keyboard_keys	uml_compliant
WM6	**Yes.** Because one of the most frequent things that happen are missclicks, and it is important to think that the user is dumb and will always make a mess, and that way, we can diminish the possible rage a delete would cause.	**Yes.** To readjust the layout automatically generated, and to avoid elements getting on top of each other and improve the visual semantics of the diagram.	**No.**	**Yes.**

Table D.4: Additional Comments and Suggestions

ID	Comments & Suggestions
SB1	I would like to see this extended to other type of files, for example github actions, pipelines, etc. This would highly increase this tool use cases.
SB2	
SB3	
SB4	The hash IDs could be occluded in the frontend. Give a tl;dr to explain the type of connections. Private repositories. Specify the folder of docker-compose files. Is it possible to use more than one docker-compose file, is there folder recursion?
SB5	A feature that I think could be interesting is a way to see a list of the element's connections maybe in its dropdown menu options
SB6	It would be nice to be able to click on a property on the rendered diagram to focus that specific property on the left.
WM1	Copy and past would be very helpful
WM2	Just for design purposes might be nice in the future to allow users to pick the colors of components and text? Also I believe the redo feature would be very useful. Besides that I would like to congratulate the maker of the tool, it's awesome and it looks great. Great job :))
WM3	More customization, this tool is going to be used for a single purpose, the best way to ensure that it fits that job is by allowing the user to change some app parameters. Additionally, a dark-theme would be nice.
WM4	Suggestions: - Undo feature (with button + keyboard shortcut Ctrl + Z) - An easier way to make the links. Maybe, when creating a link, the sidebar immediately opens with the option to select the source and target. The user could then just click the elements they want as each one to associate the link to the elements. Comments: I found the tool very pleasing to use on the visual side and very intuitive. I would definitely like to have had something like this over alternatives that are either limited because they require payment or existing free alternatives that seem a bit too disconnected from UML or are just a barely optimized or bettered "Insert Figure" function (like in Microsoft Office software).
WM5	Maybe consider a fullscreen mode where the diagram editor is not encapsulated within a page but is the page itself. At least for a the later phases of the software and not for the experiment. The possibility do perform this type of analysis (of a docker file) to fully fledged repositories and object classes would be absolutely extraordinary. Encountered a bug. Didn't try to replicate but pretty sure it was when trying to delete a connection that didn't have neither a target nor a source.
WM6	Suggestions: - add a link should add it right away to the screen, even if it had no connections, to make it easier to find and edit its properties (maybe in a section where all the initially created elements would go and not yet altered to make it easier to find and separate from the other ones) - the delete of an element shouldn't delete the links associated with it, only the element itself, or else it will be necessary to create all the connections again - the dropdown boxes for the source and target choice should be a dropdown with search to help identify the wanted element on the list - in editing the properties of a link, it could show close to the source and the target what side corresponds to each tip, mainly in the cases where the links are not equal on the 2 sides - the total number of links and elements listed could be beside the title of those sections - add filters in the element/link lists by type to help find a specific element/link Bugs found during the experiment: - after selecting the type of link to add; in the alert, the "cancel" button doesn't seem to be working - if an element in the visual editor is selected, when clicking a link in the list of links to edit its properties, it looks like it doesn't change the window.