

Master Thesis

Software Improvement Process

at

CERN CO-AP

by

Pedro Nuno Pinto
2008, Genève

Supervisor:
Jaime Villate
FEUP

Faculdade de Engenharia
da Universidade do Porto



FEUP



Your time is limited, so do not waste it living someone else's life.
do not be trapped by dogma - which is living with the results of
other people's thinking. do not let the noise of other's opinions
drown out your own inner voice. And most important, have the
courage to follow your heart and intuition. They somehow already
know what you truly want to become. Everything else is secondary.

Aaron Burr (1756 - 1836)

Abstract (EN)

The CERN CO-AP section is responsible for developing and maintaining software for the accelerators control system. AP has been doing this for a number of years with good results. However, the Software Improvement Process (SIP) initiative was created in order to evaluate the way in which AP develops software. This auditing process had clear objectives, which consisted, essentially, in the identification and improvement of the software development process, and practices, of the several projects from AP section. However, it has been demonstrated that AP section does not have a consistent and established software development process.

This dissertation assumes that the objective is to propose and introduce a common software development process, and to identify a set of practices, ideas and tools that could be used in order to improve the collaboration between the teams of developers, and the productivity and quality of the code developed. In order to do this, an analysis initiative of the software engineering activity has been promoted into the context of the software development processes.

The issue of the software crisis remains as well. We still do not know how to develop perfect software, capable of meeting the requirements, with quality and within the deadline. These and other issues exist since the origins of the software development. Along the years, several development models have appear in order to solve this. Multiple approaches have been proposed. This dissertation presents an overview on this topic, from the so called *heavyweight processes*, until the *lightweight* or *Agile* ones. Throughout this overview, some solutions, suitable enough to solve the aforementioned AP problem, are proposed. This proposal focuses on CMMI, with the Scrum development model and XP practices in the spotlight, in what will be called, the Agile CMMI.

Abstract (PT)

A secção CERN-CO-AP é responsável por desenvolver e manter software para o sistema de controlo dos aceleradores de partículas. A AP tem feito esta actividade há já alguns anos, e com bons resultados. Assim, de modo a avaliar a forma como a AP desenvolve software, foi criada a iniciativa Software Improvement Process (SIP). Trata-se de um processo de auditoria que tem um claro objectivo, e que consiste, essencialmente, em identificar e melhorar o processo de desenvolvimento de software, e práticas usadas nos diversos projectos em curso na secção. Contudo, foi demonstrado que a AP não possui um consistente e estabelecido processo de desenvolvimento de software.

Esta dissertação assume o objectivo de propor e introduzir um processo de desenvolvimento de software comum, identificando um conjunto de práticas, ideias e ferramentas capazes de melhorar a colaboração entre as equipas de desenvolvimento, a sua produtividade e a qualidade do código desenvolvido. De modo a alcançar este objectivo, é desenvolvida uma análise da actividade que é a engenharia de software, e em particular do contexto que é os processos de desenvolvimento de software.

A eterna questão da crise no desenvolvimento de software ainda prevalece. De facto, ainda não sabemos como desenvolver software perfeito, capaz de cumprir os requisitos, com qualidade e dentro dos prazos. Estas e outras questões prevalecem desde as origens da engenharia de software. Ao longo dos tempos, diversos modelos de desenvolvimento surgiram no sentido de solucionar esta problemática, tendo sido propostas diversas abordagens. Esta dissertação, desenvolve um percurso que se inicia com a análise dos chamados heavyweight processes, até aos lightweight ou Agile. Assim, o objectivo passa por propôr um conjunto de soluções capazes de resolver a referida problemática na AP. Assim, esta proposta irá focar-se no modelo de CMMI, com o modelo de desenvolvimento Scrum e práticas XP, no que será denominado de Agile CMMI.

To the dedication, professionalism and enthusiasm

of the CERN CO-AP people

Contents

Contents	e
List of Figures	1
List of Tables	2
I Context	3
1 Introduction	5
1.1 CERN	5
1.2 Accelerators Complex	6
1.3 Accelerators Controls	7
1.4 The Applications Section (AP)	9
2 Thesis Proposal	12
2.1 Scenario	12
2.1.1 Purpose	13
2.1.2 Objectives	13
2.2 Motivation	14
2.3 Organization	15
II Analysis	16
3 Software Engineering	18
3.1 Introduction	18
3.1.1 Software Crisis	19
3.1.2 Silver Bullet	19
3.1.2.1 Why Developing Software is Difficult	20
3.2 Software Development Process (SDP)	23
3.2.1 Definition	23

3.2.2	The Need for a Defined SDP	23
3.2.3	The Need for a Standard SDP	24
4	Software Development Models	25
4.1	Introduction	25
4.2	Waterfall	26
4.2.1	Advantages vs Disadvantages	28
4.2.2	Application	29
4.3	Incremental	29
4.3.1	Advantages vs Disadvantages	30
4.3.2	Application	31
4.4	Evolutionary	31
4.4.1	Advantages vs Disadvantages	32
4.4.2	Application	33
4.5	Spiral	33
4.5.1	Advantages vs Disadvantages	35
4.5.2	Application	35
4.6	Iterative and Incremental Development (IID)	35
4.6.1	Brief History	37
4.7	Agile Software Development	39
4.7.1	Manifesto for Agile Software Development	40
4.7.1.1	Principles Behind	41
4.7.2	The People Factor	41
4.8	CMMI	42
4.8.1	Description	42
4.8.2	Process Area Categories	46
4.8.3	Maturity Levels	47
4.8.4	Goals and Practices	48
5	Agile Development Models	50
5.1	Scrum	50
5.1.1	Basics	51
5.1.2	Roles	52
5.1.2.1	Product Owner	52
5.1.2.2	ScrumMaster	52
5.1.3	Practices	53
5.1.3.1	Product Backlog	53
5.1.3.2	Sprint Planning Meeting	53
5.1.3.3	The Sprint Review Meeting	54
5.1.3.4	The Daily Scrum Meeting	55

5.1.4	Advantages vs Disadvantages	56
5.2	Extreme Programming (XP)	57
5.2.1	Roles	58
5.2.2	The Twelve Practices	58
5.2.2.1	Small Releases	59
5.2.2.2	The Planning Game	60
5.2.2.3	Refactoring	60
5.2.2.4	Testing	61
5.2.2.5	Pair-Programming	61
5.2.2.6	Sustainable Pace	62
5.2.2.7	Team Code Ownership	62
5.2.2.8	Coding Standards	63
5.2.2.9	Simple Design	63
5.2.2.10	Metaphor	63
5.2.2.11	Continuous Integration	64
5.2.2.12	On-Site Customer	64
5.2.3	Values and Principles	64
5.2.4	Advantages vs Disadvantages	65
5.2.5	Application	66
5.3	Feature Driven Development (FDD)	66
5.3.1	Roles	67
5.3.2	Activities	67
5.3.2.1	Develop an Overall Model	67
5.3.2.2	Build a Features List	68
5.3.2.3	Plan by Feature	68
5.3.2.4	Design by Feature	69
5.3.2.5	Build By Feature	69
5.3.3	Practices	69
5.3.4	Advantages vs Disadvantages	70
5.3.5	Application	71
5.4	Some Statistics	71
5.5	Some Considerations	72
6	The AP from the Scratch	74
6.1	The Section	74
6.1.1	Organization	74
6.1.2	Staff vs Temporary Personnel	75
6.1.3	Qualification	75
6.1.4	Working Conditions	76
6.2	Control System	77

6.3	Software Development Process	78
6.3.1	Roles	79
6.3.1.1	Client	79
6.3.1.2	Project Leader	79
6.3.1.3	Section Leader	80
6.3.2	Practices and Considerations	80
6.3.2.1	Domain Analysis	80
6.3.2.2	Incremental and Iterative	80
6.3.2.3	Requirements Analysis and Specification	81
6.3.2.4	Small Releases	81
6.3.2.5	Continuous Integration	81
6.3.2.6	On-Site Client	81
6.3.2.7	Individual Code Ownership	82
6.3.2.8	Documentation	82
6.3.3	Tools	83
6.3.3.1	JIRA	83
6.3.3.2	Wikis	83
6.3.3.3	Technical Meetings (TMs)	84
6.3.3.4	Software Metrics	84
6.4	Evaluation with CMMI	85
III	Proposal	86
7	Looking Forward	88
7.1	Agile CMMI	89
7.1.1	Why Adopting CMMI?	89
7.1.2	CMMI to improve Agile	90
7.2	Why Scrum with XP?	94
7.2.1	Complementary Practices	94
7.2.2	Scrum Roles and Practices	95
7.2.2.1	ScrumMaster	95
7.2.2.2	Product Owner	96
7.2.2.3	Sprints and Sprint Planning Meetings	96
7.2.2.4	Daily Scrums	97
7.2.2.5	Sprint Review	97
7.2.2.6	Sprint Retrospective	98
7.2.2.7	Keep the Team Together and Isolated	98
7.2.3	XP Practices	98
7.2.3.1	Code Standard	98
7.2.3.2	TDD	99

7.2.3.3	On-Site Client	99
7.2.3.4	Pair Programming	100
7.2.3.5	Other Practices	101
7.2.4	Some Suggestions	101
7.3	Documentation	101
7.3.1	Structural Documentation	101
7.3.2	Code	102
7.4	Tools	102
7.4.1	Training Course	102
7.4.2	Open Day	103
7.4.3	Software Metrics	103
7.4.3.1	Wikis	104
7.4.4	ProKey	104
7.4.5	Additional Notes	105
7.5	Control System Perspectives	105
 IV Conclusions		106
 8 Critic		108
8.1	Considerations	108
8.1.1	Resistance to Change	109
8.2	Improvements	110
8.3	Acknowledgments	110
 ProKey		111
 Nomenclature		114
 Bibliography		116

List of Figures

1.1	CERN Accelerators Complex [Vanoli(2006)]	7
1.2	CERN Control Centre [Brice(2006)]	9
3.1	Software Engineering	22
4.1	Waterfall model	27
4.2	Incremental model	30
4.3	Spiral model	34
4.4	Waterfall Development	36
4.5	Iterative Development	37
5.1	Scrum process	51
6.1	AP lines of code	78
6.2	JIRA at CO-AP	83
8.1	ProKey - Products to Keywords I	112
8.2	ProKey - Products to Keywords II	112
8.3	ProKey - Keywords to Products I	113
8.4	ProKey - Keywords to Products II	113

List of Tables

5.2	Which practices are more used within a Agile process	72
5.3	Agile tools currently used or planned to be used in the next 6-12 months	72

Part I

Context

The skill of writing is to create a context

in which other people can think.

Edwin Schlossberg (1945 -)

Chapter 1

Introduction

1.1 CERN

The European Organization for Nuclear Research, commonly known as CERN¹ is the largest Particle Physics Laboratory in the World. It sits astride the Franco-Swiss border near Geneva. The name CERN is derived from the French Conseil Européen pour la Recherche Nucléaire, or European Council for Nuclear Research, a provisional body founded in 1952 with the mandate of establishing a world-class fundamental physics research organization in Europe. When the organization officially came into being in 1954, the Council was dissolved, and the new organization was given the title European Organization for Nuclear Research, although the name CERN was retained.

CERN was one of the Europe's first joint ventures and includes now 20 member states and scientists from several nationalities and cultures which unit efforts to study the building blocks of matter and the forces that hold them together. CERN was created to provide them with the necessary tools. These tools are called accelerators, which are huge machines with the purpose of accelerate particles to almost the speed of light, and detectors to make the particles visible.

The accelerators complex aforementioned is built around three principal, inter-dependent accelerators. The oldest one, the Proton Synchrotron (PS), was built in the 1950s and was briefly the world's highest energy accelerator. The Super Proton Synchrotron (SPS), built in the 1970s, was the scene of CERN first Nobel Prize in the 1980s. The Large Electron-Positron (LEP) collider came on stream in 1989 and it was the laboratory's flagship research machine until 2000.

¹French: Organization européenne pour la recherche nucléaire

Currently under construction, and inside the same tunnel where the LEP was, is the **L**arge **H**adron **C**ollider (LHC), which is scheduled to begin operation in June 2008. The LHC is expected to become the world's largest particle accelerator, with the highest energy, as well as the biggest machine ever built by men.

Our current understanding of the Universe is incomplete. The *Standard Model*² of particles and forces summarizes our present knowledge of particles physics. This model has been tested, by various experiments, and it has proven particularly successful in anticipating the existence of previously undiscovered particles. However, it still leaves many unsolved question, which the LHC will help to answer. For instance, the *Standard Model* does not explain the origin of mass, or why some particles are very heavy while others have no mass at all. The answer may be the so-called Higgs mechanism. According to the theory of the Higgs mechanism, the whole of space is filled with a "*Higgs field*", and by interacting with this field, particles acquire their masses. So, particles that interact intensely with the Higgs field are heavy, while those that have feeble interactions are light. The Higgs³ field has at least one new particle associated with it, the Higgs boson. If such a particle exists, experiments at the LHC will be able to detect it.

1.2 Accelerators Complex

The accelerator complex at CERN is a succession of machines with increasingly higher energies. Each machine injects bunches of particles, designated as beam⁴, into the next one, which takes over to bring the beam to an even higher energy, and so on. In the LHC - the last element of the chain - each particle beam is accelerated up to the record energy of 7 TeV. In addition, most of the other accelerators in the chain have their own experimental halls, where the beams are used for experiments at lower energies. As it was mentioned above, LHC stands for **L**arge **H**adron **C**ollider. **L**arge due its size (approximately 27 km in circumference), **H**adron because it accelerates protons or ions, which are hadrons, and **C**ollider because these particles form two beams traveling in opposite directions, which collide at four points around the machine circumference.

There are four major experiments at the LHC: **A** **L**arge **I**on **C**ollider **E**xperiment

²the *Standard Model* is a collection of theories that embodies all of our current understanding of fundamental particles and forces

³the name comes from Professor Peter Ware Higgs who proposed the Higgs mechanism and the Higgs boson

⁴each beam consists of about 3000 bunches of protons, each bunch containing up to 100 billion protons

(ALICE), the **A** Toroidal **LHC** Apparatu**S** (ATLAS), the **C**ompact **M**uon **S**olenoid (CMS), the **L**arge **H**adron **C**ollider beauty (LHCb). All of them are installed in huge underground caverns built around the four collision points of the LHC beams. The purpose of these experiments is distinct, however all of them are designed to observe phenomena that involves highly massive particles, which were not observable using earlier lower-energy accelerators. This might shed light on new theories of particle physics beyond the *Standard Model*. So, the purpose of all of this is that, by accelerating and smashing particles, physicists can identify known components, or new ones, revealing the nature of the interactions between them⁵.

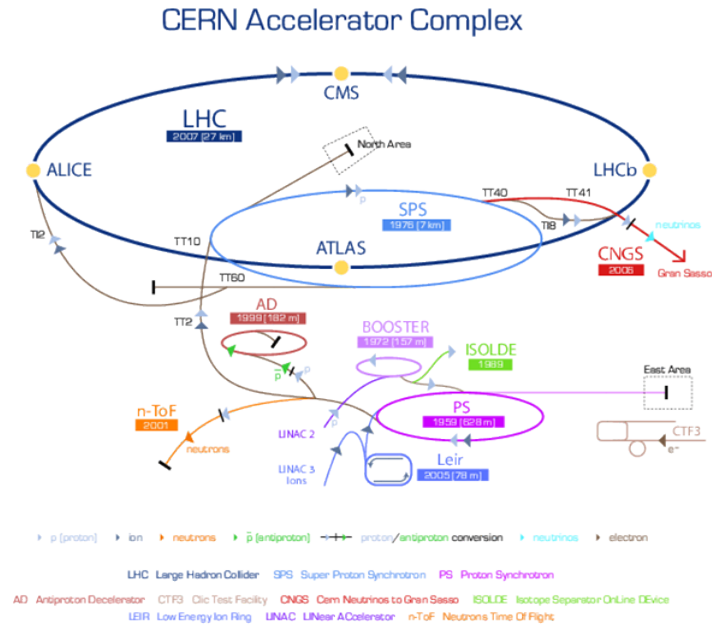


Figure 1.1: CERN Accelerators Complex [Vanoli(2006)]

1.3 Accelerators Controls

The LHC scale and complexity are unprecedented in the field of particle accelerators. It has the largest number of components and the widest diversity of systems of any accelerator in the world. As many as 500 objects, around the 27 km circumference, from passive valves to complex experimental detectors, could,

⁵[CERN(2007)]

in principle, move into the beam path in either the LHC ring or the transfer lines between the LHC and the SPS. The operation of this huge structure will be extremely complicated for a considerable number of reasons, including: critical technical sub-systems, a large parameter space, real-time feedback loops and the need for on-line magnetic and beam measurements. In addition, the LHC is the first superconducting accelerator built at CERN, which brings a complete set of new challenges.

This amount of complexity means that repairs of any damaged equipment will take a long time. As an example, it will take about 30 days to change a superconducting magnet, because this will demand the warm-up of the magnet, which is expected to be near the absolute zero⁶. Then there is the question of damage if the systems go wrong. The energy stored in the beams and magnets is more than twice the levels of other comparable machines. Taking into account all of this, as well as the time (more than 10 years), investment of the project (more than 4 billion euros), and the security issues involved in the operation, means that the LHC must be protected at all costs. If an incident occurs during operation, it is critical the determination of what has happened, and the tracing of the cause. Moreover, operation should not resume if the machine is not back in a good working state.

The **A**ccelerators and **B**eams - **C**ontrols group (AB-CO) at CERN has spent the last four years developing a new software and hardware control system architecture, based on the many years of experience controlling the particle injector chain⁷. The resulting LHC controls infrastructure is based on a classic three-tier architecture: a basic resource tier that gathers all of the controls equipment, located close to the accelerators; a middle tier of servers; and a top tier that interfaces with the **C**ERN **C**ontrol **C**entre (CCC) operators. This controls infrastructure is crucial to ensure the correct operation of this complex machine, and it is estimated that involves collective effort amounts to some 300 person-years and a cost of 13.5 million euros⁸.

The CO group is constituted by a number of sections, which are sub-divisions with specific roles in the context of the group. One of the nuclear sections of those ones is the the **A**pplications (AP) section.

⁶absolute zero is the lowest possible temperature where nothing could be colder, and no heat energy remains in a substance. By international agreement, absolute zero is defined as precisely 0 K on the Kelvin scale, which is a thermodynamic (absolute) temperature scale, and -273.15 on the Celsius (centigrade) scale.

⁷involves the PSB, PS, SPS and the transfer lines (TI2 and TI8) to inject the beam into the LHC ring

⁸[[Charrue\(2008\)](#)]



Figure 1.2: CERN Control Centre [Brice(2006)]

1.4 The Applications Section (AP)

The Applications section is a group of more than 20 developers, between staff and temporary personnel, that develops and maintains a set of products and services for the accelerators controls group (CO). Those services can be split into the following topics:

- Supporting the current accelerators control system
- Providing software for the control of the PS, SPS and LHC in collaboration with the **O**peration section (OP)
- Supplying the software & services competence to the CO, OP⁹ and equipment groups

AP has grown considerably in the past few years, following the path of the LHC increasing demands. This means that new projects have been born, old ones, already used in the LHC chain injector, have been upgraded and adapted to the new challenges, and this put an inevitable amount of responsibility in the software developed here. On the other hand, the amount of work has increase as well, with thinner schedules and quality demands.

⁹the OP section is responsible for the accelerators complex control, and it is based at the CCC

As it was aforementioned, AP has several projects/products under continuous development. Next, it will be presented a list of some of them followed by a brief description.

- **LHC Software Application (LSA)** - is a system that covers all the most important aspects of the LHC controls: optics (twiss, machine layout), parameter space, settings generation and management (generation of functions based on optics, functions and scalar values for all parameters), trim (coherent modifications of settings, translation from physics to hardware parameters), operational exploitation, hardware exploitation (equipment control, measurements) and beam-based measurements.
- **LHCAlarm SERVICE (LASER)** - is an alarm service for the operation of all of the CERN accelerator chain and technical infrastructure that provides the collection, analysis, distribution, definition and archiving of information about abnormal situations. It is responsible for processing about 180 000 alarms events each day and currently has more than 120 000 definitions.
- **Software Interlock System (SIS)** - is a protection system meant to replace the old one called SSIS. It protects the SPS and LHC machines by surveilling and analyzing the state of several key equipments and cutting the beam if a potentially dangerous situation occurs.
- **Fixed Displays Framework (FDF)** - is a framework that allows to create displays applications capable of monitoring, in real time, relevant information retrieved from the accelerators complex¹⁰.
- ...

This diversity involves the commitment of several different teams dedicated to each one of these projects. In some cases, it happens that the same developer is committed to more than one project; however, in general, this does not happen. The size of a team depends a lot on the complexity and demands of each project; however, it goes from 2 until 5/6 developers (with both staff and temporary personnel). Each project has a leader responsible for reporting the state of the project to the AP leader. These projects have a continuous development and improvement, and some of the them have more than 10 years of life cycle. AP also guarantees the maintainers of the released products, and even gives training to the his major clients, the OP members.

It is also important to mention that AP is a Java oriented, in the sense that it is the official programming language used in all the products that come to

¹⁰[Pinto(2007)]

light here. This decision was done a number of years ago, around the year 2002, based in constrains of the business domain aforementioned, but also because:

Java is simple, object-oriented, distributed, interpreted, robust,
secure, architecture neutral, portable, multithreaded and dynamic¹¹

This is complemented by the fact that every day new Java oriented libraries appear, which is a considerable assert as well. Of course the section also uses other support technologies, for instance, the Sprint Framework¹², and all the development is done using the Eclipse Integrated Development Environment (IDE)¹³.

It is also important to mention that the section has weekly meetings, called technical meetings (TM) where the developers are encouraged to expose the technical problems, that they are facing in their projects, as well as presenting solutions, or new technologies that might be useful for other developers. Once a month, there are also section meetings (SM) where the evolution of the work done is presented for every project of the section, by the correspondent project leader. Still, these and many other topics will be explored deeply in the dedicated chapter **The AP from the Scratch**¹⁴.

¹¹[[Microsystems\(\)](#)]

¹²Spring Framework <http://www.springframework.org/>

¹³Eclipse IDE <http://www.eclipse.org/>

¹⁴ 6 on page 74

Chapter 2

Thesis Proposal

2.1 Scenario

As it was mentioned in the previous chapter, the AP has several projects with different teams allocated to them. This structure has been the same for a number of years with good results, that were actually noted several times by the CO, as well as by higher elements in the **A**ccelerators and **B**eams (AB) department hierarchy.

However, and being aware of their respectable position, the AP members started in the beginning of October 2007, an introspection initiative of their own processes and ways of working. This was promoted in order to identify problematic situations and fields where it could be possible to improve and, at the end, provide a even better service. This auditing process was dubbed Software Improvement Process (SIP).

A mandate creation, with the main guidelines of the initiative, was the initial SIP step. In order to do this, a number of problematic situations considered relevant for this context were identified. One of them, which was immediately recognized and also constitutes a good example, was the lack of communication among developers of different projects in the AP. In truth, most developers do not have a feeling about the context of the projects in which they are not involved. This happens in spite of the SM monthly meetings, and the TM weekly meetings. On the other hand, it is understandable that projects with different scopes, in the context of controls software, need full-time dedicated developers. Nevertheless, it is a fact that this context of low connection among projects and, consequently, among developers (sometimes even inside the same project) brings to light a problem. A poor share of knowledge like this one creates a set of situations such as: replication of ideas, architectures, designs, ... , algorithms. In ultimate case, this represents unnecessary amounts of repeated lines of code.

Nevertheless, this practical example hides a deeper problem related with the software development process itself. In fact, the AP does not have a homogeneous process of software development, which results in a diversity of dubious processes and practices amount the projects of the section. This major conclusion was the result of a preliminary analysis period that allowed to extract and clarify a set of purposes and objectives to regulate the SIP initiative. It is also important to mention that the commitment to this was assumed by the AP as a whole. This is the only way to achieve real results.

2.1.1 Purpose

Following up to was written above, the purposes of SIP can be summarized by the following major topics:

- Find a common, useful and complete software development process for the products delivered within the AP
- Avoid duplication of software components created and used by 3rd parties, which provide services needed in many projects
- Promote and develop frameworks, libraries and components on which the software is built, rather than individual one-time only solutions
- Provide better and more comprehensive documentation of the process and components (software capital of AP)
- Manage and limit the number of 3rd party products used

2.1.2 Objectives

Concerning the purpose of the SIP, there are also a set of objectives that the initiative identified and hopes to reach. However, it is certain that the analysis of all those objectives would require several dissertations. In fact, the purpose of SIP is so wide and involves so many subjects that it was not reasonable to focus all of them in this dissertation. So at this stage it was imperative to make choices. This master thesis dissertation will be focused on the following SIP major objectives:

- Identify the software development process and practices used by different teams and find a common denominator
- Propose and introduce a common software development process (adaptable depending on project needs)

- Identify and introduce practices, ideas and tools that could be used to improve the collaboration among the teams of developers, and the productivity and quality of the code developed

The objectives presented before translate the most pressing necessities of the AP. The identification and proposal of a common, useful and complete software development process, capable of being adapted to the specific needs of each project, but still following common principles, approaches and practices is critical for the AP. It is evident that these objectives will require a profound analysis of considerable amounts of subjects in a wide range of domains. This master thesis dissertation will assume the goal of promoting this analysis and discussion in order to reach the aforementioned objectives.

2.2 Motivation

In the early days of computing, software was developed by many individuals following their own processes or methods. Often, the methods employed some form of “*code and fix*”, where the programmer writes some code and then tests it to see how it performs. The programmer then uses the test results to modify or fix the code and tests again. Programmers were able to get by with this type of development for two reasons. First, no better way had been developed, and second, software was not that complex.

As software grew more complicated and organizations relied on computers for more of their operations, including finances, operations control, and even human lives, this laissez-faire approach to programming gave way to more disciplined methods. The overall framework in which software is conceived, developed, and maintained is known as the ***Software Development Process***.

The AP has clear demands in terms of producing valuable control software, and has been doing a good job so far. However, there are improvements and even profound structural modifications that can be done in this field in order to achieve excellency. This dissertation will be an attempt to analyze the AP from scratch in the context of software development processes. This will involve, apart from many other things, the identification and analysis of relevant software development processes streams, along with their context, advantages and disadvantages, suitable enough to be implemented or to serve as inspiration for the AP. On the other hand, this evaluation will have always in mind the business domain and the constraints associated to AP’s activity at CERN, in order to evaluate the viability of the discussions that will be brought to this initiative.

2.3 Organization

This master thesis dissertation is divided into four parts, each one of them with a variable number of chapters.

The *Part I Context* intends to put CERN and the CO-AP into context, and to explain the objectives and motivation for this master thesis dissertation.

The *Part II Analysis* starts by an overview of software engineering, its evolution, difficulties and major achievements. After that, a number of software development processes are described, starting by the so called “*heavyweight*” processes, moving into the “*lightweight*” ones, commonly known as “*Agile*”. This analysis intends to show how software development has progressed during its short history, exposing the major characteristics of each of the processes or models presented, with particular emphasis on the *Agile* ones. After that, a deep analysis of the AP is done, identifying a set of problematic issues.

The *Part III, Proposal* presents an objective proposal for a software development model, adapted to the AP, in order to solve the identified problematics and reaching the objectives proposed for this dissertation.

The *Part IV Conclusions* promotes an overview of what was presented and discussed in this thesis, presenting a set of conclusions and improvement suggestions that could have been explored.

Part II

Analysis

Analysis and synthesis ordinarily clarify matters for us about as much as taking a Swiss watch apart and dumping its wheels, Sprints, hands, threads, pivots, screws and gears into a layman's hands for reassembling, clarifies a watch to a layman.

Author Unknown

Chapter 3

Software Engineering

3.1 Introduction

According to the IEEE, software is a collection of computer programs, procedures, rules, associated documentation and data. On the other hand, software engineering is the discipline of software process, development, practices and tools. It comprises standard specification, design techniques, formal analysis, established processes, architecture, and more. It addresses practical constraints and involves mathematics, management, psychology, and economics in addition to computer science.

Software engineering has evolved steadily from its founding days in the 1940s until today in the 2000s. Still, the term software engineering first appeared in the late 1950s and early 1960s. The difficulties of building big software loomed so large that, in 1968 the NATO Science Committee (Garmisch, Germany) convened some 50 top programmers, computer scientists and captains of industry to plot a course out of what had come to be known as the “*software crisis*”¹. Although the experts could not contrive a road map to guide the industry toward firmer ground, they did coin a name for that distant goal: software engineering, now defined formally as “*the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software*”. Many believe this conference marked the official start of the discipline of software engineering².

¹in fact the term was coined in this conference by F. L. Bauer

²[WP0(2008b), Mahoney(2004)]

3.1.1 Software Crisis

In the root of the software crisis are a set of causes linked to the overall complexity of developing software, as well as the immaturity of software engineering as a discipline. This crisis was originally defined in terms of productivity, but evolved to emphasize quality and had manifested itself in several ways³:

- Projects cost and budget overruns
- Projects running over-time
- Software was of low quality
- Software often did not meet requirements
- Projects were unmanageable and code difficult to maintain

3.1.2 Silver Bullet

For decades, solving the software crisis was paramount to researchers and companies producing software tools. Seemingly, they trumpeted every new technology and practice from the 1970s to the 1990s as a “*silver bullet*” to solve the software crisis⁴. The expression “*silver bullet*” is a metaphor that applies to any straightforward solution perceived to have extreme effectiveness. The phrase typically appears with an expectation that some new technology or practice will easily cure a major prevailing problem.

As a matter of fact, tools, disciplines, formal methods, process, and qualification were touted as silver bullets:

- **Tools** - structured programming, object-oriented programming (C++, Ada, Java), Computer-Aided Software Engineering⁵ (CASE) tools, documentation, standards, Unified Modeling Language (UML), ...
- **Discipline** - some pundits argued that the software crisis was due to the lack of discipline of programmers.
- **Formal methods** - some believed that if formal engineering methodologies would be applied to software development, then production of software would become as predictable an industry as other branches of engineering. They advocated proving all programs correct.

³[WP0(2008c)]

⁴[WP0(2008b)]

⁵Consists on the use of software tools to assist in the development and maintenance of software. Tools used to assist in this way are known as CASE Tools

- **Process** - many advocated the use of defined processes and methodologies like the **Capability Maturity Model (CMM)**.
- **Qualification** - lack experience and quality of software engineers, which has been increasing during the years.

In 1987, Fred Brooks published the “*No Silver Bullet*” article⁶, arguing that:

Not only are there no silver bullets now in view, the very nature of software makes it unlikely that there will be any—no inventions will do for software productivity, reliability, and simplicity what electronics, transistors, and large-scale integration did for computer hardware

Still, advocates of several tools, processes and others continued arguing for years that their favorite technology would be a silver bullet. Skeptics disagreed. Eventually, almost everyone accepted that no silver bullet would ever be found. Yet, claims about silver bullets pop up even today.

3.1.2.1 Why Developing Software is Difficult

The complexity of software is an essential property, not an accidental one⁷. Many of the classic problems of developing software products derive from this essential complexity and its nonlinear increases with size. On the other hand, the algorithmic nature of software brings a set of complex issues, like for instance:

- **Brittleness** - an algorithm is not unlike a chain. Break a link and the entire chain is broken. As a result, algorithmic programs tend to suffer from catastrophic failures even in situations where the actual defect is minor and globally insignificant.
- **Temporal Inconsistency** - with algorithmic software it is virtually impossible to guarantee the timing of various processes because the execution times of subroutines vary unpredictably. They vary mainly because of a construct called “*conditional branching*”, a necessary decision mechanism used in instruction sequences. But that is not all. While a subroutine is being executed, the calling program goes into a coma. The use of threads and message passing between threads does somewhat alleviate the problem but the multithreading solution is way too coarse and unwieldy to make a difference in highly complex applications. And besides, a thread is just another algorithm. The inherent temporal uncertainty (from the point of view of the programmer) of algorithmic systems leads to program decisions happening at the wrong time, under the wrong conditions.

⁶[Brooks(1987)]

⁷[Savain(2006)]

- **Unresolved Dependencies** - the biggest contributing factor to unreliability in software has to do with unresolved dependencies. In an algorithmic system, the enforcement of relationships among data items (part of what Fred Brooks defines as the essence of software⁸) is solely the responsibility of the programmer. That is to say, every time a property is changed by a statement or a subroutine, it is up to the programmer to remember to update every other part of the program that is potentially affected by the change. The problem is that relationships can be so numerous and complex that programmers often fail to resolve them all. Fortunately, in the last years IDEs like Eclipse tend to help the developers with this issue.

There is an uncountable number of projects that throughout the years did not see the day light. The primary reason for these failures, other than those mentioned above, is due to bad software engineering practices adopted⁹. Some of the worst software practices include:

- No historical software-measurement data
- Rejection of accurate cost estimates
- Failure to use automated estimating and planning tools
- Excessive, irrational schedule pressure and creep in user requirements
- Requirements change very often
- Failure to monitor progress and to perform risk management
- Failure to use design reviews and code inspections

In order to avoid these and other failures, one approach could pass by a better understanding of the software development process, better estimation techniques and quality measures. Meanwhile, from the complexity comes the difficulty of communication among team members. Albert Einstein once argued that there must be simplified explanations of nature, with the quote:

God does not play dices

According to Fred Brooks:

No such faith comforts the software engineer. Much of the complexity that he must master is arbitrary complexity, forced without rhyme or reason by the many human institutions and systems to

⁸[Brooks(1995)]

⁹[SE0(2007)]

which his interfaces must conform. These differ from interface to interface, and from time to time, not because of necessity but only because they were designed by different people, rather than by God

This means that besides all the constraints aforementioned, software is done by persons, and this introduces a high degree of uncertain. People are human, they have their individual personalities, concerns and ideas, and they do mistakes as well, and this and other things affects widely the software development. There is a book called the *“Peopleware — Productive Projects and Teams”*¹⁰, from Tom DeMarco, that says that:

The major problems of our work are not so much technological as sociological in nature

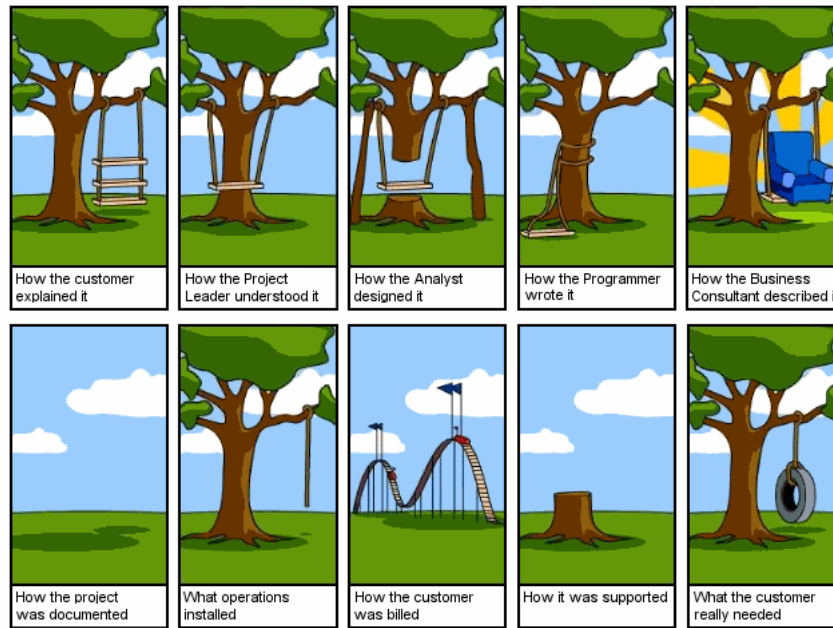


Figure 3.1: Software Engineering

A software development process, more than looking to the technical aspects, must look to the organization, the persons that compose that organization, trying to understand their needs and expectations, in order to create a structure capable to absorb that, and able to give responses.

¹⁰[Tom DeMarco(1987)]

3.2 Software Development Process (SDP)

3.2.1 Definition

A process is a way to transform inputs to outputs i.e. a product. So, a software development process (SDP) is a set of activities, methods and practices involving transformation that people use to develop and maintain software. According to the definition provided by the wikipedia¹¹, a software development process is *“a structure imposed on the development of a software product. Synonyms include software life cycle and software process”*. Concerning the word process, the english dictionary says that *“is a sequence or arterial network of logically related and time-based work activities to provide a specific output for an internal or external customer”*.

3.2.2 The Need for a Defined SDP

At present a large number of problems exist due to a chaotic software development process and the occasional success depends on individual efforts. Therefore to be able to deliver successful software projects, a focus on the process is essential. This focus would help in the predictability of outcomes, project trends, and project characteristics. On the other hand, when several people work cooperatively on a common project, they need somehow to coordinate their work. In the case of relatively small or simple tasks, this can often be done in a informal way, but with a large number of people or more sophisticated activities, more formal arrangements are needed. For example, process definition can be compared with football training. While the sequence of plays will change from game to game, the winning team generally has worked out their plays in advance, knows when to use them, and can perform them with skill. The software process is much the same. Unfortunately, not to many software teams work out their plays in advance, even though they know the key problems they will encounter. In fact, for the same reason, they act as if late requirements changes, regressions, or system integration problems will not recur.

The software process is a technical and management framework established for applying tools, methods, and people to the software task. A defined process not only prepares for likely eventualities; it also provides a mechanism for organized learning. As projects improve their processes for handling key tasks, these can be incorporated in the repertoires of *“plays”* available to the rest of the organization. This process definition makes it easier for each new project to build on the experiences of its predecessors, and it also protects against the dangers of ill-prepared crisis reactions.

¹¹[WP0()]

An important observation of process management¹² is that process changes adopted in a crisis are generally misguided. Crises are the times when shortcuts are most likely and when organizations are most prone to omit critical tasks. These shortcuts often lead to truncated testing, skipped inspections, and deferred documentation. With time at a premium, rationalization is most likely and process judgments are least reliable. Because it is difficult to justify many tests, analysis, or inspections in the heat of the moment, a thoughtfully defined and approved process can be a great help. When the crisis occurs, it has been anticipated, and established means are at hand to deal with it. In short, a defined process provides the software professionals with the framework they need to do a consistently professional job¹³.

3.2.3 The Need for a Standard SDP

While there are often needs for project-specific process tailoring's, there are also compelling reasons for a standard SDP framework, such as the following ones:

- Process standardization is required to permit training, management, review, and tool support
- With standard processes, each project's experiences can contribute to overall process improvement in the organization
- Process standards provide a structured basis for measurement
- Because process definitions take time and effort to produce, it is impractical to produce new ones for each project
- Because the basic tasks are common to most software projects, a standard process framework will need only modest customization to meet most special project needs

¹²process management is concerned with the knowledge and management of the software development process, its technical aspects and also ensures that everything is being followed as expected and improvements are shown

¹³[Humphrey(1989)]

Chapter 4

Software Development Models

4.1 Introduction

Software development processes are usually referred to as models and, essentially, a model is a particular abstraction that describes phases of a software development process effort, and the order in which those phases are executed. There are several different kind of models known and used, and many companies adopt their own. However, many of them are minor variations on just a small number of basic models. Generally, the classic or *heavyweight* software development models include the following activities:

- **Domain analysis** - consists in the investigation of the domain on which the new software will be build. it is a kind of prelude to extract and gather the requirements.
- **Requirement analysis and specification** - this involves all the process of analyzing and gathering the requirements for which the software is being developed. This step demands, as well, the scope analysis of the candidate product, to clarify what is pretended, in order to actually specify and describe the requirements in a rigorous way. The essential purpose of this phase is to find the need and to define the problem that needs to be solved.
- **System analysis and design** - consists in the specification of an abstract representation of the system, having always in mind the requirements of the product, as well as ensuring that future requirements can be addressed. Analysis and design are very crucial in the whole development cycle. Any glitch in the design phase could be very expensive to solve in the later stage of the software development. Much care is taken during this phase by the fact that the logical system of the product is developed here.

- **Implementation** - consists in the conversion of the previous specification into code. If the design is performed in a detailed manner, code generation can be accomplished without much complication. With respect to the type of the application, the right programming language is chosen.
- **Testing** - consists in the process of testing the parts of the software. The importance of this step increases according to the size of the project, and consequently, to the number of developers involved on that. This phase is often split into three separated phases: Unit testing, Integration testing and Acceptance testing. The first two may be part of a repeated cycle of coding and testing, while acceptance testing verifies requirements compliance.
- **Deployment** - after the implementation and the testing, and once the product has been approved, it is moved into production environment, in order to be available for business use.
- **Documentation** - consists in the documentation of the design of the software, for the purpose of future maintenance and improvement.
- **Maintenance** - refers to all the process of maintaining and improving the software to handle with discovered problems, which can go from simple fixing bugs to extensions of the system. The software should be developed to accommodate changes that could happen during the post implementation period.

In the following topics, will be done an overview of the major known software development models, starting by the ones that are called the classic or heavy-weight ones. These models follow variations of the activities structure presented above. The description of each model will try to emphasize the his characteristics, advantages, disadvantages and applications.

4.2 Waterfall

The Waterfall model is a highly structured development model that was proposed by Winston W. Royce in 1970¹, and it is the most common and classic software development model. It is really simple to understand and use, and is a linear sequential model. In this model each phase must be completed in its entirely before the next phase can begin. At the end of each phase, a review takes place to determine if the project is on the right path. This model is in fact, considered the “*traditional*” approach to software development and was derived

¹[Royce(1970)]

from defense and aerospace projects processes. In fact, it is still used by several companies and, for instance, by the U.S Department of Defense² and NASA³.

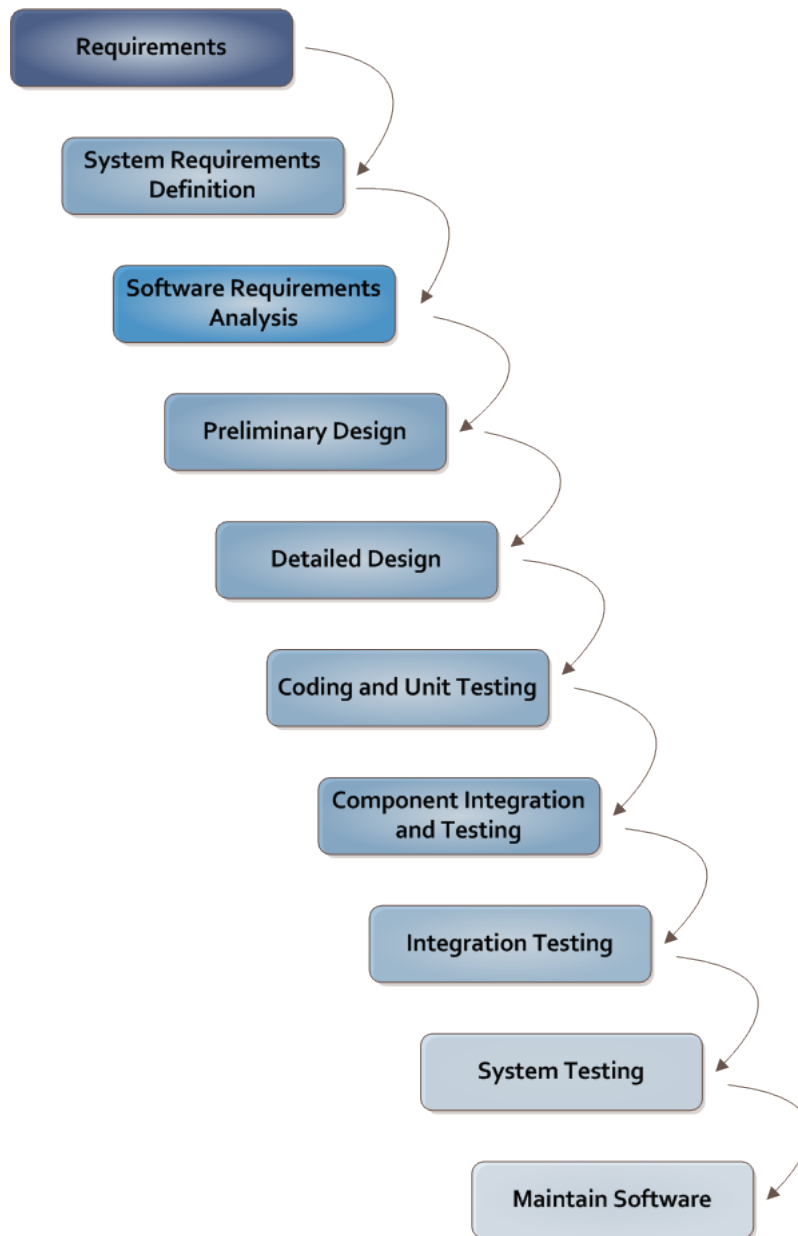


Figure 4.1: Waterfall model

²U.S Department of Defense <http://www.defenselink.mil/>

³National Aeronautics and Space Administration www.nasa.gov

The Waterfall model is documentation-intensive, with earlier phases documenting what must be done, and subsequent phases, adding greater detail and defining how it should be done. The output from one phase serves as the input to the next phase, with the project flowing from one step to the next in a waterfall fashion. Phases are assumed to be sequential, with only localized feedback during the transition between phases. This is accomplished by using reviews as gates. Comprehensive reviews validate the work of one phase, and require the resolution of any problems before development is allowed to proceed to the next phase.

An important consideration for the waterfall model is that fixes or modifications are often put off until the maintenance phase. This can be very costly, as the cost to correct a problem gets higher with each successive phase. This gives a capital importance to the requirements definition and analysis as well. Requirements should be set in stone before design. For all these reasons, the waterfall model can be described as a heavyweight process.

4.2.1 Advantages vs Disadvantages

Advantages

- Simple and easy to use
- Easy to manage due to the rigidity of the model
 - Each phase has specific deliverables and a review process
- System is well documented
- Phases correspond with project management phases⁴
- Cost and schedule estimates may be lower and more accurate
- Time spent early on making sure that requirements and design are absolutely correct will save you much time and effort later
- Details can be addressed with more engineering effort if software is large or complex

Disadvantages

- All risks must be dealt with in a single software development effort
- Because the model is sequential, there is only local feedback at the transition between phases

⁴this topic may require some background in the subject of project management

- Significant adjustments in the scope during the development process can kill a project
- Corrections must often wait for the maintenance phase
- Progress and success are not observable until the later stages
 - If a mistake or deficiency exists in the documentation of earlier phases, it may not be discovered until the product is delivered
- No working software is produced until late during the development process

4.2.2 Application

The Waterfall model can be successfully used when requirements are well understood in the beginning and are not expected to change or evolve over the life of the project. The project risks should be relatively low. Heavyweight processes, such like this one, can be a good choice when you have multiple teams working at different locations and you need tighter control to formalize key parts of the project⁵.

4.3 Incremental

The Incremental model is a intuitive approach to the waterfall model and consists essentially in a series of waterfall cycles. Those cycles are splitted up into smaller and more easily managed iterations. The requirements are known at the beginning of the project and are divided into groups for incremental development. A core set of functions is identified in the first cycle or iteration and is built and deployed as the first release. So, we have working software early on during the software development process. The software development process is repeated, with each release adding more functionality until all requirements are met. Each development cycle acts as the maintenance phase for the previous software release. While new requirements that are discovered, during the development of a given cycle, they can be implemented in subsequent cycles.

However, this model assumes that most requirements are known up front. The effort is planned and executed to satisfy the initial list of requirements. A modification to the incremental model allows development cycles to overlap, this means that, a subsequent cycle may begin before the previous cycle is complete.

⁵[SDP(2003), Lewallen(2005)]

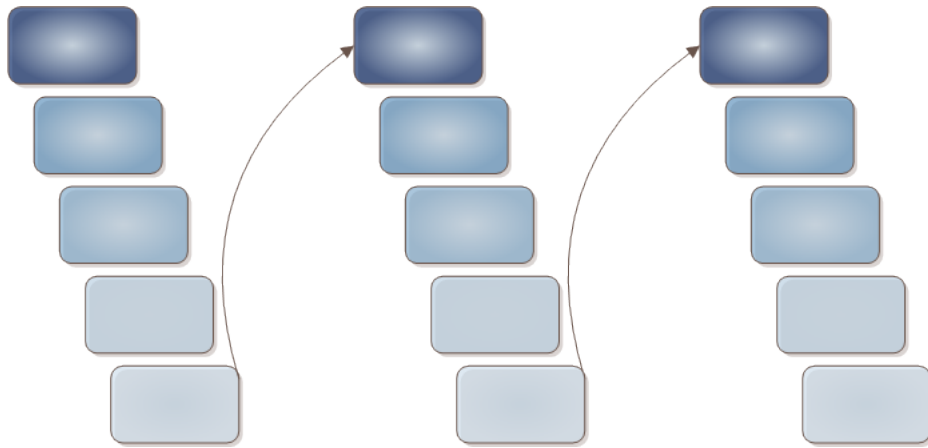


Figure 4.2: Incremental model

4.3.1 Advantages vs Disadvantages

Advantages

- Provides some feedback, allowing later development cycles to learn from previous cycles
- Requirements are relatively stable and may be better understood with each increment
- Allows some requirements modification and may allow the addition of new requirements
- It is more responsive to user needs than the waterfall model
- A usable product is available with the first release, and each cycle results in greater functionality
- The project can be stopped any time after the first cycle and leave a working product
- Risk is spread out over multiple cycles
- This method can usually be performed with fewer people than the waterfall model
- Each iteration is an easily managed milestone
- Easier to test and debug during a smaller iteration

Disadvantages

- The majority of requirements must be known in the beginning
- The development is spread out over multiple iterations, which demands well-defined interfaces between modules since the beginning
- Cost and schedule overruns may result in an unfinished system
- Operations are impacted as each new release is deployed
- Users are required to learn how to use a new system with each deployment

4.3.2 Application

The Incremental model is good for projects where requirements are known at the beginning, but which need functionality early in the project or which can benefit from the feedback of earlier cycles. Because each cycle produces a working system, it may also be advantageous for projects whose continued funding is not assured and may be cut at any time. It is best used on low to medium-risk programs. If the risks are too high to build a successful system using a single waterfall cycle, spreading the development out over multiple cycles may lower the risks to a more manageable level⁶.

4.4 Evolutionary

The Evolutionary model, like the incremental model, develops a product in multiple cycles. However, unlike the incremental model, which simply adds more functionality with each cycle, this model produces a more refined prototype system with each iteration, following what is denominated as incremental and iterative development (IID) process.

The process, begins in the center with initial requirements and plans, and progresses through multiple cycles of planning, risk analysis, engineering, and customer evaluation. Each cycle produces a prototype that the customer evaluates, followed by a refinement of requirements.

Specification, development, and testing activities are carried out concurrently with rapid feedback. Since requirements continue to change, documentation is minimal, although essential information must still be included for understanding the system and for future support. Implementation compromises are often made in order to get the prototype working – permanent fixes can be made with the next prototype. Operational capability is achieved early, but users must be willing to learn how to use each new prototype.

⁶[SDP(2003), Lewallen(2005)]

On the other hand, general system requirements must be known prior to development. This is particularly helpful where evolving technology is being introduced into the project. The evolutionary model relies heavily on user feedback after each implementation to refine requirements for the next evolutionary step⁷.

4.4.1 Advantages vs Disadvantages

Advantages

- Project can begin without fully defining or understanding requirements
- Final requirements are improved and more in line with real user needs
- Risks are spread over multiple software builds and controlled better
- Operational capability is achieved earlier in the program
- Newer technology can be incorporated into the system as it becomes available during later prototypes
- Documentation emphasizes the final product instead of the evolution of the product
- This method combines a formal specification with an operational prototype

Disadvantages

- Because there are more activities and changes, there is usually an increase in both cost and schedule over the waterfall method
- Management activities are increased.
- Instead of a single switch over to a new system, there is an ongoing impact to current operations
- Configuration management activities are increased
- Greater coordination of resources is required
- Users sometimes mistake a prototype for the final system
- Prototypes change between cycles, adding a learning curve for developers and users
- Risks may be increased in the following areas:

⁷[SDP(2003), Lewallen(2005)]

- Requirement - temptation to defer requirements definition
- Approval – vulnerable to delays in funding approval, which can increase schedule and costs
- Architectural – initial architecture must accommodate later changes
- Short term benefits – risk of becoming driven by operational needs rather than program goals
- Risk avoidance – tendency to defer riskier features until later
- Patchwork quilt effects – if changes are poorly controlled, the product quality can be compromised

4.4.2 Application

The Evolutionary model can be employed on most types of acquisitions. However, it is usually employed on medium to high-risk systems. The evolutionary model should be considered for systems where requirements are not all known or not yet refined, but are expected to evolve. It is more suitable to be used with the development of new systems than upgrading existing software. The developing and using organizations must be flexible and willing to work with evolving prototypes. Programs well suited to employ the evolutionary model have some or all of the following characteristics:

- Software intensive systems
- Have a large number of diverse users
- Have rapidly changing software technology
- Developing an unprecedented system
- Humans are an integral part of the system
- Limited capability is needed quickly

4.5 Spiral

The Spiral model was developed with the goal of reducing risk in the software life cycle. It combines elements of the Waterfall, Incremental and Evolutionary models, and depending on how it is implemented can strongly resemble any combination of the others. The project starts at the center (take a look to the figure) and progresses through multiple cycles, each working through the software development activities associated with the four quadrants:

1. Determine objectives, alternatives and constraints
2. Evaluate alternatives. Identify and resolve risks
3. Develop the next-level product
4. Plan the next phase

Risk management is a key element of the Spiral model and each round of the spiral identifies problems with the highest risk and develops solutions for that set of problems. The process may even resemble a waterfall with additional risk management techniques. Each cycle ends in a review in which stakeholders agree on plans for the next cycle. While a prototype may be produced, software is usually not developed for release until the last cycle.

The Spiral model has been used extensively in the commercial world because of its performance in a market-driven environment. It significantly reduces technical risk and more easily incorporates new technology and innovations.

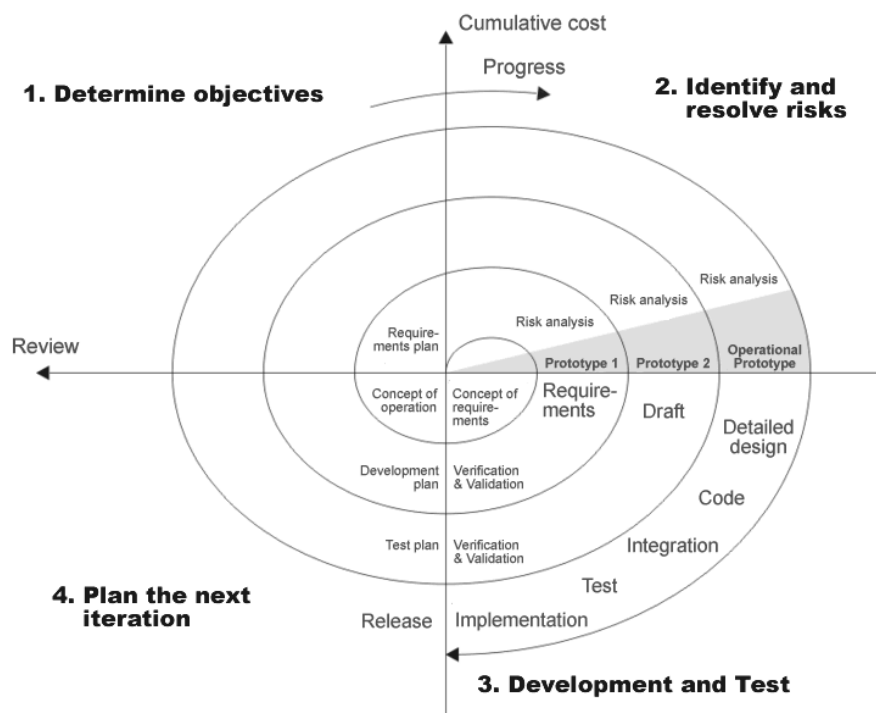


Figure 4.3: Spiral model

4.5.1 Advantages vs Disadvantages

Advantages

- It provides better risk management than other models
- Requirements are better defined
- System is more responsive to user needs

Disadvantages

- The spiral model is more complex and harder to manage
- This method usually increases development costs and schedule

4.5.2 Application

The Spiral method should be considered for projects where risks are high, requirements must be refined, and user needs are very important⁸.

4.6 Iterative and Incremental Development (IID)

In this stage, it is important to make the distinction between incremental development practices, and iterative ones. Incremental development is a scheduling and staging strategy in which, the various parts of the system, are developed at different times or rates, and integrated as they are completed. Each increment is fully coded and tested, and the common expectation is that the work of an iteration will not need to be revisited. It does not imply, require nor preclude Iterative development or Waterfall development - both of those are rework strategies. This type of development is clearly used in the Waterfall⁹ and Incremental¹⁰ models. However, the Evolutionary¹¹ and Spiral¹² models combine the elements of the Waterfall model with the iterative philosophy of prototyping¹³.

Iterative development is a rework scheduling strategy in which time is set aside to revise and improve parts of the system. It does not presuppose incremental development, but works very well with it. A typical difference is that the output from an increment is not necessarily subject to further refinement,

⁸[SDP(2003), Lewallen(2005)]

⁹ 4.2 on page 26

¹⁰ 4.3 on page 29

¹¹ 4.4 on page 31

¹² 4.5 on page 33

¹³prototyping is the process of quickly putting together a working model (a prototype) in order to test various aspects of a design, illustrate ideas or features and gather early user feedback

and its testing or user feedback is not used as input for revising the plans or specifications of the successive increments. On the contrary, the output from an iteration is examined for modification, and especially for revising the targets of the successive iterations. As it was described previously, the Evolutionary model uses the iterative, but also the incremental development¹⁴.

However, there is some controversy concerning this subject. Some argue that the word “*iterative*” development is badly chosen¹⁵, in the sense that, it might hide the idea that iterations can offer to a development process two key things: iterative refinement, where the process improves what already exists and is being done, and incremental development, where the process results in progress against project objectives. Additionally, the term increment has the obvious implication that, there should be more of something at the end of an iteration than there was at the start. Without a clear notion of an increment, iterations are likely to just go in circles¹⁶.

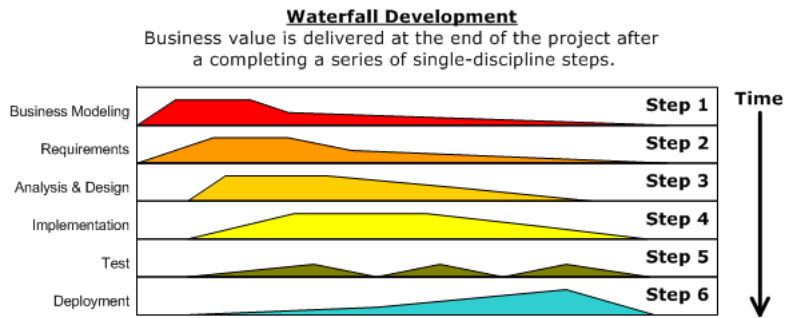


Figure 4.4: Waterfall Development

¹⁴[SDP(2008)]

¹⁵[Cockburn(2008)]

¹⁶[Henney(2007)]

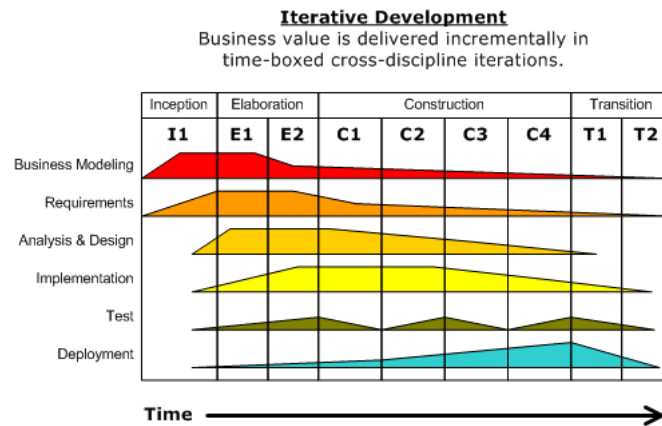


Figure 4.5: Iterative Development

4.6.1 Brief History

The IID grew from the 1930s work of Walter Shewhart, a quality expert at Bell Labs who proposed a series of short “*Plan-Do-Study-Act*” (PDSA) cycles for quality improvement. Starting in 1940s, quality guru W. Edwards Deming began vigorously promoting PDSA, which he later described in 1982 in *Out of the Crisis*. In 1970, Winston W. Royce published the *Managing the Development of Large Software Systems*, in which he shares his view about software development, and on what would become known as the Waterfall model¹⁷. In 1975, Vic Basili and Joe Turner published a paper about the iterative enhancement that clearly described classic IID:

The basic idea behind iterative enhancement is to develop a software system incrementally, allowing the developer to take advantage of what was being learned during the development of earlier, incremental, deliverable versions of the system. Learning comes from both the development and use of the system, where possible. Key steps in the process were to start with a simple implementation of a subset of the software requirements and iteratively enhance the evolving sequence of versions until the full system is implemented. At each iteration, design modifications are made along with adding new functional capabilities.

¹⁷many - incorrectly - view Royce’s paper as the paragon of single-pass waterfall. In reality, he recommended an approach somewhat different than what has devolved into today’s waterfall concept.

In 1976, Tom Gilb¹⁸ published *Software Metrics*¹⁹, in which he discussed his IID practice - evolutionary project management - and introduced the terms “*evolution*” and “*evolutionary*” to the process lexicon. This is the earliest found book that had a clear IID discussion and promotion, especially of evolutionary delivery:

“Evolution” is a technique for producing appearance of stability. A complex system will be most successful if it is implemented in small steps and if each step has a clear measure of successful achievement as well as a “retreat” possibility to a previous successful step upon failure. You have the opportunity of receiving some feedback from the real world before throwing in all resources intended for a system, and you can correct possible design errors. . .

The book marked the arrival of a long-standing and passionate voice for evolutionary and iterative development. In 1980, Weinberg wrote about IID in *Adaptive Programming: The New Religion* published in Australasian Computerworld. Summarizing the article, he said:

The fundamental idea was to build in small increments, with feedback cycles involving the customer for each

A year later, Tom Gilb wrote in more detail about evolutionary development. In another mid-1980s questioning of the sequential life cycle, Gilb wrote *Evolutionary Delivery versus the Waterfall Model*²⁰. In this paper, Gilb promoted a more aggressive strategy than other IID discussions of the time, recommending frequent (such as every few weeks) delivery of useful results to stakeholders. In 1988, Gilb published *Principles of Software Engineering Management*²¹, the first book with substantial chapters dedicated to IID discussion and promotion.

By the 1990s, especially the latter half, public awareness of IID in software development was significantly accelerating. Hundreds of books and papers were promoting IID as their main or secondary theme. Dozens more IID processes sprang forth, which shared an increasing trend to time-boxed iterations of one to six weeks. It was during this period that had appear IID processes like Scrum, FDD, XP, AUP, DSDM and many others. In February 2001, a group of 17 process experts interested in promoting modern, simple IID processes and principles met in Utah to discuss common ground. From this meeting came the *Agile Alliance* and the now popular catch phrase “*agile processes*”, all of which apply IID²².

¹⁸Tom Gilb is an American systems engineer and author

¹⁹[Gilb(1976)]

²⁰[Gilb(1985)]

²¹[Gilb(1988)]

²²[Craig Larman(2003)]

4.7 Agile Software Development

The agile software development is a conceptual framework that promotes development iterations throughout the life cycle of a project²³. As it was aforementioned, the modern definition of agile software development evolved in the mid-1990s as part of a reaction against the so called “*heavyweight*” processes, as typified regulated, regimented, micro-managed use of the Waterfall model of development. The processes based on the use of the Waterfall model were seen as bureaucratic, slow and inconsistent with ways that software developers actually perform effective work.

Agile processes were called “*lightweight*” processes, and are described as a style of software development characterized by an emphasis on people, communication, working software, and responding to change. Its principles are summarized in its seminal declaration, the *Manifesto for Agile Software Development*. One of the most notable characteristics of this manifesto is that, it is worded as a value statement and not a concrete plan or process. This underscores Agile’s nature as a style of development consistent with the values laid out in the manifesto.

All agile models engage in an iterative workflow and incremental delivery of working software in short time-boxed iterations. An iteration is essentially a small release of software. Generally, during each iteration, many activities will occur in parallel, such as requirements, coding, and testing. Iterations are typically a fixed length (although this length changes between the different models) and thus are referred to as time-boxed. The time allocated to each iteration is sometimes referred to as a cycle time.

On the other hand, agile models also differ in the activities and artifacts that they produce. The ones that have many artifacts (or work products) and steps along the way are referred to as “*higher ceremony*”. Conversely, models with less emphasis on these are referred to as “*lower ceremony*”. The number of people involved and the number of “*sign offs*” also helps to define the amount of ceremony around a particular model. It is important to strike an appropriate balance between the two in any given context.

There are several agile models available. However, the ones that seem to be more popular nowadays are the following ones:

- Scrum
- Extreme Programming (XP)
- Feature Driven Development (FDD)

²³[WP0(2008a)]

- Dynamic Systems Development Method (DSDM)
- Adaptive Software Development (ASD)
- Crystal
- Lean
- Agile Unified Process (AUP)
- ...

4.7.1 Manifesto for Agile Software Development²⁴

The *Agile Manifesto* is a statement of the principles that underpin the agile software development. It was drafted from 11 to 13 February, at The Lodge at Snowbird ski resort in the Wasatch mountains of Utah. It was signed by representatives of various new software development models such as XP, Scrum, DSDM, Adaptive Software Development, Crystal, FDD, Pragmatic Programming, and others sympathetics, that met to discuss the need for lighter alternatives to the traditional heavyweight processes.

We are uncovering better ways of developing

software by doing it and helping others do it.

Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on

the right, we value the items on the left more.

²⁴[SDP(2001a)]

4.7.1.1 Principles Behind²⁵

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a number of weeks to a number of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity—the art of maximizing the amount of work not done—is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

4.7.2 The People Factor

The most important implication to managers working in a agile manner is that it places more emphasis on people factors in the project: amicability, talent, skill, and communication. These qualities become a primary concern for the

²⁵[SDP(2001b)]

would-be agile team. The attention to the human issues gives agile projects a particular feel. It is not only a matter of reducing paperwork and hoping that everything else will fall into place. Agile development teams focus on individual competency as a critical factor in project success. Too often, software engineering and rigorous process adherents confuse process and competence. Process can provide a useful framework for groups of individuals to work together, but process *per se* cannot overcome a lack of competency, in fact “*people trump process*“. When we look under the cover of XP, Scrum, Adaptive Software Development, Crystal Methods, FDD or DSDM, the emphasis on people and their talent, skill, and knowledge becomes evident²⁶.

A project is built from people having differing personalities and differing skills, working in a physical environment within an organizational culture. The people, environment, and organizational culture all influence one another. Persons are not like bricks that can be easily replaced by others with the same shape. Each person as his own. When a strong person leaves, the organization rearranges itself to compensate; when the team spreads itself across multiple floors, communications change; and so on. The project is an ecosystem.

Using the term ecosystem to convey the totality of the project at hand, brings us to the discussion of processes and ecosystems and fitting the two together. Installing a process within an ecosystem offers two choices: Fit the ecosystem to the process or fit the process to the ecosystem. In reality, both the process and ecosystem change, but the agile approach honors the ecosystem and recognizes that not every process will work in every ecosystem.

4.8 CMMI

4.8.1 Description

In 1991 the Carnegie Mellon® Software Engineering Institute (SEI), with major funding from the U.S Department of Defense, unveiled its Capability Maturity Model (CMM). The CMM is a process capability maturity model which aids in the definition and understanding of an organization’s processes. The CMM is a process that, at last, persuaded many programmers to concentrate on measuring the process by which they produce software, a prerequisite for any industrial engineering discipline.

In fact, the CMM had a major influence on software development process and quality improvement by his use for assessing organizational maturity and process capability throughout the world. Based on the CMM, the Capability Maturity Model Integrated (CMMI) has been presented in 2000, integrating

²⁶[Cockburn and Highsmith(2001)]

CMM for Software (SW-CMM), the Capability Model for Systems Development (EIA/IS 731) and the CMM for Integrated Product Development (IPD-CMM). The current release of CMMI is version 1.2 and there are two available version 1.2 models:

- **CMMI for Development (CMMI-DEV)** - version 1.2 was released in August 2006. It addresses product and service development processes.
- **CMMI for Acquisition (CMMI-ACQ)** - version 1.2 was released in November 2007. It addresses supply chain management, acquisition, and outsourcing processes in government and industry.

A **Software Process Improvement (SPI)** assumes that a well-managed organization, with a defined engineering process, is more likely to produce software that consistently meets the users' requirements, within schedule and budget, than a poorly managed organization with no such engineering process. In other words, the project failure is usually a process failure. CMMI – as SPI's describes managerial processes to attack software development difficulties at five maturity levels:

1. **Initial** - processes are usually ad hoc and chaotic. The organization traditionally does not provide a stable environment for developing and maintaining software. Success in these organizations depends on the competence and heroics of the people in the organization and not on the use of proven processes. In spite of this ad hoc, chaotic environment, maturity level 1 organizations often produce products and services that work; however, they frequently exceed the budget and schedule of their projects. Maturity level 1 organizations are characterized by a tendency to over commit, abandon processes in the time of crisis, and not be able to repeat their past successes. Success depends on having an exceptional manager and a seasoned and effective software team. So at level 1, capability is a characteristic of the individuals, not of the organization.
2. **Repeatable** - some software development processes are repeatable, possibly with consistent results. Still, the processes may not repeat for all the projects in the organization. On the other hand, the organization may use some basic project management to track cost and schedule. Process discipline is unlikely to be rigorous, but where it exists, it may help to ensure that existing practices are retained during times of stress. When these practices are in place, projects are performed and managed according to their documented plans. Project status and the delivery of services

are visible to management at defined points²⁷. Basic project management processes are established to track cost, schedule, and functionality. The minimum process discipline is in place to repeat earlier successes on projects with similar applications and scope. However, there is still a significant risk of exceeding cost and time estimates.

3. **Defined** - The organization's set of standard processes, which is the basis for maturity level 3, is established and improved over time. These standard processes are used to establish consistency across the organization. Projects establish their defined processes by tailoring the organization's set of standard processes according to tailoring guidelines. The organization's management establishes process objectives based on the organization's set of standard processes, and ensures that these objectives are appropriately addressed.

A critical distinction between maturity level 2 and maturity level 3 is the scope of standards, process descriptions, and procedures. At maturity level 2, the standards, process descriptions, and procedures may be quite different in each specific instance of the process (for example, on a particular project). At maturity level 3, the standards, process descriptions, and procedures for a project are tailored from the organization's set of standard processes to suit a particular project or organizational unit. The organization's set of standard processes includes, the processes addressed at maturity level 2 and maturity level 3. As a result, the processes that are performed across the organization are consistent, except for the differences allowed by the tailoring guidelines.

4. **Managed** - using process metrics, management can effectively control the process. In particular, management can identify ways to adjust and adapt the process to particular projects without measurable losses of quality or deviations from specifications. Organizations at this level set quantitative quality goals for both software process and software maintenance. Sub-processes are selected that significantly contribute to overall process performance. These selected sub-processes are controlled using statistical and other quantitative techniques. A critical distinction between maturity level 3 and maturity level 4 is the predictability of process performance. At maturity level 4, the performance of processes is controlled using statistical and other quantitative techniques, and may be quantitatively predictable. At maturity level 3, processes are only qualitatively predictable.

27

(a) for example, at major milestones and at the completion of major tasks

5. **Optimizing** - At maturity level 5, an organization has achieved all the specific goals of the process areas assigned to maturity levels 2, 3, 4, and 5 and the generic goals assigned to maturity levels 2 and 3. Processes are continually improved, based on a quantitative understanding of the common causes of variation inherent in processes. Maturity level 5 focuses on continually improving process performance, through both incremental and innovative technological improvements. Quantitative process-improvement objectives for the organization are established, continually revised to reflect changing business objectives, and used as criteria in managing process improvement. The effects of deployed process improvements are measured and evaluated against the quantitative process-improvement objectives. Both the defined processes and the organization's set of standard processes are targets of measurable improvement activities.

It is important to note that the CMMI process models do not contain prescriptive processes that can be used right out of the box. Instead, CMMI provides a way to assess the state of an organization's ability to build software in a repeatable, predictable way. Applying CMMI as a mean to increase process capabilities is an organization-wide challenge.

The average time for an organization to move up one level is between 21 and 37 months, of course this depends a lot from the size of the organization. Still, over three quarters of the organizations that submitted to the CMMI evaluation, reported that implementing any key SPI activity took longer than expected. But the effort pays off since *“software process management maturity is positively associated with project performance”*.

In 1994, 75% of the evaluated organizations were still stuck in level 1. They had no formal process, no measurements of what they do, and no way of knowing when they were on the wrong track or off the track altogether. The remaining 24% were at levels 2 or 3. Only two elite groups have earned the highest CMM rating, a level 5. Motorola²⁸'s Indian programming team in Bangalore held one title. Loral's²⁹ (formerly IBM's) on-board space shuttle software project claimed the other. The Loral team has learned to control bugs so well that it were able to reliably predict how many would be found in each new version of the software.

In order to reach a certain level, an organization has to fulfill all process areas of that level as well as those of lower levels. A process area is a summary of all requirements for a certain topic, e.g. project management, organizational training or causal analysis and resolution. To satisfy a process area all of its associated goals – specific ones and generic ones – have to be met.

²⁸Motorola <http://www.motorola.com/>

²⁹Loral - Space - Communications <http://www.loral.com/>

Specific goals apply to a process area and address the unique characteristics that describe what has to be implemented to satisfy the process area. To meet a specific goal CMMI suggests a set of specific practices. A specific practice is an activity that is considered important in achieving the associated specific goal.

Generic goals are called “*generic*” because the same goal statement appears in multiple process areas. In the staged representation, each process area has only one generic goal. To meet a generic goal, CMMI suggests a set of generic practices. Generic practices provide institutionalization to ensure that the processes associated with the process area will be effective, repeatable, and lasting. Still, it is important to say that the CMMI model mostly deals with what processes or practices should be implemented, and not so much with how they can be implemented³⁰.

4.8.2 Process Area Categories

There are four process area categories used in CMMI for Development (CMMI-DEV):

- Project Management
- Process Management
- Engineering
- Support

Process Management

- **OID** - Organizational Innovation and Deployment
- **OPD** - Organizational Process Definition +IPPD
- **OPF** - Organizational Process Focus
- **OPP** - Organizational Process Performance
- **OT** - Organizational Training

Project Management

- **PP** - Project Planning
- **PMC** - Project Monitoring and Control
- **SAM** - Supplier Agreement Management

³⁰[Jeffries(2000), Gibbs(1994), Fritzsche and Keil(2007)]

- **IPM** - Integrated Project Management +IPPD
- **RSKM** - Risk Management
- **QPM** - Quantitative Project Management

Engineering

- **REQM** - Requirements Management
- **RD** - Requirements Development
- **TS** - Technical Solution
- **PI** - Product Integration
- **VER** - Verification
- **VAL** - Validation

Support

- **CM** - Configuration Management
- **PPQA** - Process and Product Quality Assurance
- **MA** - Measurement and Analysis
- **DAR** - Decision Analysis and Resolution
- **CAR** - Causal Analysis and Resolution

4.8.3 Maturity Levels

As it was described previously in the **Description**, there are five maturity levels. However, maturity level ratings are awarded for levels 2 through 5.

Maturity Level 2

- **CM** - Configuration Management
- **MA** - Measurement and Analysis
- **PMC** - Project Monitoring and Control
- **PP** - Project Planning
- **PPQA** - Process and Product Quality Assurance
- **REQM** - Requirements Management
- **SAM** - Supplier Agreement Management

Maturity Level 3

- **DAR** - Decision Analysis and Resolution
- **IPM** - Integrated Project Management +IPPD
- **OPD** - Organizational Process Definition +IPPD
- **OPF** - Organizational Process Focus
- **OT** - Organizational Training
- **PI** - Product Integration
- **RD** - Requirements Development
- **RSKM** - Risk Management
- **TS** - Technical Solution
- **VAL** - Validation
- **VER** - Verification

Maturity Level 4

- **QPM** - Quantitative Project Management
- **OPP** - Organizational Process Performance

Maturity Level 5

- **CAR** - Causal Analysis and Resolution
- **OID** - Organizational Innovation and Deployment

4.8.4 Goals and Practices

As it was aforementioned, there are two categories of goals and practices: generic and specific. Specific goals and practices are specific to a process area. Generic goals and practices are a part of every process area. A process area is satisfied when organizational processes cover all of the generic and specific goals and practices for that process area.

Generic Goals and Practices

- **GG 1 Achieve Specific Goals**
 - GP 1.1 Perform Specific Practices
- **GG 2 Institutionalize a Managed Process**
 - GP 2.1 Establish an Organizational Policy
 - GP 2.2 Plan the Process
 - GP 2.3 Provide Resources
 - GP 2.4 Assign Responsibility
 - GP 2.5 Train People
 - GP 2.6 Manage Configurations
 - GP 2.7 Identify and Involve Relevant Stakeholders
 - GP 2.8 Monitor and Control the Process
 - GP 2.9 Objectively Evaluate Adherence
 - GP 2.10 Review Status with Higher Level Management
- **GG 3 Institutionalize a Defined Process**
 - GP 3.1 Establish a Defined Process
 - GP 3.2 Collect Improvement Information
- **GG 4 Institutionalize a Quantitatively Managed Process**
 - GP 4.1 Establish Quantitative Objectives for the Process
 - GP 4.2 Stabilize Sub-process Performance
- **GG 5 Institutionalize an Optimizing Process**
 - GP 5.1 Ensure Continuous Process Improvement
 - GP 5.2 Correct Root Causes of Problems

Chapter 5

Agile Development Models

As it was described in the last chapter, there are several available Agile development models. Most of them are based in an incremental and iterative development(IID). In this chapter, it will be done a review of those development models, that were consider more prominent and relevant for this global analysis part. It is relevant to mention that all of these models, in contrast with the previous ones, are in constant evolution and improvement, much because of their relatively short existence.

5.1 Scrum

Scrum is software development process that was the result of the collaboration between several individuals. The first approach was described by Hirotaka Takeuchi and Lkujiro Nonaka in *The New Product Development Game*¹. The noted that projects using small, cross-functional teams historically produce the best results, and referred to this as the “*rugby approach*“. In 1991, Peter DeGrace and Leslie Stahl in his *Wicked Problems, Righteous Solutions*², referred to this approach as Scrum, referring to a rugby term mentioned by Takeuchi and Nonaka.

In the early 1990s, Ken Schwaber used an approach that led to the implementation of Scrum at his company, **Advanced Development Methods (ADM)**³. At the same time, Jeff Sutherland developed a similar approach at Easel Corporation⁴, and was the first to call it Scrum. In 1996, Sutherland and Schwaber

¹[Takeuchi and Nonaka(1986)]

²[DeGrace and Stahl(1991)]

³responsible for build and sold process management software in the early 1990s

⁴a Burlington, Massachusetts software tools company founded in 1981. It was acquired by VMARK Software in 1995. Easel developed client/server tools based on its ESL technology and its Smalltalk-based ObjectStudio.

jointly presented a paper describing Scrum at OOPSLA '96⁵ in Austin, its first public appearance. Schwaber and Sutherland collaborated during the following years to merge the above writings, their experiences, and industry best practices into what is now known as Scrum. In 2001, the well known *Agile Software Development with SCRUM*⁶ was published.

Scrum is both an iterative and a incremental process. It is iterative in the sense that plan for the work of one iteration, to be improved upon in subsequent iterations. However, it is also incremental because completed work is delivered throughout the project.

5.1.1 Basics⁷

Scrum projects progress through a series of thirty-day iterations called “sprints”. At the start of each sprint, the team determines the amount of work it can accomplish during that sprint. Work is selected from a prioritized list called the “product backlog”. The work the team believes it can complete during the Sprint is moved onto a list called the “Sprint backlog”. A brief daily meeting, the “daily scrum”, is held to allow the team to inspect its progress and to adapt as necessary. Graphically, Scrum can be exposed according to the following figure.

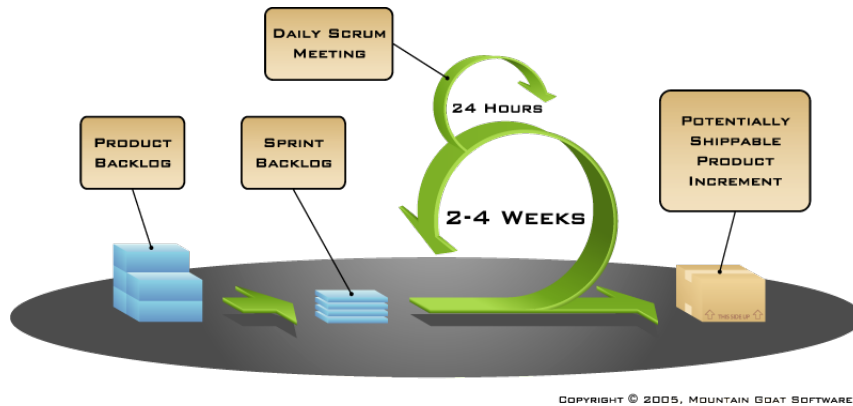


Figure 5.1: Scrum process

⁵OOPSLA (Object-Oriented Programming, Systems, Languages & Applications) is an annual Association for Computing Machinery (ACM) conference

⁶[Schwaber and Beedle(2001)]

⁷[Cohn(2007), Schwaber and Beedle(2001)]

5.1.2 Roles

A Scrum team typically consists of four to seven developers plus or minus two. A Scrum team shares the common “*we’re all in this together*” attitude. So, if there is testing to be done and there is no dedicated tester available, then someone else does the testing. In Scrum teams all work is owned collectively and the team is self-organizing. Because of this, role names such as programmer, architect and tester are generally not used on “*Scrum teams*”. Everything about how a team accomplishes its works is left up to the team.

This core team is supplemented by two key people: the “*Product Owner*” and the “*Scrum Master*”. The Product Owner is largely responsible for placing items onto and prioritizing the product backlog list of needed functionality. The Scrum Master is similar to a project manager except that the role is much more one of leadership than of managing.

5.1.2.1 Product Owner

The Product Owner represents the voice of the customer. He is the only person responsible for managing and controlling the Product Backlog. For commercial development, the Product Owner may be the product manager. For in-house development efforts, the Product Owner could be the project manager. This is the person who is officially responsible for the project. This person maintains the Product Backlog, adding requirements (user-stories), prioritizing them and ensuring that they are visible for everyone. Otherwise, multiple conflicting lists flourish and the Scrum teams do not know which list to listen to. Still, this visibility requires the Product Owner to do his best, and makes this role both a demanding and a rewarding one.

5.1.2.2 ScrumMaster

The ScrumMaster is responsible for ensuring that Scrum values, practices, and rules are enacted and enforced. The ScrumMaster is the driving force behind all of the Scrum practices. He is the one that sets them and makes them happen. The roles of the ScrumMaster in the team can be resumed in the following topics:

- Represents management and the team to each team member reports. He compares what progress has been made to what progress is expected, based on Sprint goals and predictions made during the previous Daily Scrum
- Works with the customers and management to identify and institute a Product Owner

- Works with the Product Owner and the Scrum teams to create a Product Backlog for a sprint
- Works with the Scrum teams to plan and initiate the sprint
- Responsible, during the sprint, for the conduction of the Daily Scrums, and is responsible for ensuring that impediments are promptly removed and decisions are promptly made
- Responsible for working with the management to gauge progress and reduce backlog

The role of ScrumMaster is very often assumed by the Team Leader, Project Leader or Project Manager. It is important to mention that with inexperienced Scrum teams this role should be filled with an experienced (senior) manager or a Scrum consultant.

A suitable ScrumMaster should be active, make decisions, remove impediments or causes them to be removed as soon as possible, in order to provide to the team all the conditions to work at the highest possible level of productivity. Some people are not appropriate as ScrumMasters. They are not comfortable being that visible and taking that much initiative. Removing impediments requires determination and stubbornness.

5.1.3 Practices

5.1.3.1 Product Backlog

The product backlog is the master list of all the functionality desired in the product. When a project is initiated there is no comprehensive effort to write down all foreseeable features. Typically, the Product Owner and the team write down everything obvious, which is almost always more than enough for a first sprint. The product backlog is then allowed to grow and change as more is learned about the product and its customers.

Product Backlog is dynamic and sorted by priority. Top priority Product Backlog drives immediate development activities. The higher a backlog item's priority, the more urgent it is, the more it has been thought about, and the more consensus there is regarding its value. Higher priority backlog is clearer and has more detailed specification than lower priority backlog.

5.1.3.2 Sprint Planning Meeting

A “*Sprint Planning Meeting*” is held at the start of each sprint. The meeting usually lasts up to a full day and is attended by the Product Owner, the Scrum

Master and the entire team of developers. Also attending may be any interested and appropriate management or customer representatives.

During the first half of the Sprint planning meeting, the Product Owner describes the highest priority remaining features to the team. The team asks enough questions that during the second half of the meeting they can determine which items they will move from the product backlog to the sprint backlog.

The Product Owner does not have to describe every item being tracked on the product backlog. Depending on the size of the backlog and the velocity of the team it may be sufficient to describe only the higher priority items, saving the discussion of the lower ones for the next “*Sprint Planning Meeting*”. Typically, the Scrum team will provide guidance when they start to get further into the backlog list than they know can be done in the upcoming sprint.

Collectively the team and the product owner define a Sprint goal, which is a short description of what the team plans to achieve during that sprint. In a “*Sprint Review Meeting*” held at the end of the sprint, the team will assess their success against the sprint goal, rather than against each specific item selected from the product backlog.

During the second half of the Sprint planning meeting, the team meets separately to discuss what they heard and decide how much they heard and decide how much they can commit to during the coming sprint. Conceptually, the team starts at the top of the prioritized product backlog list and draws a line after the lowest of the high prioritized product backlog list and draws a line after the lowest of the high priority tasks they feel they can complete. Still, it is possible to introduce some variations of this.

Once the sprint starts, only the team may add work to the sprint. The CEO cannot approach the team and ask for something the Product Owner left out. Nobody can appear and ask more features for a special customer. And the Product Owner cannot change his mind and ask for additional features. The only time work may be added to a sprint is if the team finds itself ahead of schedule. At that time the team can ask the Product Owner to help identify additional items.

If something so significant happens that the organization feels it needs to change the sprint, then the current sprint is aborted and a new sprint is started by beginning with another Sprint Planning Meeting.

5.1.3.3 The Sprint Review Meeting

Each sprint is required to deliver a potentially product increment. This means that at the end of each month-long sprint the team has produced a coded, tested and usable piece of software. At the end of each sprint, a Sprint Review Meeting

is held. During this meeting the team shows what they accomplished during the sprint. Typically this takes the form of a demo of the new features.

The Sprint Review Meeting is intentionally kept very informal, typically with rules forbidding the use of presentation slides and allowing no more than two hours of preparation time for the meeting. A Sprint Review Meeting should not become a distraction for the team; rather, it should be a natural result of the sprint.

The entire team as well as the Product Owner and ScrumMaster participate in the Sprint Review Meeting. Others who are interested in the project (such as management, customers or developers from other projects) may attend the sprint review if they desire.

During the Sprint review meeting the project is assessed against the sprint goal that was previously determined during the Sprint Planning Meeting.

5.1.3.4 The Daily Scrum Meeting

The Daily Scrum Meeting is typically held as early as possible each day, but after the entire team has arrived for work. After that, everyone on the team is required to attend, including developers, the Product Owner and the ScrumMaster. The meeting is kept short: usually 15minutes or less and never longer than 30. To keep the meeting short, some teams required participants to stand.

During the daily scrum each team member answers the following three questions:

- **What did you do yesterday?**
- **What will you do today?**
- **What obstacles are in your way?**

One of the main goals of the meeting is to have each developer make commitments in front of his or her peers. Commitments are not made to the manager or the organization but to each other.

The daily scrum should stick as closely as possible to the three questions above, and should not move off into designing parts of the system or resolving issues that were raised. Such items are noted during the meeting but are resolved afterwards.

The daily scrum provides everyone on the team a quick daily snapshot of where things stand on the project. Another benefit to holding daily meetings is that they can serve as checkpoints for the Product Owner, managers or anyone else interested in the state of the project.

5.1.4 Advantages vs Disadvantages

Advantages

- Productivity increases:
 - Some Scrum teams have recorded a 4x increase in productivity
 - Most improve productivity by 10-20% depending on management commitment
- Continuous improvement
 - Scrum enables continuous, rapid, bottom-up re-engineering
- Leverages the chaos:
 - The product becomes a series of manageable chunks
 - Progress is made, even when requirements are not stable
 - Everything is visible to everyone
 - Team communication improves
 - The team shares successes along the way and at the end
 - Customers see on-time delivery of increments
 - Customers obtain frequent feedback on how the product actually works
 - A relationship with the customer develops, trust builds, and knowledge grows
 - A culture is created where everyone expects the project to succeed
- Straightforward to learn and easy to follow

Disadvantages

- Requires hand-on management, but not micromanagement
 - Management must be willing to make changes to help Scrum teams succeed
 - Scrum requires constant monitoring both quantitatively and qualitatively
- Requires management to delegate decision-making authority to the Scrum team

- Managers must let Scrum teams make their own decisions, even allowing them to fail if necessary
- Scrum is new and different
 - People are resistant to change
 - Some workers are not comfortable with the responsibility that Scrum enables

5.2 Extreme Programming (XP)⁸

Extreme Programming (XP) is one of the more well known agile methodologies, based in an incremental and iterative development(IID). The roots of XP lie in the collaboration of Kent Beck and Ward Cunningham in the late 1980's and during the early 90's, extending their ideas of a software development approach that was both adaptive and people-oriented. Finally, in 1999 with the publication of *Extreme Programming Explained*⁹ and other publications, the Extreme Programming went into some detail, explaining the various aspects of the approach. As his name suggests, it is a programmer-centric process that emphasizes technical practices to promote skillful development through frequent delivery of working software. This development process is based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are, and to tune the practices to their unique situation. Still, much of the XP process is derived from good practices that have been around for a long time.

Extreme Programming (XP) was created in response to problem domains whose requirements change, where customers may not have a firm idea of what the system should do. You may have a system whose functionality is expected to change every few months. In many software environments dynamically changing requirements is the only constant.

⁸[Cohn(2007), Jeffries(2001), Coffin(2006), ASD(2006), Highsmith(2002)]

⁹[Beck(1999)]

5.2.1 Roles

In the XP process, the customer¹⁰ is responsible for writing user stories¹¹, prioritizing stories, and writing and executing tests that demonstrate that stories were developed as expected. The XP customer may be a user of the system being built but that is not necessary. If not a user, the XP customer is often a product manager, project manager or business analyst.

On some projects the customer role may actually be filled by a customer team, which would be composed of multiple highly interested individuals. The customer team will often include testers who assist with creating the acceptance tests. When a project has multiple customers, it is important that they speak with one voice. They can achieve this in many ways, but most commonly by designating one individual on the customer team as the first among equals.

The XP programmer role encompasses a broad range of technical skills. XP projects tend not to draw distinctions between programmers, designers, database administrators, and so on. All programmers are expected to work as a team and to share many responsibilities that might otherwise be assigned to specific individuals on a non-XP project. Almost all processes expect programmers to unit test their code; XP takes this expectation seriously and programmers are expected to develop automated unit tests for everything they write.

Finally, many XP teams benefit from the use of an XP coach and possibly a project manager. These roles are sometimes combined within a single individual. The coach is responsible for monitoring the team's use of the XP practices and gently nudging them back on track when they stray. The project manager is more leader than manager and is responsible for sheltering the team from bureaucracy and removing as many obstacles as possible.

5.2.2 The Twelve Practices

Extreme Programming is characterized by the twelve practices described in Kent Beck's original "*white book*"¹². Any team that might want to implement XP on a project, is highly encouraged to adopt all of the practices. XP's twelve practices are highly synergistic and interdependent. The presentation of XP's practices have both positive and negative aspects.

At one level, they sound like rules - do this, do not do that. However, this

¹⁰equivalent to the Product Owner in Scrum

¹¹user stories serve the same purpose as use cases but are not the same. They are used to create time estimates for the release planning meeting. They are also used instead of a large requirements document. User Stories are written by the customers as things that the system needs to do for them. They are similar to usage scenarios, except that they are not limited to describing a user interface. They are in the format of about three sentences of text written by the customer in the customer's terminology without techno-syntax.

¹²[Beck(1999)]

practices are more like guidelines than rules, guidelines that priable depending on the situation. Still, Ron Jeffries makes the point that the practices also interact, counterbalance, and reinforce each other, such that picking and choosing which to use and which to discard can be tricky. For example, the practice of refactoring is made easier by the practices of pair programming, simple design, collective ownership, continuous integration, and testing.

The twelve practices are not a random collection of good practices or ideas from which an XP team is allowed to pick their favorites. After becoming experienced with XP, the team may choose to drop or alter a practice, but this customization should be delayed until becoming experienced with standard XP.

5.2.2.1 Small Releases

XP projects progress in a series of iterations, each of which is each typically one to three weeks long. Features, as described by user stories, are fully delivered within a single iteration. A team is not allowed to deliver half of a feature. Similarly, the team is not allowed to deliver the full feature but at half the quality. By the end of each iteration, the team is responsible for delivering working, tested code that can immediately be put to use.

At the start of the project the team selects an iteration length that it will use for the duration of the project. The iteration length is typically one or two weeks and never longer than four. The team should select an iteration length that is as short as possible yet that will still deliver tangible value to the business. Every release should be as small as possible, containing the most valuable business requirements. This mirrors two of Tom Gilb's principles of evolutionary deliverable from his book *Principles of Software Engineering Management*¹³:

All large projects are capable of being divided into many useful partial result step's *and* "Evolutionary" steps should be delivered on the principle of the juiciest one next

Small releases provide the sense of accomplishment that is often missing in long projects, as well as more frequent (and more relevant) feedback. On the other hand, this helps the customer to gain confidence in the progress of the project.

Iteration are firm time-boxes. A team cannot reach the planned last day of an iteration and decide they need two more days. The iteration ends on the scheduled day. The amount work the team does (but not the quality of that work) is adjusted to accommodate the iteration.

¹³[Gilb(1988)]

5.2.2.2 The Planning Game

The Planning Game is the XP name for release and iteration planning during which the developers and customer collaboratively make predictions about the future. Prior to initiating planning, the customer has written user stories on note cards and the developers have estimated the cost or magnitude of each story and have written the estimate on the story card.

To start planning, the developers estimate how much work they can complete in an iteration of the length selected for the project. The customer then looks over all of the story cards and selects her top priority stories for inclusion in the first iteration. The customer is allowed to select stories that sum to, but do not exceed the amount of work the developers estimate they can do in the iteration. Once the first iteration is full of work, the customer moves into the selection of more stories for the second and subsequent iterations.

After some number of iterations, the customer decides that enough stories have been placed into iterations that collectively define a release. The release plan almost certainly does not accurately reflect what will be developed and in what order. The release plan is a hypothesis about how development may proceed but it will be updated at the start of each iteration as priorities change, as more becomes known about the rate of progress by the team, and as the developers learn more about the true expected costs of each story.

Prior to the start of each iteration, the team and customer plan the iteration. This involves selecting the highest priority stories that can be completed in the iteration and then identifying the specific tasks necessary to complete the story.

5.2.2.3 Refactoring

Refactoring refers to the restructuring or rewriting code so as to improve the code without changing its external behavior. A method designed for one purpose is changed slightly to handle a special condition. Then, since it already handles that special condition, it is changed again to handle another special condition. And so on until the method becomes too fragile for further modifications.

The XP process advocates constant attention to refactoring. Whenever a programmer makes a change to code that should be refactored, he is required to refactor it. In this way code avoids the slow, sometimes hard to detect decay that eventually results in obsolescence.

Refactoring is one of the techniques used by the XP to replace upfront design. Rather than spending time upfront thinking through a system in advance of coding it, and therefore taking guesses at some aspects of its behavior, XP systems are refactored and kept in a state that perfectly meets known, implemented requirements.

5.2.2.4 Testing

One of the XP major practices is focus on testing. On a XP project the developers write automated unit tests and the customers write acceptance tests, which are often automated either by the customers themselves or with some help from the developers. Many XP developers have really found the benefits of early and frequent testing. Further, traditional resistance by developers to testing has gone down because XP unit tests are typically automated by writing test code that exercises the operational code.

In traditional development tests are written (if they are written at all) after the code. This becomes a problem because once code is written and appears to work, it can be human nature not to push too hard on the code. So, many developers push gently on their code and call it tested. XP changes this and puts the tests right up front a practice called *test-driven development (TDD)*.

In addition to programmer unit testing, customer tests are an important part of XP. For each story, the customer is responsible for defining a series of tests that are used to determine if the story was developed in accordance with the customer's expectations and assumptions. In many ways, these customer-written acceptance tests replace the requirements documents of a waterfall process.

Test-Driven Development (TDD)

In test-drive development, tests are written before the code. Developers follow short (minutes, not hours) cycle of test-code-test-code and so on. They follow a rule that no operational code may be written except in response to a failing test. So, they write a test that fails. The program is run to verify that the test fails. Only then does a programmer write the enough code that makes the program pass the test, then the programmer refactors the code, primarily to improve the readability and remove duplication.

Test-driven development guarantees that code will remain well factored and restable. It also leads to more maintainable code since that code is effectively in a maintenance mode right from the start. TDD is hard, but trying to do TDD on code base that wasn't built using TDD from start, that's really hard.

5.2.2.5 Pair-Programming

One of the XP's more controversial practices is pair programming. Pair programming refers to two programmers sharing one keyboard and one monitor but using their two brains to write code. While one programmer is typing at the keyboard (and mentally thinking a few lines ahead in his code), the second programmer is watching the code developed and thinking more broadly about where the code may lead and what problems it may encounter. Shifting roles

and pairs frequently is important.

While pair programming may sound tremendously inefficient, in 2001, Alistair Cockburn and Laurie Williams have studied it and found that not to be the case. They found that for 15% increase in total programming time, pair programming leads to:

- Better design quality
- Lower defects counts
- Less code is written to solve the same problem
- Problems being solved more quickly
- More people understand each piece of code
- An increase in developer job satisfaction
- Improves the knowledge spread within the group
 - Learning and training

On the other hand, pair programming is exhaustive because it focus the mind of the pair very intensely and should not be done all day. There is also the case of developers that do not feel comfortable with pair programming. So, it is very important to encourage people and provide the right tools, but at the end, let them experiment with it at their own pace.

5.2.2.6 Sustainable Pace

XP teams are encouraged to work at a sustainable pace. The belief is that an XP team moving at a consistent but brisk pace will achieve more over a period of time than will a team working at a pace they cannot sustain over a long period. This does not mean that an XP teams works exactly forty hours each week and then heads for home. It is up to the team to determine their sustained pace, and it is likely to be different for different members of the team.

An XP coach is responsible for monitoring the team for burnout. If the coach senses a team is burning out, he will help the team shift back to working at a sustainable pace.

5.2.2.7 Team Code Ownership

In XP teams all code is owned by everyone. Under this model of team ownership, any pair of developers (in case of using pair programming) can change any code.

In fact, because of the refactoring practice, pairs are expected to change code they did not write.

Individual ownership is practiced to ensure coherent design and to keep all responsibilities of a module balanced. In XP this burden is borne by TDD development. A strong suite of unit tests ensures that changes do not introduce unpredicted side effects.

5.2.2.8 Coding Standards

Because XP collectively own their source code, it is important that they follow a coding standard. The coding standard lays out the main rules and conventions that the team members should follow when writing code. For instance, how will variables and methods be named? how will lines be formatted?

A small and close team may avoid a written and formalized coding standard. However, larger teams will profit from writing down their coding standards but still keeping them as brief and essential as possible.

5.2.2.9 Simple Design

XP teams pursue a goal of having the simplest possible design that delivers the features according to the customer needs. Kent Beck, in 2000, defines four constraints that indicate when a design is the simplest it can be:

- The operational code and the test code fully convey the programmer's intent for the behavior of that code
- There is no duplicated code
- The system uses the least number of classes
- The system uses the least number of methods

5.2.2.10 Metaphor

XP teams develop a common vision of how the program works, which we call the "metaphor". At its best, the metaphor is a simple evocative description of how the program works, such as *"this program works like a hive of bees, going out for pollen and bringing it back to the hive"* as a description for an agent-based information retrieval system.

Sometimes a sufficiently poetic metaphor does not arise. In any case, with or without vivid imagery, XP teams use a common system of names to be sure that everyone understands how the system works and where to look to find the functionality you're looking for, or to find the right place to put the functionality you're about to add.

5.2.2.11 Continuous Integration

XP not only reminds us of the potential for serious integration errors, but provides a revised perspective on practices and tools. In XP teams keep the system fully integrated at all times. For example, a developer completes a small change, he checks the change into the source code repository where a process notices the change and initiates a full build. When the build is complete a suite of automated tests is run. If any of the tests fail, the developer is emailed and told about the failure. Integration problems are fixed one at a time in extremely small batches as soon as they occur. The benefit of this practice can be seen by thinking back on projects where the build process was weekly or less frequently, and usually led to *"integration hell"*, where everything broke and no one knew why.

Infrequent integration leads to serious problems on a software project. First of all, although integration is critical to shipping good working code, the team is not practiced at it, and often it is delegated to people who are not familiar with the whole system. Second, infrequently integrated code is often buggy code. Problems arise in at integration time that are not detected by any of the testing that takes place on a unintegrated system. Third, weak integration process leads to long code freezes. Code freezes mean that you have long time periods when the programmers could be working on important shippable features, but that those features must be held back. This weakens the position of the organization in the market, or with the end users.

5.2.2.12 On-Site Customer

In XP the the customer is expected to sit with and be part of the development team. The customer writes the stories and the acceptance tests and also is on hand to answer questions, from the developers, as soon as they arise.

The on-site customer is essential to successful use of the user story approach because of the many conversations that must occur between the customer and the developers. If the customer is not on site, delays will disrupt the predictable progress of the XP team.

5.2.3 Values and Principles

In addition to its practices, XP advocates four values:

- Communication
- Simplicity
- Feedback

- Courage

XP process values communication, but not all modes of communication are equal. Most desirable is face-to-face communication where it is possible to talk, respond, gesture and draw on a whiteboard. Less desirable are written documents. XP stresses communication through practices such as pair programming.

Simplicity is a value of XP teams because it keeps their focus on creating a solution to the problem faced today, not the problem anticipated tomorrow. XP teams do not architect a system with the support for features beyond those that are necessary for developing the features of the current iteration. They remain constantly focused on doing the simplest thing that could possibly work.

XP teams value feedback, and the more immediately the feedback the better. XP developers get feedback from the automated test that they run so often. They get feedback from their continuous integration process. Customers are part of the team, even sitting in the same place with the developers, and provide feedback through constant interaction with the team and through the acceptance tests they write.

Finally, but not less, XP teams value courage. For example, they have courage to refactor their code and to proceed without an overall master architecture, because they maintain a simple design through refactoring and TDD practices.

So, the principles that come from the practices and values of XP, can be summarized by the following topics:

- Provides rapid feedback to its customers and learns from that feedback
- Favors simplicity and always attempts a simple solution before moving to a more complex one
- Improves the software through small, incremental changes
- Embraces change because they know they are truly adept at accommodating and adapting
- Insists that the software consistently exhibits the highest level of quality workmanship

5.2.4 Advantages vs Disadvantages

Advantages

- Strong technical practices (pair programming, TDD, coding standard, ...)
- Customer ownership of feature priority, developer ownership of estimates

- Frequent feedback opportunities
- Continuous integration
- All the advantages of using an agile software development process

Disadvantages

- Requires on site customer, or at least a collaborative one
- Documentation primarily through verbal communication and code
 - For some teams these are the only artifacts created, others create minimal design and user documentation
- Difficult for new adopters to determine how to accommodate architectural and design concerns

5.2.5 Application

XP is generally most appropriate on smaller, highly dynamic projects. Although many of its practices can provide value when combined with other management processes. XP has also been scaled to companies with hundreds of developers, but handling large projects is a customization that a company has to make - it is not inherent to the XP process due to his intense focus on constant, quick feedback and simplicity.

5.3 Feature Driven Development (FDD)¹⁴

Feature Driven Development (FDD) was developed by Nebulon's Jeff De Luca over his years of extensive IT and project management experience. It incorporates several ideas and practices from the last 30 years of the industry.

FDD is a model-driven short-iteration process. The creation of a domain model is the foundation step of FDD. This requires the collection of domain knowledge from all domain experts (sometimes called **Subject Matter Experts (SMEs)**), and integrating this knowledge into a unified model (or set of models) representing the problem domain. Once this model and any other requirements have been evaluated, a rough plan is drawn up to determine what resources will be needed to build the masterpiece. Meanwhile, a small set of features are identified for a team to work on, for a period of time recommended to last no more than two weeks. Once the initial set of features is delivered, another set is tackled. Multiple teams can work in parallel on different sets of features, with all activity being tracked on a per feature basis.

¹⁴[DeLuca(2008), Palmer(2002), Ltd().]

5.3.1 Roles

FDD relies on specific roles and responsibilities more than almost any other agile development model. FDD also tends to move away from shared ownership of code and artifacts that other agile models promote, and the team roles reflect this. The nine roles defined for FDD are:

- **Project Manager** - responsible for all administrative aspects of the project, including the financial and reporting ones.
- **Chief Architect** - responsible for the overall design of the system, including running all design sessions, code reviews, and technology decisions.
- **Development Manager** - on the hook for the daily development activities. Coordinating the development team and their activities, and dealing with resource issues.
- **Chief Programmer** - a senior developer involved in ongoing design and development activities, and is assigned to be responsible for one or more Feature Sets.
- **Class Owner** - a developer who works under the direction of a Chief Programmer to design, code, test, and document features as they are implemented.
- **Domain Expert** - any business related stakeholder who can define required features that the system should provide. These would include clients, users, and various analysts; anyone who has knowledge of how the business works that could impact the system.
- **Tester** - responsible for verifying that each feature performs as defined.
- **Deployer** - deals with not only the actual deployment of code to various environments, but also the definition and/or conversion of data from one format to another.
- **Technical Writer** - responsible for creating and maintaining all the on line and printed documentation that a user will need for the final system.

5.3.2 Activities

5.3.2.1 Develop an Overall Model

Domain and development members, under the guidance of an experienced component/object modeler (chief architect), work together in this process. Domain members present an initial high-level, highlights-only walk-through of the scope

of the system and its context. The domain and development members produce a skeleton model, the very beginnings of that which is to follow. Then the domain members present more detailed walk-throughs. Each time, the domain and development members work in small sub-teams (with guidance from the chief architect), every sub-team presents the results, which are merged into a common model (again with guidance from the chief architect), adjusting the model shape along the way. In subsequent iterations of this process, smaller teams tackle specialized domain topics. Domain members participate in many, yet not all of those follow-up sessions.

5.3.2.2 Build a Features List

The knowledge that is gathered during the initial modeling is used to identify a list of features. Features can be combined into “*Feature Sets*“, and “*Feature Sets*“ can be aggregated into “*Subject Areas*“. Requirements are usually gathered in a top down approach, defining all the “*Subject Areas*“ for the system, breaking these down into “*Feature Sets*“, and eventually down into “*Features*“, from which tasks can actually be defined and estimated. “*Feature Sets*“ typically reflect a business activity, and “*Subject Areas*“ commonly correspond to general business practices.

The basic unit of work in FDD is a “*Feature*“, which is defined as a small, client-valued function expressed in the form:

<action> the <result> <by | for | of | to> a(n) <object>

where an object is a person, place, or thing (including roles, moments in time or intervals of time, or catalog-entry-like descriptions).

For Example: “*Calculate the total amount of operational power converters in the LHC*“

Features should not take more than two weeks to complete, else they should be broken down into smaller pieces. A small feature is a tiny building block for planning, reporting, and tracking. It’s understandable. It’s measurable. It’s do-able (with several other features) within a two-week increment.

5.3.2.3 Plan by Feature

Using the hierarchical, prioritized, weighted features list, the project manager, the development manager, and the chief programmers establish milestones for “*design by feature, build by feature*” iterations.

5.3.2.4 Design by Feature

A chief programmer takes the next feature, identifies the classes likely to be involved, and contacts the corresponding class owners. This feature team works out a detailed sequence diagram. The class owners write class and method prolongs. The team conducts a design inspection.

5.3.2.5 Build By Feature

Starting with a “*Design by Feature*” package, each class owner builds his methods for the feature. He extends his class-based test cases and performs class-level (unit) testing. The feature team inspects the code, perhaps before unit test, as determined by the chief programmer. Once the code is successfully implemented and inspected, the class owner checks in his class(es) to the configuration management system. When all classes for this feature are checked in, the chief programmer promotes the code to the build process.

5.3.3 Practices

FDD is built around a core set of industry-recognized best practices, derived from software engineering. These practices are all driven from a client-valued feature perspective. It is the combination of these practices and techniques that makes FDD a compelling process. The best practices that make up FDD are shortly described below.

- **Domain Object Modeling** - consists of a exploration and explanation of the domain of the problem to be solved, resulting in a framework within which to add features.
- **Developing by Feature** - any function that is too complex to be implemented within two weeks is further decomposed into smaller, client-valued functions, until each sub-problem is small enough to be called a feature. This makes it easier to deliver correct functions and to extend or modify the system.
- **Individual Class (Code) Ownership** - having a single person who is responsible for the consistency, performance, and conceptual integrity of each class.
- **Feature Teams** - doing design activities in small, dynamically formed teams so that multiple minds are always applied to each design decision, and multiple design options are always evaluated before one is chosen.

- **Inspections** - applying the best-known defect-detection technique and leveraging the opportunities it provides to propagate good practice, conventions, and development culture.
- **Version Control** - identifying the latest versions of completed source code files and providing historical tracking of all information artifacts in the project.
- **Regular Builds** - ensuring that there is always a demonstrable system available, and flushing out any integration issues that manage to get past the design and code inspections. Regular builds provide a known baseline to add more function and against which a quality assurance team can test.
- **Progress Reporting** - frequent, appropriate, and accurate progress reporting at all levels, inside and outside the project, based on completed work.

5.3.4 Advantages vs Disadvantages

Advantages

- Supports multiple teams working in parallel
- Focus developers on producing working results every two weeks
- Provides detailed planning and measurement guidance
- Design by feature and build by feature aspects are easy to understand and adopt
- Track and reports progress with surprising accuracy
- Scales to large teams of projects well

Disadvantages

- Promotes individual code ownership as opposed to shared/team ownership. This might be a disadvantage with teams that change very often
- Iterations are not as well defined by the process as other Agile models
- The model-centric aspects can have huge impacts when working on existing systems that have no models

5.3.5 Application

FDD is a process for helping teams produce frequent, tangible working results every two weeks. On the other hand, FDD includes planning strategies, and tracks progress with precision. It can function as a complete Agile process, or can be combined with Scrum, Lean or XP to produce a customized integration of techniques.

5.4 Some Statistics

There are several available statistics about the state of agile software development. One of the more relevant studies about this issue is the “*State of Agile Development*” survey conducted by VersionOne¹⁵. This global survey ran in June and July of 2007 and was completed by almost 1700 individuals in 71 countries. The conclusions presented are really interesting, however they will be highlighted the following observations:

- 90% of the individuals say that, with the adoption of a agile process, they experienced, in fact, an productivity increase, and 85% also argue that it allowed to reduce software defects
- The processes that gather more attention are **Scrum (37%)**, followed by **Scrum with XP Hybrid (23%)** and **XP (12%)**
 - The Custom solutions gathers 9.2% of the replies
- The major concerns about the adoption of Agile development are:
 - Lack of Up-Front Planning (36%)
 - Lack of Documentation (30%)
 - Loss of Management Control and Lack of Predictability (26%)
 - Lack of Engineering Discipline (24%)
- The major barriers to further adoption of Agile in the organizations are:
 - General resistant to change (36%)
 - Personnel with Agile experience (34%)
- 81.5% are currently trying to implement Agile in their organizations, or at least thinking about doing so

¹⁵[[ASD\(2007\)](#)]

Iteration Planning	65%
Unit Testing	60%
Daily Stand up	55%
Release Planning	54%
Continuous Integration	54%
Refactoring	46%
Retrospectives	39%
TDD	38%

Table 5.2: Which practices are more used within a Agile process

Wiki	Currently Use	Plan to Use	Do Not Need
Index Cards	68%	10%	10%
Spreadsheet	46%	6%	37%
Bug Tracker	74%	3%	21%
Traditional Proj. Manag. Tool	81%	7%	7%
Agile Proj. Manag. Tool	43%	14%	13%
Unit Test Tool	72%	13%	3%
Acceptance Test Tool	36%	22%	7%
Refactoring Tool	41%	13%	12%
Continuous Integration Tool	57%	17%	6%

Table 5.3: Agile tools currently used or planned to be used in the next 6-12 months

5.5 Some Considerations

In this chapter was done a evaluation of some software development models, their context, practices, values, advantages, disadvantages, applications and other relevant subjects. However, this evaluation wasn't random at all. In fact, it was done considering the context of the AP (will be analyzed in the following chapter) and the potential interest in those models.

On the other hand, the description of models like the Waterfall and Incremental was done, not because of the practical value as a model to be adopted,

but to show that software engineering has been following a path search for an effective “*silver bullet*“. Considering this logic, other models might have been described as well, although, in an analysis like this, it is imperative to do choices. There are several other models that would be interesting to consider. Yet, they probably would not add enough valuable information, to this particular context, in order to justify that.

Chapter 6

The AP from the Scratch

6.1 The Section

The AP is a melting pot of cultures. In fact, this section as people from almost every country of Europe (Spain, France, Italy, Switzerland, Germany, UK, Sweden, Poland, Greece, Russia, ...) and even abroad. This multicultural scenario brings a set of unique characteristics.

In every organization, the team elements have their unique personalities, with different backgrounds, feelings, speculations and ambitions. However, in a place with so many cultures like AP, these differences increase a lot. From the experience of the author in his stay at AP in the last year, it is possible to notice that the methods of working, the ways of dealing with problems differ a lot according to the person. Each person has his own style, strengths and weaknesses which is a constant but motivating challenge. It is also very important to highlight that the elements from AP are very united. They are a real team, where exists respect between each others and where everyone is focused and committed in a common and ambitious goal.

6.1.1 Organization

The organization and size of the development teams in AP depends a lot from the context, size and demands of the project. However, in general the teams are quiet cross functional and they could have from 2 developers until 6 or 7. Normally with different degrees of experience and expertise.

In every team committed to a project, there is a project leader. In general, the project leader is the team most experienced developer, having the responsibility of the main decisions in the project. Every team element reports to his project leader or supervisor. On the other hand, the project leaders reports

directly to the section leader, which is the head of the AP. Still, there is not a rigid hierarchy, by the contrary. Everyone has the freedom to propose and discuss ideas with each person in the organization.

6.1.2 Staff vs Temporary Personnel

There are two types of personnel, the staff and the temporary ones. The staff are people with undetermined time contract and, most of them, already work here for more than 5 years. They are experienced developers with a profound AP domain knowledge. In fact, most of them are project leaders, or at least they have a considerable amount of responsibilities.

On the other hand, there are the temporary personnel. In this category, we have different groups. In the *“bottom of the chain”* we have the technical students (the author case), which are personnel that must be registered at a university or higher technical school throughout the entire duration of their stay at CERN. The work completed here (CERN) is often part of the student’s final year studies. The contract duration of a technical student can be a maximum of 14 months. After that, there are the project associates and fellows, which are personnel with, in theory, a higher amount of responsibility. On the other hand, these positions are normally filled by people with a reasonable degree of experience. Their contract can have an extension of 2 years.

This personnel policy has the advantage of providing to young and inexperienced developers, the opportunity of working in a multicultural development environment like this one, with valuable experienced developers knowledge and with challenging projects. On the other hand, in the authors opinion, this has also a big disadvantage. The integration of new elements into a project takes time. More than time, it takes effort from the new element, but also patience from the team that integrates that element. In the case, for instance, of the technical students, when this new developer starts to become really productive and an asset to the team, it arrives the time for departing. This has the inconvenient of being always teaching new developers, most of the times from the scratch. Meanwhile, there is also the issue of losing knowledge by the team when the developer departs which, for instance, will be the case of the author.

6.1.3 Qualification

All of AP personnel has a academic degree, that could go from 3 or 4 years of high studies, until a PhD. Although if we take a look to the composition of AP, it is clear that most of the elements are from the field of Computer Science, and few come from Software Engineering. This might look the same, but their are

fundamental differences between the two disciplines, which might be interesting to explore.

According to the *Britannica Encyclopedia*¹, computer science is:

Study of computers, their design, and their uses for computation, data processing, and systems control, including design and development of computer hardware and software, and programming. The field encompasses theory, mathematical activities such as design and analysis of algorithms, performance studies of systems and their components, and estimation of reliability and availability of systems by probabilistic techniques.

On the other hand, if we remind what was said previously²:

Software engineering is the discipline of software process, development, methods and tools. It comprises standard specification, design techniques, formal analysis, established processes, architecture, and more. It addresses practical constraints and involves mathematics, management, psychology, and economics in addition to computer science.

So, it is clear a difference between both fields. In AP this difference does not look to exist in the sense that software engineering is very often confused with computer science. In fact, very few people seem to understand that different. The author would say that software engineering does not have much of space in AP. This does not mean that people do not show interest in this area when there are initiatives to promote it. Although, it is crucial for a organization (as it was described in previous chapters) like AP to focus on software engineering. This does not mean that computer science isn't important. By the contrary, it is, but software engineering is also important, or even more important, for a software development organization.

6.1.4 Working Conditions

The AP stands in the building 936 at the CERN Preessin site. This means that all the teams work in the same physical space. The building is small with only one floor and has a kind of central hall, which increases the chances of a straight contact between the personnel. In general, the project leaders have their own office, and the other developers share an office with one or two others more. This distribution was done having in mind the developers teams composition

¹[[APA\(2008\)](#)]

² [3.1](#) on page [18](#)

by project. Yet, this does not happen in all cases, which means that there are situations where developers from different projects share the same space. However, this is thanks to the 936 space limitations, considering the more than 30 developers that work there.

This description may sound unuseful, but it is not. This intends to show that the AP personnel has a straight contact, which is important for the communication of the team, and this has impact into the software development. On the other hand, this building is in front of the CCC, home of the OP, which means that there is also a close contact with the AP clients.

6.2 Control System

As it was mentioned in the contextualization, it was said that AP was responsible for supporting the current accelerators control system. However, the question is, do we really have a control system? In the authors opinion, objectively, no. In relation to this, the English dictionary says that a control system is:

Interconnections of components forming system configurations
which will provide a desired system response as time progresses

This does not really happen in AP, and in CO as well. It does not happen in the sense that, what is called the control system, is not in fact a homogeneous and unique system of interconnected components, but actually a set of collaborating, but also competing applications. Collaborating in the sense that they work together. Competing because sometimes they overlay the features of others, which means that they do pretty much the same by different ways.

On the other side, applications developed at AP have different designs, some of them can be called vertical applications in the sense that they cross a set of abstraction layers (p.ex. LASER), which is not the case of others (p.ex. SIS) that are built on the top of established layers of software provided by other projects. The author believes that, from the very beginning, there was not a common strategy behind the system conception. This might have several explanations. One of them could have been the demands of a control system (as soon as possible) to accommodate the quick CERN accelerators complex development. The pressure on that created a system without a common view and strong quality guarantees. However, it is somehow understandable that the business domain behind all this, is in fact extremely complex, and besides that, the control system had to deal with legacy applications and devices with several years³, which increased greatly the situation described.

³do not forget that CERN operates since the 1950s

According to statistics done by AP, the actual amount of lines of code is more a less around 5 million.

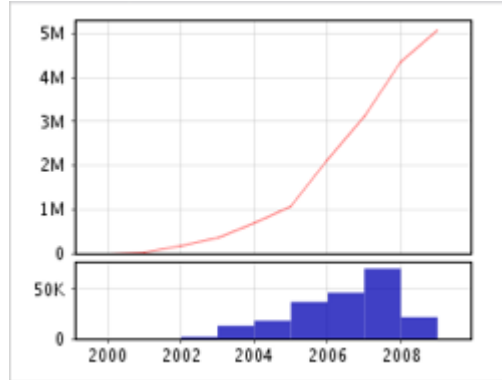


Figure 6.1: AP lines of code

This is huge, and also a problem. In fact, with the LHC commissioning start-up, it is spectacle that the amount of new features development, and new projects as well, will decrease. The question that comes to light is: do we have a software development process and practices to respond to the challenge of maintaining and extending, efficiently, these 5 million lines of code (thousands of them with low quality) from the different projects? Do we have tools, procedures or practices to evaluate the quality and predict the bugs of that code?

6.3 Software Development Process

In AP each project as his own particular context. However, all the projects are developed for the mutual accelerators controls goal, which gives a number of common similarities between them. Concerning the development process of those projects, and considering the activities and practices of most of the general development processes (models) described in previous chapters, it is possible to say that none of those projects follows a well defined model. In fact, in some cases, the notion of development process as a process, or does not exist at all, or is quiet dubious (by a optimistic perspective). Still, before judging these matters, it will be done a evaluation of the roles of each character involved in the development of software at AP, and the different used practices as well.

6.3.1 Roles

6.3.1.1 Client

The word “*customer*” was avoided by the fact that this might suggest that AP develops software for a outside client, getting profit from that, which is not the case. In fact, since AP develops software for the accelerators controls system (in-house), this means that the client is inside the same organization (CERN CO). So, the author considers more suitable to use the word “client”.

The clients from AP projects are, in almost all cases and independently from the project, from the OP, as it was referred in the contextualization part. So, thanks to their experience, AP project developers have a pretty clear notion of the OP objectives, necessities and expectations. This is a clear advantage in the software development business. Still, not all the OP members are the same. In fact, each one of them has its own style, personality, behavior and availability. On other side, a OP member is, in most of the cases, someone that knows very well the domain of the project in which he is involved. This does not mean that he always knows what he wants, or what he can ask for.

Another important topic is the client availability and commitment into a project. These also changes quite a lot according to the scope, objective, risk and scheduling of the project. In some projects the schedules are more tight, or the demands of the client are more difficult to understand, which reflects into different client commitment degrees. Still, there is always involvement in the sense that the client is present when the requirements are gathered and, in general, he is available to understand the difficulties that might appear during the development. On the other hand, it is important to say that a client can be only one person, or a group of persons with the same interests. Still, in most of the cases he has only one voice in what concerns the AP side.

6.3.1.2 Project Leader

As it was referred, all the projects have a project leader that reports directly to AP leader. The project leader works like a project coordinator. It is someone that manages the development team, assumes the main technological decisions, and also contributes greatly to the coding process. All the team developers report the state of their work to him. However, this changes a lot depending on the project development team. In some teams this report is done every day, but in others, which is in most of the cases, it is done time to time. This behavior is related by the use of different development practices among the teams, which will be described and analyzed ahead.

6.3.1.3 Section Leader

The section leader⁴ works like a major coordinator of the section. His responsibilities are focus on keeping the AP projects on rails and defending the section good name in CO group and abroad. He also keeps the teams informed about the developments outside AP, and takes the responsibilities of the main decisions taken in AP.

The section leader could be seen as the project manager inside AP , which also has the straight involvement of the CO group leader.

6.3.2 Practices and Considerations

6.3.2.1 Domain Analysis

This topic is critical in AP. On the context of controls software the knowledge of the domain is crucial, not only to understand the objectives and roles of the section, but also to be integrated into a project context. Considering this topic, the newer comers deserve a special mention. When they arrive to AP development teams, they have several difficulties to pass through. The major ones are the understanding of how the accelerators control system works, and what is the role of each project in that context. The author considers that AP has serious gaps concerning this integration. The available information or documentation about the control system, and from each and every project is very few, as well as the time from the older personnel to promote that instruction. In fact, when someone arrives, he is basically thrown to the arena without the necessary knowledge.

For the rest of the team, there is always the necessity to keep the knowledge of the domain up to date. In fact, there are contexts that change very often, and the LHC project constant demands requires a continuous attention and information by AP project developers. The lack of basic documentation is for sure an obstacle to reach this goal.

6.3.2.2 Incremental and Iterative

AP projects have different degrees of development. In fact, there are projects with some years of development time (more than 5 years), and others with less than one year. This is a challenge in the sense that besides different contexts, the projects also have different degrees of maturation. However, it is relatively clear that all of them try to follow an incremental and iterative development philosophy. This does not mean that they actually follow the same software

⁴Eugenia Hatziangeli

development model, or even if they have one. In fact, the author would say that in all projects the IID practices are not clear at all, looking more like intentions.

6.3.2.3 Requirements Analysis and Specification

There is not a clear requirements specification process with the production of basic documents in AP, independently of the project. The author would say that, in general, the requirements are discussed with the client and are added to a features list (written somewhere in note paper). In most of the cases, when involved, the CO clients define most of the important features to implement, without the notion of iteration. Still, almost none documentation is created, even if the project is giving his first steps. The pressure to have working software as soon as possible, in general, creates situations where the documentation is skipped at all and the requirements are transmitted in simple oral chats with the client.

6.3.2.4 Small Releases

Most of the projects have short time and small releases. Although, the author considers that in none of them there is a fixed time-box. In fact, depending of the maturation degree of the project, the client needs and the control system demands, the time-box changes. It is very often that the pressure of the client defines the end of the iteration⁵ and the preparation of a release, even if the defined features for that iteration are not completed at all.

6.3.2.5 Continuous Integration

In almost all projects it is common to build the product at least in a week base. This is critical in controls software, and AP teams have a huge experience on that. When a product is integrated, the team needs to have certain guaranties that everything will work smoothly, and a way to improve those guaranties is to promote continuous integration practices.

6.3.2.6 On-Site Client

In all the projects exists a close relation between the client and the project team. Of course this relation of cooperation changes according to the project, and obviously according to the client. Still, in general, the client is available and might be present in the development of the software. On the other hand, this reminds situations of tests that were conducted in the CCC, and for which it is often necessary to add new features to products, as well as some preparation

⁵the term iteration in this context might have been used in an abusive way

(tests and so on) and planning that did not happen at all. In that times it is common to see project clients or even OP elements, that do not own any specific product, arriving to AP teams asking for new features or even new products for the following week (or even for the same!!!). This brings a huge amount of instability to the organization.

6.3.2.7 Individual Code Ownership

Individual code ownership is practiced to ensure coherent design and to keep all responsibilities of a module or functionality balanced. On the other hand, this creates situations of dependency in the sense that only one developer really knows what is there. In places where the team owns the code, the knowledge is more spread and everyone would be able to work and change pieces of code that were not coded by them from the beginning. The individual code ownership practice is not used in all projects. Although, it is the natural choice. The result of this is a different set of practices and code standards. Of course there are other explanations for that as well. For instance, the AP does not have a single code standard to define and regulate the way how Java code is created.

6.3.2.8 Documentation

There is no basic documentation of the main projects foundations. The most important features, design, main technological decisions are not documented at all. Of course there are some few documents with some explanations and UML models for instance, but totally disconnect between each others and most of them are not updated since their creation. There is no common way of developing documentation integrated with the software development.

Concerning the code, main projects start to have some “*Javadoc*”⁶, but there are thousands of lines of code without this concern. The **Java API for Parameter Control (JAPC)** project might be a good example of documented code. However, this is more an exception than a rule.

On the other hand, there are not any kind of manuals, or introductory documents for those that arrive to the section. This is a serious gap. For instance, in all of his projects, the AP develops **Graphical User Interfaces (GUIs)** however, there is no manual to define principles and standards for this. The result is a wide range of GUIs with different looks and philosophies.

⁶is a documentation generator from Sun Microsystems (Sun Microsystems www.sun.com) for generating API documentation in HTML format from Java source code

6.3.3 Tools

There are several available tools in AP, but are they really used by the developers? The answer to this question has two sides. In first place, most of the available tools are not in fact used by the majority of the developers. On the other hand, if some of them are used, like JIRA, they are not integrated in a consistent development process, which takes off much of the tool purpose.

6.3.3.1 JIRA

JIRA⁷ is a issue and bug tracking, and project management application developed to make this process easier and more transparent to a development team. It is used by more than 5.500 organizations, such like, NASA, 3M, CNET Networks, MIT, Nokia, Oracle, SAP, Shell, BMW, Accenture, European Parliament, ONU, World Bank and many others.

CERN is one of his clients as well. The AP has JIRA tool available from 2/3 years, although, it is not used by all the developers, by the contrary. In the author's opinion, there is plenty of space to integrate this tool, more suitably, into a common development process, in order to keep tracking the features/issues life cycle of each product.

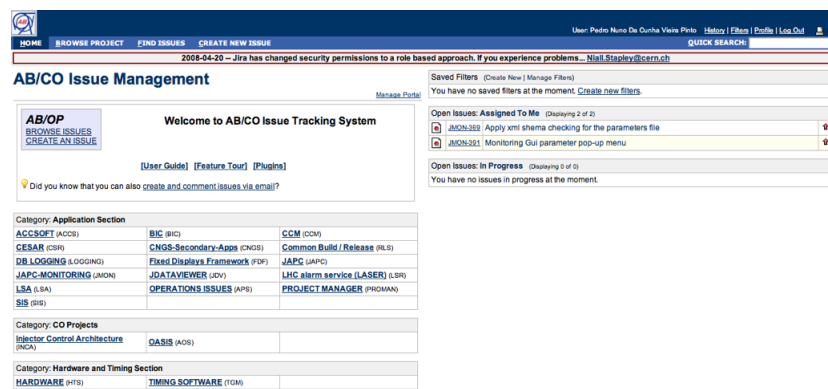


Figure 6.2: JIRA at CO-AP

6.3.3.2 Wikis

Wikis are software that allows users to collaboratively create, edit, link and share knowledge between a community of users. There are several types of wikis with different approaches and functionalities. Still, the main principle

⁷JIRA <http://www.atlassian.com/software/jira/>

behind is really sharing knowledge. The AP saw the potential of these tools, and implemented their use in order to document the major features of a project, in a sort of user manuals, but also to describe some technical details of each project. The idea of this is to share valuable knowledge amount the developers, but also to be seen by the client of each project.

In the author's view, the potential of these tools is being under used for a number of reasons. First of all, not all the developers use this. Second, most of the clients are not aware of the existence of these tools, so they cannot profit from their information. On the other hand, its common that some of the information available is not updated for a number of time, which could mean that it is not useful anymore.

However, it is notorious in the last year an contents quality improvement and an developers utilization increase, which revels, at least, more will to use the wikis and more sensibility to their value.

6.3.3.3 Technical Meetings (TMs)

The concept behind the technical meetings (TMs) is to discuss matters related to each project, in a perspective of presenting solutions that might be useful for other developers in other projects. On the other hand, it is a way to present difficulties that other developers may already had in the past. This initiative was started half a year ago, and in a global view, the results are quiet positive. Almost all the section attends to these meetings and is becoming an important way to share knowledge. Still, it is common to notice that the participation of the developers it is not homogeneous. In fact, most of the contributions come from the most experienced developers, and it is often to see the youngest ones with a passive position.

These meetings are done every Fridays, when they is not section meeting⁸ (SM), and after that, all the personnel gathers into the central hall to share a cake and to “*chillout*” a bit. This promotes heavily the contact and the straight relation between the developers.

6.3.3.4 Software Metrics

Concerning this topic, the author's answer is quite simple, there are no software metrics in AP. There are intentions from some few elements, but that is all.

⁸done once a month.

6.4 Evaluation with CMMI

During the elaboration of this master thesis dissertation the author spoke with some AP elements. In one of those discussions it was used the CMMI model to evaluate the state of the “AP software development process“. The conclusion was quiet consensual. The AP section fits the **level 1 - Initial** of the CMMI mode. If we recover the description of this level ⁹:

Processes are usually ad hoc and chaotic. The organization usually does not provide a stable environment for developing and maintaining software. Success in these organizations depends on the competence and heroics of the people in the organization and not on the use of proven processes. In spite of this ad hoc, chaotic environment, maturity level 1 organizations often produce products and services that work; however, they frequently exceed the budget and schedule of their projects. Maturity level 1 organizations are characterized by a tendency to over commit, abandon processes in the time of crisis, and not be able to repeat their past successes. Success depends on having an exceptional manager and a seasoned and effective software team, so at level 1, capability is a characteristic of the individuals, not of the organization.

Considering the previous description and comparing it with the AP analysis, the author believes that there are undeniable similarities. In fact, the author believes that AP success gives much to the strong commitment and skills of some great developers, in a organization that does not have a established software development process.

⁹ 1 on page 43

Part III

Proposal

*In this, the idea is that we interpret the input from our senses in
terms of a simple and elegant model*

Karl Popper (1902 - 1994)

Chapter 7

Looking Forward

The analysis of the aforementioned development models, and obviously the AP analyze as well, allowed to identify several situations. First of all, considering what was analyzed, and looking to the author's experience, it is possible to say that AP has a set of strong characteristics suitable to implement an Agile development process, such like:

- Experienced, creative and skilled developers
- Extremely multicultural place
- Small development teams
- Developers with a strong and informal contact between each others
 - They even share the same space (one floor building)
- Communication is always present
- Straight contact and involvement with the clients
 - Rapid changes in the requirements

However, the AP analysis demonstrated other things as well. AP develops software that some how could be consider critical software. The expectations would say that this requires, for instance, software with a certain degree of quality or available documentation. Still, this is not a general rule in AP. This means that AP also needs a process capable to assure some rigid, but necessary practices. A process capable to give shape to AP organization. Of course that Agile processes have practices, capable of giving some guaranties in these matters as well, still, it is not in their own nature to do it (a good example is the documentation practices).

As it was described, Agile is about people. People are the core of every organization, and the AP knows this very well. For these and other reasons, and in other to conciliate this preliminary observations, the author proposes a hybrid process. A process that will use the CMMI structure mixed with an Agile model, and some other Agile practices, in order to profit from both approaches. This may look like a paradox still, it will be proven that it is not the case.

Considering the analyzed Agile development processes, obviously the description of Scrum and XP models were not random at all. The FDD was also analyzed for a specific purpose, that will be explored during the explanation of this proposal. In this proposal the author will try to accommodate the Scrum development model and XP practices and principles into a CMMI advice structure¹. This will be a challenging and perhaps controversial proposal. The author does not look for a silver bullet. Instead, the idea is to integrate the AP nature, culture, history and particularities into a software development process capable to bring discipline to the organization and effectively develop better software.

7.1 Agile CMMI

7.1.1 Why Adopting CMMI?

The proposal idea passes by using CMMI to help AP to institutionalize Agile Processes². In software development it is still common to hear that Agile Processes are a sort of disguise for undisciplined hacking and of some individuals who claim to be Agile just because they “don’t document”. The author disagrees with this idea, still he believes that the real value from Agile Processes can only be obtained through disciplined use. CMMI has a concept of “*institutionalization*” that can help establish this needed discipline.

“*Institutionalization*” is defined in CMMI as “*the ingrained way of doing business that an organization follows routinely as part of its corporate culture*”. Others have described “*institutionalization*” as simply “*this is the way we do things around here*”. This concept is an organizational-level concept that supports multiple projects. CMMI supports it through the Generic Practices (GP) associated with all process areas³.

As it was aforementioned in the CMMI analysis, there exist five maturity levels⁴. This proposal does not pretend to implement a CMMI level 5 in AP. This would not be realistic. Although, pretends to create structures and release

¹not to forget that CMMI model mostly deals with what processes or practices should be implemented, and not so much with how they can be implemented

²[Sutherland et al.(2007)Sutherland, Jakobsen, and Johnson]

³ 4.8.4 on page 49

⁴ 4.8.1 on page 43

ideas to build the path in order to, one day, reach that level. So, by a question of wisdom, this proposal will launch an approach to a CMMI implementation at levels 2 and 3. In order to reach this purpose, the author will consider the twelve generic practices (GP) associated with those maturity levels, and how they might help an organization to establish the needed discipline to any Agile Process. However, for these objective proposal it will be the Scrum development model.

7.1.2 CMMI to improve Agile

Establish and maintain an organizational policy (GP 2.1)

The first step toward institutionalization of Agile Processes is to establish how and when they will be used in AP. AP might determine that Agile Processes will be used on all projects or some subset of projects based on size, type of product, technology, client, or other factors. This policy is a way to clearly communicate the organization's intent regarding Agile Processes. In keeping with the Agile Principle of face-to-face, conversions at a AP section meeting might be a good option to communicate the policy.

Establish and maintain the plan (GP 2.2)

This practice can help to ensure that Agile Processes do not degrade into undisciplined hacking. The expectation is that Agile Processes are planned and that a defined process exists and is followed. The defined process should include a sequence of steps capturing the minimum essential information needed to describe what a project really does. The plan would also capture the essential aspects of how the other 10 generic practices are to be implemented in the project. With Scrum, some of this planning is likely to be captured in a “*Product Backlog*” and/or “*Sprint Backlog*”, most likely within a tool as opposed to a document.

Provide adequate resources (GP 2.3)

Every project wants, needs, and expects competent professionals, and AP certainly already have it, as well as, appropriate facilities, and tools⁵. Implementing an activity to explicitly manage these wants and needs has proved useful. In Scrum, for example, these needs may be reviewed and addressed at the “*Sprint Planning Meeting*” and reconsidered when significant changes occur.

Assign responsibility and authority (GP 2.4)

For a project to be successful, clear responsibility and authority need to be defined. Usually this includes a combination of role descriptions and assignments.

⁵AP has funds to acquire new necessary tools

The definitions of these roles identify a level of responsibility and authority. In Scrum this demands an individual or individuals to the roles of “*Product Owner*“, “*ScrumMaster*“, and “*Team*“. Expertise in the “*Team*“ is likely to include a mix of domain experts, system engineers, software engineers, architects, programmers, analysts, QA experts, testers, UI designers, etc. Scrum assigns the team, as a whole, the responsibility for delivering working software. The “*Product Owner*“ is responsible for specifying and prioritizing the work. The “*ScrumMaster*“ is responsible for assuring that the Scrum process is followed. Management is responsible for providing the right expertise to the team.

All of this is possible to be done in AP. The role of “*Product Owner*“ in fact exists in AP and is actually filled by the project leader. Still, his tasks are not exactly as they should be, but this will be developed ahead. On the other hand, AP does not have certified “*ScrumMasters*“. Still, training could be done. Considering that AP has several projects, the temptation is to say that every project leader should have “*ScrumMaster*“ qualifications. However, the author believes that, in order to promote a share of knowledge, and a straight relationship between the projects, this role should be filled by only one developer (this will be developed further).

If we consider the teams qualifications, in fact, AP has human resources able to fill most of the proposed roles. Of course each developer would fill more than one role, which in fact, already happens in a disorganized way.

Train the people to performing Agile Processes (GP 2.5)

The right training can increase the performance of competent professionals and supports introducing new methods into an organization. Institutionalization of Scrum requires consistent training. This practice includes determining the individuals to train, defining the exact training to provide, and performing the needed training. Training can be provided using many different approaches, including programmed instruction, formalized on-the-job training, mentoring, and formal and classroom training. It is important a mechanism definition to ensure that training has occurred and is beneficial.

All of this can be implemented in AP. To implement Scrum it is fundamental to have at least one “*ScrumMaster*“, as it was mentioned in the GP2.4. Still, some practices from Scrum, and in particular from XP (that will be introduced ahead), need extra training. Some would say that people could learn by themselves. In authors opinion this could not be more far from the truth. That kind of logic brings chaos to the organization. A process like Scrum privileges communication, which was for many times referred as a characteristic of Agile development. Implementing a process like this one is an organization initiative. So, training should be a way to pass common practices, ideas and standards. It

is in fact an excellent opportunity that should be explored deeply.

Place designated work products under appropriate level of configuration management (GP 2.6)

In every organization, the purpose of a project is to produce deliverable product(s). AP is not exception. These products should be a collection of a number of intermediate or supporting work products (code, manuals, software systems, build files, etc.). Each of these work products has a value and often goes through a series of steps that increase their value. The concept of configuration management is intended to protect these valuable work products by defining the level of control, for example, version control or baseline control and perhaps multiple levels of baseline control to use within the project.

Identify and involve the relevant stakeholders as planned (GP 2.7)

Involving the client as a relevant stakeholder is a strength of Agile Processes. This practice further identifies the need to ensure that the expected level of stakeholder involvement occurs. So, for each increment, build, or sprint, if the involvement falls short of expectations it is then necessary to communicate to a appropriate level, individual, or group in the organization, to allow for corrective action as corrective action may be beyond the scope of the project team.

In AP, the projects heavily depend on customer's feedback (as it was referred in the previous chapter) and involvement so, this practice it is really crucial. The client must be involved in the development of the product, must provide feedback, must be an active element. This must involve a management commitment as well. In AP this commitment must be assumed by the section leader, within coordination with the CO group leader. This coordination must involve the OP leader as well, in order to raise concern into his member for this issue.

Monitor and control Agile Processes against the plan and take appropriate corrective action (GP 2.8)

This practice involves measuring actual performance against the project's plan and taking corrective action. Direct day-to-day monitoring is a strong feature of the "*Daily Scrum Meeting*", the "*Release Burndown Chart*"⁶ shows how much work remains at the beginning of each sprint, and the "*Sprint Burndown Chart*"⁷ shows total task hours remaining per day. Scrum enhances the effectiveness of the plan by allowing the "*Product Owner*" to inspect and adapt to maximize "*Return on Investment*" (ROI), rather than merely assuring plan accuracy. In AP this makes all sense. Very often the "*Product Owner*" is interested about

⁶ 7.4.3 on page 103

⁷ 7.4.3 on page 103

the progress of the project, and if the investment of time and resources is being justified. This might not have a big impact in small projects (Fixed Display⁸), but in larger ones (LASER, LSA, ...) it makes all sense.

Objectively evaluate adherence to the Agile Processes and address noncompliance (GP 2.9)

This practice is based on having someone, not directly responsible for managing or performing project activities, to evaluate the actual activities of the project. Some organizations implement this practice as both an assurance activity and coaching activity. The coaching concept matches many Agile Processes. The “*ScrumMaster*” has primary responsibility for adherence to Scrum practices, tracking progress, removing impediments, resolving personnel problems, and is usually not engaged in implementation of project tasks. The “*Product Owner*” has primary responsibility for assuring software meets requirements and is high quality. In AP this practice should be the responsibility of the “*ScrumMaster*”, but this will be explored ahead.

Review the activities, status, and results of the Agile Process with higher-level management and resolve issues (GP 2.10)

The purpose of this practice is to ensure that higher-level management has appropriate visibility into the project activities. Agile Processes have a high level of interaction, for example, Scrum has a “*Sprint Planning Meeting*”, “*Daily Scrum Meetings*”, a “*Sprint Review Meeting*”, and a “*Sprint Retrospective Meeting*”. Management needs are supported by transparency of status data produced by the “*Scrum Burndown Chart*”, combined with defect data. Management responsibilities are to:

- Provide strategic vision, business strategy, and resources
- Remove impediments surfaced by Scrum teams that they cannot remove themselves
- Ensure growth and career path of staff
- Challenge the Scrum teams to move beyond mediocrity

This role has a clear target in AP organization, which is the section leader. Concerning this, AP has the lucky to have a great section leader that really knows when and what to do.

⁸Not the same as FDF (Fixed Displays Framework). A Fixed Display is a target product that uses the FDF.

Establish and maintain the description of Agile Processes (GP 3.1)

This practice is a refinement of GP2.2 above. The only real difference is that the description of Agile Projects in this practice is expected to be organization-wide and not unique to a project. The result is that variability in how Agile Projects are performed would be reduced across the organization; and therefore more exchange between projects, people, tools, information and knowledge can be supported.

This is perfectly reachable in a organization like AP by the reasons presented in the previous chapter. Apart from other things, AP teams have a really close contact, and all the section shares a 1st floor building, which clearly contributes to increase the communication and flow of knowledge as well.

Collect the results from using Agile Processes to support future use and improve the organization's approach to Agile Processes (GP 3.2)

This practice supports the goal of learning across projects by collecting the results from individual projects. The Scrum "*Sprint Retrospective Meeting*" could be used as the mechanism for this practice.

7.2 Why Scrum with XP?

All of the previous generic practices have been very useful in organizations to create a path in order to implement other processes. We have seen that a number of these generic practices have at least partial support in Scrum (or even in other Agile Processes). Concerning this, after analyzing several models documentation in this field, the author believes that Scrum model is a excellent choice for working with CMMI. However, there are XP practices that must be consider as well. In fact, Scrum has been employed, in several organizations, successfully as a management wrapper for XP engineering practices. Scrum provides the agile management mechanisms; XP provides the integrated engineering practices that ensure top-quality, focused code. In fact, XP does not have any management practices.

7.2.1 Complementary Practices

Scrum and XP provide complementary practices and rules. They overlap, for instance, at the planning game (XP) and sprint planning (Scrum). Both encourage similar values, minimizing otherwise troublesome disconnects between management and developers. Combined, they provide a structure within which a customer can evolve a software product that best meets his or her needs, and can implement quality functionality incrementally to take advantage of business

opportunities. Following there are several shared practices that facilitate this functionality:

- **Iterations** - all work is done iteratively, with the OP client's being able to steer and direct the project every iteration.
- **Increments** - every iteration produces an increment of the OP client's highest-priority functionality.
- **Emergence** - only that functionality that the client has selected for the next iteration is considered and built. The client does not "pay" for functionality that he or she might not select, and the developers do not have to code, debug, and maintain irrelevant code.
- **Self-organization** - the client says what he or she wants; development determines how much they can develop during an iteration and figures out the tasks to do so.
- **Collaboration** - management and engineering collaborate about how best to build the product, and what the product should do between iterations.

7.2.2 Scrum Roles and Practices

This proposal predicts a complete introduction of the Scrum model major practices. However, there will be presented concrete implementations suggestions for a number of practices.

7.2.2.1 ScrumMaster

The author believes that the person that fills this position should be responsible for all AP projects, doing it in a full time task. This role would have a straight relationship and coordination with the section leader.

The ScrumMaster must be someone with experience, profound AP knowledge and capable to gather the respect and sympathy of the majority of the developers. AP has elements with the profile to fulfill these conditions. However, in authors opinion, there is one person that is in the top of the list, which is Vito Baggiolini. In fact, it is a person with 10 years of CERN, that knows the AP business domain, objectives, background and legacy. Besides that, he is technically very skilled, knows the domain and state of several projects, and gathers the necessary respect and admiration of the developers and the section leader.

Meanwhile, the author would add to this role, the “*Chief Architect*”⁹ activities. On the other hand, the ScrumMaster can also be assumed by the AP leader, leaving more space to the element that would assume the “*Chief Architect*” role, which is not predicted in the standard Scrum model.

7.2.2.2 Product Owner

The Product Owner, as it was referred in the CMMI 2.4 topic, would be filled by the project leader of each project. Still, this role demands a much more disciplined acting way than the one that is done at the moment. The Product Owner should assume the role’s activities proposed by the Scrum model, and his demands in what concerns the insertion, prioritization and maintenance of a list of features (“*Product Backlog*”). The Product Owner will also be responsible for gathering and discussing the requirements with the OP clients in order to create and update the “*Product Backlog*” in each iteration. This assures that the client controls the iteration. The AP has capable projects leaders to do this, as well as a efficient tool to keep the “*Product Backlog*”, which is JIRA.

7.2.2.3 Sprints and Sprint Planning Meetings

The Sprint Planning Meetings should be implemented in all the projects as an essential Scrum practice. However, the duration of the sprints shouldn’t be fixed for all the projects. In fact, since they are projects with different sizes and demands, each project leader should decide what is more convenient for the project. So, the author would say that projects like the Fixed Displays should have, for instance, sprints of 15 days. On the other hand, projects like LSA probably will profit with longer sprints, like 30 days¹⁰. Of course this should be discussed with the members of each team. Do not forget, Scrum is about people and communication.

After that, these meetings must be attended by all the team members, ScrumMaster and Product Owner, in the AP meeting room. The section leader should not attend this meeting. On the other hand, members from other projects might profit from this. In order to avoid delays, it could be implemented some initiatives such like: the team members that arrive after the hour scheduled, are responsible for preparing the coffees, for all the team, in the end of the meeting.

The purpose of this meeting is to discuss and plan the sprint. In order to do that, the Product Owner could print out the most high priority JIRA items (Product Backlog), and bring them to the meeting. The results that outcome from this meeting (“*Sprint Backlog*”) should be inserted into the project

⁹ 7.2.4 on page 101

¹⁰ 7.2.2.3 on the following page

wiki¹¹ by the Product Owner (project leader) in order to be accessible to everybody (in particular team members), and JIRA should be used to execute that outcome.

On this phase it has to be decided if bug-fixing tasks (stories) should be considered normal tasks, or if it is simply assumed that the team will spend a certain amount of time (for example 50%), each sprint, fixing those JIRA reported bugs¹². The author considers that the second choice might give more options, in the sense that the a bug fixing duration is uncertain, and this might be difficult to estimate in a Sprint Planning Meeting.

Sprints Synchronization

Since AP has different projects, it will have different sprints. Some ScrumMasters argue that the synchronization of the sprints might be useful. Considering AP, the author believes this would be wise, but not mandatory. This synchronization should be handled by the ScrumMaster.

7.2.2.4 Daily Scrums

The Daily Scrums should be small meetings of 10-15 minutes, with the project leader, in the beginning of the morning. For this meeting, all the team members should gather in a common space with the project leader and the ScrumMaster (may not be essential his presence, depending from the complexity of the project). The important thing to keep is that this meeting intends to promote the discussion between what the developers have done in the previous day, what they intend to do in that day and what were the difficulties that they found in the middle. On the other hand, this can be seen as a daily checkpoint, of the project, to the project leader (Product Owner).

7.2.2.5 Sprint Review

This meeting intend to show the sprint results to the client, section leader and even to other development teams. The author thinks that this might have sense, not as a rule but as a practice to be done when justified. This would be decided and coordinated by the ScrumMaster. One idea could pass thought the TMs or even the SMs to present this review to the section leader¹³. Still, the results need to be presented to the product client, but considering his involvement and close contact, this might be done in a informal way at the end of a sprint.

¹¹ 7.4.3.1 on page 104

¹² [Kniberg(2007)]

¹³ AP SMs (Section Meetings) have been working pretty much with this purpose

7.2.2.6 Sprint Retrospective

This meeting intends to do a retrospective of a team sprint, in order to evaluate how the sprint was done, and what things could be improved. This could be done during the Open Day¹⁴.

7.2.2.7 Keep the Team Together and Isolated

The team should share the same development space (office, open space, ...). This increases heavily the contact between the developers, promotes positive discussion, and isolates the team from other developers outside the team. AP has done some efforts in order to assure this. As it was mentioned in the previous chapter, there are cases where developers from different teams share the same space but, considering the AP space limitations, this is being improved in the last few months.

On the other hand, the isolation of the team is really crucial for a sprint success. Once a team is committed within a sprint, she cannot be disturbed. The client cannot appear and add requirements to the Product Backlog. This kills the sprint!!! The ScrumMaster and the section leader have the responsibility to protect the team and assure that this situations will not happen.

7.2.3 XP Practices

This topic intends to introduce some XP practices into this model proposal. Not all the twelve practices will be presented, but only the ones that the author considers critical.

7.2.3.1 Code Standard

If programmers all adhere to a single coding standard (including everything from tabs vs. spaces and curly bracket placement to naming conventions for things like classes, methods, and interfaces), everything just works better. It is easier to maintain and extend code, to refactor it, and to reconcile integration conflicts, if a common standard is applied consistently throughout. The standard itself matters much less than adherence to it. Fortunately, modern IDEs (such like Eclipse) make it trivial to apply many kinds of formatting, even after the fact. Another idea could pass through the integration of a code verification/validation tool into the common build¹⁵. So, once the developer try's to release a product, this will be verified by this validator.

¹⁴ 7.4.2 on page 103

¹⁵ is a AP software tool for automating software build processes

Of course, first of all it is imperative to define a code standard, and this standard should be the same for all the AP projects. So, in order to do this, the author believes that it is necessary to gather the agreement of most of the developers and elaborate a document accessible (should be visible in AP wiki page) and followed by everybody. This discussion should be open to all the developers, and not only between the project leaders and the ScrumMaster. The section leader could participate as well. Yet, at the end, everyone must assume the commitment to adopt the defined standards. This should be monitored by the project leaders and the ScrumMaster.

7.2.3.2 TDD

This practice should be seen as a revolutionary way of coding. AP has very few code with tests, and the code that is created does not follow, in general, this path. TDD could be seen as a way to change this logic and to introduce tests from the beginning. Concerning this, there exist several manuals and tutorials to explain how to use TDD¹⁶.

TDD is strongly recommended for AP, as a way to give a huge quality jump with the new created code, and even to the old one which, for sure, will not complain for the attention¹⁷!!!

However, TDD takes time to be implemented. This should be progressive, supported with training and adopted, in a first stage, by the more experienced developers. After that, those ones should spread their knowledge through the other developers.

7.2.3.3 On-Site Client

This practice is somehow assured by the Scrum model. However, in this topic is important to mention that the OP clients should be present to clarify and help the team if necessary. So, situations like the ones described in the last chapter¹⁸, can not happen anymore. This means that the client can not interfere in the work of the developers during a sprint. If new features need to be added, this should be discussed with the Product Owner (project leader) and wait for the beginning of a new sprint.

Of course, this behavior will demand a change of mentality by the CO members, which should be promoted by the section and group leader in order to protect the projects development teams.

¹⁶some ideas were present in the XP model analysis 5.2.2.4 on page 61

¹⁷not to forget that AP has 5 million lines of code to maintain, some of them with low quality

¹⁸6.3.2.6 on page 81

7.2.3.4 Pair Programming

The author believes that the pair programming practice would be a strong AP asset, for a number of reasons:

- Improves the quality of the code produced, which is a critical issue in controls software
- Increases the discipline and better time management. Developers are less likely to skip writing unit tests, code writing standards or other violations of discipline
- Learning and training the developers. This can have two sides:
 - It can be very useful to improve the skills of the developers, introducing new technologies and practices with more experienced developers. This allows to spread knowledge more quickly amount the team
 - Can be an excellent opportunity to introduce the new comers to the projects and to a set of practices. The contact with a more experienced developer is a opportunity to promote a smoothly integration and adaptation to the team practices, improving significantly more the understanding about the system and about the software development.
- Information flows and, after a few time, multiple people understand each piece of the project. This is a way to introduce another XP practice, which is the **Team Code Ownership**¹⁹. This practice is very important for AP, considering the amount of temporary developers. So, AP will be less affected with the leaves of personnel.

In fact, most of the developers that are against this practice actually have not tried it. On the other hand, programming has traditionally been taught and practiced as a solitary activity and many experienced programmers are very reluctant to program with another person. Some say their code is “*personal*“, or that another person would only slow them down.

Although, several studies agree that pair programming is more effective than individual programming²⁰, and his implementation at AP could be an excellent example to demonstrate that. Still, this practice should not be used every day, but time-to-time depending of the circumstances, and of course, his implementation and success depends a lot of mentality changes.

¹⁹ 5.2.2.7 on page 62

²⁰ [Cockburn(2000)]

7.2.3.5 Other Practices

The other XP practices are fully recommend as well. In fact, some of them are already used, but without integration with a development model. With CMMI structure and the Scrum model, it is possible to accommodate most of those practices in a smooth way.

7.2.4 Some Suggestions

This topic is here to state that, there are other software development models that have good practices, and can be integrated as well, according to the needs of the projects. A model is not static and must always look for new approaches and good ideas. A good example are the FDD inspections to remove defects and improve quality. XP uses pair programming, but both could be seen as variants of the same thing! Pair programming is a process of continuous inspections, while FDD uses discrete inspections. So, in some projects is possible to choose one or another according to the needs.

Another good idea is the role of Chief Architecture ²¹ proposed by the FDD. In fact, the author proposes the implementation of this role in order to keep the coherence of the several AP projects, since they represent the same “*control system*“. This makes all sense because it is also a way to create a common understanding and view between projects. It was suggested as well, that this role could be filled by the proposed ScrumMaster and, of course, should profit from the help of the different project leaders.

7.3 Documentation

7.3.1 Structural Documentation

As it was referred in the previous chapter, AP does not have structural documentation of his projects. This is a problem that has to be solved. For that reason, the author suggests the creation of a team capable to do this job. This team should be constituted by the older, and more experienced projects developers. Some would say that this step is contrary to the Agile approach. In the authors opinion, it is an essential step to promote this proposal, by the fact that without a full understanding of the projects, accessible to everyone, any kind of model will always have problems to establish his practices.

The author is not defending the creation, for instance, of requirements documents for each project in the beginning of a Scrum sprint, by the contrary. The Scrum Product Backlog is enough to assure this. What the authors defends are

²¹ 5.3.1 on page 67

documents that state the foundations of the projects, with their main decisions, the business domain (use of UML diagrams), design and first requirements, in order to have a reference to understand what was done in the beginning, and why. Manuals about how the products work are also welcome. All of these information's should be available in each product wikipage.

7.3.2 Code

The code documentation is a essential practice that should be done by all the developers without exception. This is a important measure of quality and fundamental to the code understanding by other developers. Since AP is Java oriented, the creation of Javadoc has to be a demand for all the team developers.

Concerning this, the “*programming by contract*” concept might be a interesting approach to explore, which will not be done in this dissertation.

7.4 Tools

7.4.1 Training Course

The author strongly defends the creation of a training course to be attended by those that arrive to the section. The idea of this course is to promote and prepare the integration of new comers. In fact, when someone arrives to AP, he should not be authorized to even look to code without attending the course.

This course should be able to provide a number of essential things, such like:

- Training period (one or two weeks for instance) in the CCC with OP members, in order to understand their needs and how they handle the control system
 - This is very important to promote the understanding of the business domain
- Full understanding of the software development process adopted
 - Practices of the team
 - Used tools
 - Code standards
 - ...
- Presentation of the main projects, their business domains and technologies

- Pair programming period with the project (the one he will work for) team members in order to understand:
 - The business domain
 - Main technologies
 - Design decisions
 - Code standards
- ...

7.4.2 Open Day

In Google developers have 20% of their time available for themselves. This sounds a good idea in order to encourage the creativity of the developers, and to give them space to read about new technologies, develop their own hobby projects and so on. This is possible to be done at AP with a different format, but with the same concept.

The Open Day would be in all Fridays. However, this day should still be reserved to the technical and section meetings²², but also to do sprint retrospectives, for instance, during the cake's break²³. The remaining time should be used for personnel activities. Of course, if any serious situation occurs with any product, and considering that it is critical software, that problem has to be solved, but outside a sprint context.

7.4.3 Software Metrics

First of all, it is important to say that this topic is really wide. Still, it was considered important to refer some ideas about it.

The CMMI model, in particular to reach levels 4 and 5, demands the introduction of software metrics. A software metric is a measure of some piece of software property or its specifications. There are several software metrics such like:

- Source lines of code
- Bugs per lines of code
- Code coverage, measures the code lines that are executed for a given set of software tests
- Number of lines of customer requirements

²²as it is already done and fully encouraged

²³ 6.3.3.3 on page 84

- Number of classes and interfaces

This does not look very important for a initial implementation of this proposal, that intends to reach the CMMI level 3. However, should be consider for the future.

It is important to refer that Scrum provides tools such like: Sprint Burndown Chart and Release Burndown Chart. For instance, the first one is a metric to help the team to see how she is progressing during the sprint. Still, there are other software metrics that can be implemented with the proposed model.

7.4.3.1 Wikis

Each project should have a wikipage with all the information about the product, such like: product backlogs, sprint backlogs, manuals, and other relevant information. This page should be visible to all the developers and clients and updated by the project team.

7.4.4 ProKey

The ProKey is a tool created by the author with the purpose of gathering valuable information about the projects developed by the different teams. The purpose of this is to increase the communication and cooperation between the developers in order to exchange knowledge, in a attempt to avoid repetition of ideas, which, in most of the time, reflects into repeated lines of code. So, this product intends to create a common base of knowledge:

- Relevant for the developers
- Simple, quick and intuitive
- Updated very often
- Accessible to everyone

Concept The general concept behind the application is quite simple. Every product has a context, so it can be described by a set of keywords that can report the major characteristics of the project, eventual algorithms or implementations used, and services provided. So, this information would be provided through the *product.xml* file, that each project has, through a “*info xml tag*”.

After that, the ProKey gathers all that information and displays it into two HTML pages, connected between them. In one of them, there is a list of all the available products with the correspondent keywords and general info about the project (developers, client, technologies, state and so on). In the other page,

there is a list of all the keywords, with links to all the projects that refer to each one of the keywords²⁴.

7.4.5 Additional Notes

Just to mention that exists several tools that allow to manage the Scrum and XP practices. Still, it was not possible to do a deep evaluation of some of them like: ScrumWorks²⁵ (Scrum), VersionOne²⁶ (Scrum), and XPlanner²⁷ (XP).

7.5 Control System Perspectives

The author did not wanted to propose specific ideas in what concerns the CERN control system and the problems stated in the previous chapter ²⁸. For this it would be necessary another document in another context, which is not the purpose. Instead, it will be given a perspective from how the author sees a control system, which might be interesting to consider for the future of the actual system that is handled by AP.

In the author's opinion, a control system should be a unique and homogeneous project with several modules. Each module tries to fill a specific need of that system. On the other hand, the system should be horizontal, in the sense that all the modules should be at the same level of abstraction, avoiding situations where a module crosses several layers. This project would be developed by a unique team, distributed by different modules, but with the same common view of the system. The modules would have a well defined scope and objective, which can not be overlap by another module. All the project would have a unique software development process, possibly based on the proposal done in this document. Maybe one day this could be a reality.

²⁴for more information: 8.3 on page 111

²⁵[Danube()]

²⁶[TOO()]

²⁷[XPlanner()]

²⁸6.2 on page 77

Part IV

Conclusions

*The first step towards getting somewhere is to decide that you are
not going to stay where you are.*

John Pierpont Morgan (1837 - 1913)

Chapter 8

Critic

8.1 Considerations

This chapter will present the final considerations and conclusions about all the contents that were presented previously. This intends to be an overview of the ideas presented and discussed during this master thesis dissertation.

The SIP initiative had clear objectives which consisted, essentially, in the identification and improvement of the software development process, and practices, of the CERN CO-AP projects, in order to improve the communication between teams, and to increase the quality of the code developed. However, during the analysis performed thought this dissertation, it was concluded that AP does not have a consistent software development process implemented in his projects. In fact, we even can say that AP does not have a established software development process at all.

In order to reinforce this conclusion CMMI evaluation model was used to understand in with level the section was. The conclusion did not surprised, but it was still disappointing. In fact, AP is clearly at the **CMMI Level 1 - Initial**. The AP analysis showed that there are a set of disorganized practices, but complemented by extraordinary skilled, dedicated and united developers with an excellent section leader, that allows to explain why AP has success at the eyes of the CO group. Still, this is not enough. AP is relatively aware about his limitations and mistakes and by this reason the SIP initiative was created.

So, in order to better understand what are the paths that can be tracked, this master thesis dissertation promoted an analysis of software engineering, into the context of software development processes. The issue of the software crisis still remains. We still do not know how to develop perfect software, capable of meeting the requirements, with quality and within the deadline. This issue exists since the origins of the software development until today. Several development

models had appear during the years, in order to solve this problematic. Several approaches have been proposed. This dissertation does an overview of that. Throughout this overview, some suitable solutions are proposed to solve the AP problems.

The solution proposed jumped from the middle of the Agile development processes. The Scrum development model mixed with XP best practices is certainly a good option to be integrated and to solve the way how AP developers software. Actually, according to the statistics, Agile development is growing considerably, and Scrum and XP gather a general consensus in the software development community. However, in order to institutionalize this agile development process there are proposed a set of CMMI practices as well. This will give some discipline to the agile practices, in what is called Agile CMMI.

With this proposal it is assumed the clear objective to bring AP to the CMMI level 3 in a short, medium term period (1-2 years), and maybe in a long term period (4-5 years) reach the CMMI level 4 or even 5, according to the AP performance and needs. Still, the author believes that, with the adoption of this proposal, all the conditions are created for developing better quality software, more predictable, within a consistent structure, maintainable and in a more pleasant way. On the other hand, this will also assure that all AP projects will follow the same practices and principles. The author also believes that this will increase greatly the productivity of AP developers, statistics confirm that, but also the integration and future work of the newcomers.

But Agile is about people. A process like this one only works if everybody is committed and motivated, from the section leader until each one of the developers.

8.1.1 Resistance to Change

One of the nastiest, most debilitating workplace cancers is resistance to change. Fear is palpable in organizations pursuing change initiatives. In breaking through fear-fueled resistance, it is critical to identify who's losing what, anticipate over-reaction, acknowledge the losses and give something back. The adoption of a software development model like this one involves lots of changes from the bottom (developers) to the top (management). This must be done in a consistent and resolute way, but respecting the history and legacy of AP, his work and most of all, his personnel. People must be listen. They should have the change to express their concerns and fears, and most of all, they must be informed about the advantages that come with a change like this one. This model should always focus on communication amount the developers.

However, it is crucial not to forget the OP members as well, in order to

have success in AP. The OP members must be educated and trained to understand what are the intentions of the new development model, and what are the advantages that it brings to them, and to the organization.

Only together, with the support and commitment of everybody involved, it is possible to introduce this proposal in a revolutionary, but smoothly way as well. However, considering the open mind of the AP developers and management, the author believes that this has all the conditions to happen.

8.2 Improvements

The introduction of a software development process in a organization has several issues that can be focus. In the middle of this, there are always topics that can be improved. One of them is the issue of the tools proposed by this model. There exist several tools to use with Scrum model, in what concerns the management of the process, that could have been proposed and analyzed as well.

There is also the issue of the software metrics and how to use them in this model. This also could be improved in order to study the possibility to implement a CMMI level 4 or 5. A deeper analysis of some specific AP projects might had some value as well, in order to particularize some common observations done in AP analysis.

On the other hand, there are several other Agile processes that could have been analyzed, and they weren't, such like the Crystal methods, DSDM, Lean and many others with different perspectives that might have some value for AP context. Still, in a master thesis dissertation, choices have to be done, and the author did his owns, in what he believes the best for this context.

8.3 Acknowledgments

I would like to acknowledge the contribution and ideas of some AP members for some of the issues and concerns expressed in this dissertation. To Niall Stapley, a honest thank-you for his vision about software development. To Jakub Wozniak, my supervisor and mentor, for all the knowledge and skills that passed to me during these fourteen months of stay at CERN. To David Garcia Quintas, for all the conversations and good times we passed together.

I also would like to address my gratitude to my FEUP supervisor, Jaime Villate, for all the assistance provided during the creation of this master thesis dissertation.

Finally, but not less, to my family that missed me all these incredible months abroad.

ProKey

product.xml <info> example:

```
<product>
  <info>
    <desc>Description</desc>
    <developers>
      <developer name='Pedro Nuno Pinto'>
        <contact>Pedro.Nuno.Pinto@cern.ch</contact>
        <webpage>www.system-out.com</webpage>
      </developer>
    </developers>
    <audience>Stefano Redaelli</audience>
    <status>Level 5 (Production/Stable)</status>
    <languages>
      <language>Java</language>
    </languages>
    <keywords>
      <keyword type='addressing' name='LSA Client'>
        <desc>LSA Client use description</desc>
      </keyword>
      <keyword type='addressing' name='LSA Client'>
        <desc>LSA Client use description</desc>
      </keyword>
    </keywords>
  </info>
</product>
```

SIP - Products ⓘ																											
accsoft-commons-cache (2.02.01) ⓘ																											
Description Caching API on top of Ehcache																											
Keywords <table> <tr> <th>Type</th><th>Name</th><th>Description</th><th>Webpage</th></tr> <tr> <td>providing</td><td>Cache Value</td><td></td><td></td></tr> <tr> <td>providing</td><td>Cache Group</td><td></td><td></td></tr> <tr> <td>providing</td><td>Cache Constants</td><td></td><td></td></tr> <tr> <td>providing</td><td>Cache Service</td><td></td><td></td></tr> <tr> <td>using</td><td>Ehcache</td><td></td><td>http://ehcache.sourceforge.net/</td></tr> </table>				Type	Name	Description	Webpage	providing	Cache Value			providing	Cache Group			providing	Cache Constants			providing	Cache Service			using	Ehcache		http://ehcache.sourceforge.net/
Type	Name	Description	Webpage																								
providing	Cache Value																										
providing	Cache Group																										
providing	Cache Constants																										
providing	Cache Service																										
using	Ehcache		http://ehcache.sourceforge.net/																								
accsoft-commons-core (0.10.2) ⓘ																											
Description This module provides a set of features in order to deal with tasks and to dispatch them through JMS command messages.																											
Keywords <table> <tr> <th>Type</th><th>Name</th><th>Description</th><th>Webpage</th></tr> <tr> <td>providing</td><td>JMSCommands</td><td>Dispatches JMSCommands (gets methods names, parameters, declarations, class names) taken out of the JMS messages to registered listeners.</td><td></td></tr> <tr> <td>providing</td><td>JMS Buffered</td><td>Allows to intercept listener calls. It queues them in a bounded queue, and groups together in one JMS message (to</td><td></td></tr> </table>				Type	Name	Description	Webpage	providing	JMSCommands	Dispatches JMSCommands (gets methods names, parameters, declarations, class names) taken out of the JMS messages to registered listeners.		providing	JMS Buffered	Allows to intercept listener calls. It queues them in a bounded queue, and groups together in one JMS message (to													
Type	Name	Description	Webpage																								
providing	JMSCommands	Dispatches JMSCommands (gets methods names, parameters, declarations, class names) taken out of the JMS messages to registered listeners.																									
providing	JMS Buffered	Allows to intercept listener calls. It queues them in a bounded queue, and groups together in one JMS message (to																									

Figure 8.1: ProKey - Products to Keywords I

SIP - Products ⓘ																											
accsoft-commons-cache (2.02.01) ⓘ																											
Description Caching API on top of Ehcache																											
Keywords <table> <tr> <th>Type</th><th>Name</th><th>Description</th><th>Webpage</th></tr> <tr> <td>providing</td><td>Cache Value</td><td></td><td></td></tr> <tr> <td>providing</td><td>Cache Group</td><td></td><td></td></tr> <tr> <td>providing</td><td>Cache Constants</td><td></td><td></td></tr> <tr> <td>providing</td><td>Cache Service</td><td></td><td></td></tr> <tr> <td>using</td><td>Ehcache</td><td></td><td>http://ehcache.sourceforge.net/</td></tr> </table>				Type	Name	Description	Webpage	providing	Cache Value			providing	Cache Group			providing	Cache Constants			providing	Cache Service			using	Ehcache		http://ehcache.sourceforge.net/
Type	Name	Description	Webpage																								
providing	Cache Value																										
providing	Cache Group																										
providing	Cache Constants																										
providing	Cache Service																										
using	Ehcache		http://ehcache.sourceforge.net/																								
accsoft-commons-core (0.10.2) ⓘ																											
Description This module provides a set of features in order to deal with tasks and to dispatch them through JMS command messages.																											
Keywords <table> <tr> <th>Type</th><th>Name</th><th>Description</th><th>Webpage</th></tr> <tr> <td>providing</td><td>JMSCommands</td><td>Dispatches JMSCommands (gets methods names, parameters, declarations, class names) taken out of the JMS messages to registered listeners.</td><td></td></tr> <tr> <td>providing</td><td>JMS Buffered</td><td>Allows to intercept listener calls. It queues them in a bounded queue, and groups together in one JMS message (to</td><td></td></tr> </table>				Type	Name	Description	Webpage	providing	JMSCommands	Dispatches JMSCommands (gets methods names, parameters, declarations, class names) taken out of the JMS messages to registered listeners.		providing	JMS Buffered	Allows to intercept listener calls. It queues them in a bounded queue, and groups together in one JMS message (to													
Type	Name	Description	Webpage																								
providing	JMSCommands	Dispatches JMSCommands (gets methods names, parameters, declarations, class names) taken out of the JMS messages to registered listeners.																									
providing	JMS Buffered	Allows to intercept listener calls. It queues them in a bounded queue, and groups together in one JMS message (to																									

Figure 8.2: ProKey - Products to Keywords II

SIP - Keys -> Products		
Cache Group		SIP - Software Improvement Process This page does an inventory of the available keywords and the projects (and packages) associated to those keywords Website Legend Keywords types providing: provides a service, feature, technology, etc using: uses a library, API, technology, etc addressing: addresses certain business domain, problem, etc
Product	Type	
accsoft-commons-cache	providing	
Monitoring GUI		
Product	Type	
ihc-collimators-fixedisplay	using	
AbstractDAO		
Product	Type	
accsoft-commons-springvalue	providing	
Tasks		
Product	Type	
accsoft-commons-core	providing	
CTF TGM		
Product	Type	
ctf-fixedisplay	using	

Figure 8.3: ProKey - Keywords to Products I

Product	Type
accsoft-gui-fixedisplay	providing
Spring beans	
Product	Type
accsoft-commons-core	using
accsoft-commons-springvalue	using
accsoft-gui-fixedisplay	using
japc-monitoring	using
ihc-collimators-fixedisplay	using
PSB accelerator	
Product	Type
psb-fixedisplay	addressing
User info	
Product	Type
accsoft-commons-utd	providing
Cache Value	
Product	Type
accsoft-commons-cache	providing

Figure 8.4: ProKey - Keywords to Products II

Nomenclature

AB	Accelerators and Beams department
ACM	Association for Computing Machinery
ADM	Advanced Development Methods
ALICE	A Large Ion Collider Experiment
AP	Applications section
ASD	Adaptative Software Development
ATLAS	A Toroidal LHC ApparatuS
AUP	Agile Unified Process
CASE	Computer-aided software engineering
CCC	CERN Control Centre
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integrated
CMS	Compact Muon Solenoid
CO	Controls group
DSDM	Dynamic Systems Development Method
FDD	Feature Driven Development
FDF	Fixed Displays Framework
GP	Generic Practices
GUI	Graphical User Interface
IDE	Integrated Development Environment

IEEE	Institute of Electrical and Electronics Engineers
IID	Iterative and Incremental Development
JAPC	Java API for Parameter Control
LASER	LHCAlarm SERvice
LEP	Large Electron-Positron
LHC	Large Hadron Collider
LHCb	Large Hadron Collider beauty
LSA	LHC Software Application
OP	Operation section
PDSA	Plan-Do-Study-Act
PS	Proton Synchrotron
ROI	Return on Investment
SDP	Software Development Process
SEI	Software Engineering Institute
SIP	Software Improvement Process
SIS	Software Interlock System
SM	Section Meeting
SMEs	Subject Matter Experts
SPI	Software Process Improvement
SPS	Super Proton Synchrotron
TDD	Test-Driven Development
TM	Technical Meeting
UML	Unified Modeling Language
XP	Extreme Programming

Bibliography

- [APA(2008)] 2008. URL <http://www.britannica.com/>.
- [ASD(2006)] Extreme programming, February 2006. URL <http://www.extremeprogramming.org/index.html>.
- [ASD(2007)] 2nd annual survey "the state of agile development", June - July 2007.
- [SDP(2003)] *Guidelines for the Successful Acquisition and Management of Software Intensive Systems (GSAM)*, chapter 3. USAF Software Technology Support Center, May 2003.
- [SDP(2008)] Iterative and incremental development, 2008. URL http://en.wikipedia.org/wiki/Iterative_and_incremental_development.
- [SDP(2001a)] Manifesto for agile software development, February 2001a. URL <http://agilemanifesto.org/>.
- [SDP(2001b)] Principles behind the agile manifesto, February 2001b. URL <http://agilemanifesto.org/principles.html>.
- [SE0(2007)] A brief history of software engineering, September 2007. URL <http://pagkis-software-testing.blogspot.com/2007/09/brief-history-of-software-engineering.html>.
- [TOO()] Versionone. URL <http://www.versionone.com/>.
- [WP0()] Software development process. URL http://en.wikipedia.org/wiki/Software_development_process.
- [WP0(2008a)] Agile software development, March 2008a. URL http://en.wikipedia.org/wiki/Agile_software_development.
- [WP0(2008b)] History of software engineering, 2008b. URL http://en.wikipedia.org/wiki/History_of_software_engineering.

- [WP0(2008c)] Software crisis, 2008c. URL http://en.wikipedia.org/wiki/Software_crisis.
- [Beck(1999)] Kent Beck. *Extreme Programming Explained*. 1999.
- [Brice(2006)] Maximiliem Brice. Several views of the new cern control centre. différentes vues de la nouvelle salle de contrôle du cern (ccc), March 2006.
- [Brooks(1995)] Frederick P. Brooks. *The Mytical Man-Month*. 1995.
- [Brooks(1987)] Frederick P. Brooks. No silver bullet - essence and accidents of software engineering. *Computer Magazine*, 1987.
- [CERN(2007)] CERN. Cern ask an expert service, 2007. URL <http://askanexpert.web.cern.ch/AskAnExpert/>.
- [Charrue(2008)] Pierré Charrue. Under control: keeping the lhc beams on track. *CERN Courier*, March 2008.
- [Cockburn(2008)] Alistair Cockburn. Incremental versus iterative development, February 2008. URL http://alistair.cockburn.us/index.php/Incremental_versus_iterative_development.
- [Cockburn and Highsmith(2001)] Alistair Cockburn and Jim Highsmith. Agile software development: The people factor. *Software Management*, November 2001.
- [Cockburn(2000)] Alistair A.R. Cockburn. Costs and benefits of pair programming, 2000. URL http://alistair.cockburn.us/index.php/Costs_and_benefits_of_pair_programming.
- [Coffin(2006)] Rod Coffin. A practical guide to seven agile methodologies, October 2006. URL <http://www.devx.com/architect/Article/32761/0/page/1>.
- [Cohn(2007)] Mike Cohn. *Used Stories Applied for Agile Software Development*. The Addison Wesley, 10th edition, November 2007.
- [Craig Larman(2003)] Victor R. Basili Craig Larman. Iterative and incremental development: A brief history. *IEEE Computer Society*, June 2003.
- [Danube()] Danube. Scrumworks. URL <http://www.scrumworks.com/>.
- [DeGrace and Stahl(1991)] Peter DeGrace and Leslie Stahl. *Wicked Problems, Righteous Solutions*. 1991.

- [DeLuca(2008)] Jeff DeLuca. Jeff deluca on feature driven development, 2008. URL <http://se-radio.net/podcast/2008-01/episode-83-jeff-deluca-feature-driven-development>.
- [Fritzsche and Keil(2007)] Martin Fritzsche and Patrick Keil. Agile methods and cmmi: Compatibility or conflict? *e-Informatica Software Engineering Journal*, 1, 2007.
- [Gibbs(1994)] W. Wayt Gibbs. Software's chronic crisis. *Scientific American*, 1994.
- [Gilb(1976)] T. Gilb. *Software Metrics*. Winthrop Publishers, 1976.
- [Gilb(1985)] Tom Gilb. Evolutionary delivery versus the "waterfall model". *SIGSOFT Softw. Eng. Notes*, 10(3), 1985.
- [Gilb(1988)] Tom Gilb. *Principles of Software Engineering Management*. 1988.
- [Henney(2007)] Kevlin Henney. Iterative and incremental development explained, December 2007. URL http://searchsoftwarequality.techtarget.com/news/article/0,289142,sid92_gci1284193,00.html#.
- [Highsmith(2002)] Jim Highsmith. *Extreme Programming - Agile Project Management Advisory Service*. Cutter Consortium, 2002.
- [Humphrey(1989)] M. Humphrey, W. Kellner. Software process modelling: Principles of entitly process models. Technical report, Software Engineering Institute, Carnegie Mellon University, 1989.
- [Jeffries(2001)] Ron Jeffries. What is extreme programming?, August 2001. URL <http://www.xprogramming.com/xpmag/whatisxp.html>.
- [Jeffries(2000)] Ron Jeffries. Extreme programming and the capability maturity model, 2000. URL http://www.xprogramming.com/xpmag/xp_and_cmm.htm.
- [Kniberg(2007)] Henrik Kniberg. *Scrum and XP from the Trenches*. InfoQ Enterprise Software Development Series, 2007.
- [Lewallen(2005)] Raymond Lewallen. Software development life cycle models software development life cycle models software development life cycle models software development life cycle models software development life cycle models, July 2005. URL <http://codebetter.com/blogs/raymond.lewallen/archive/2005/07/13/129114.aspx>.

- [Ltd.()] Nebulon Pty. Ltd. Agile software development using feature driven development (fdd). URL <http://www.nebulon.com/fdd/index.html>.
- [Mahoney(2004)] Michael S. Mahoney. Finding a history for software engineering. *Annals of the History of Computing*, 2004.
- [Microsystems()] Sun Microsystems. The java language - an overview. URL <http://java.sun.com/docs/overviews/java/java-overview-1.html>.
- [Palmer(2002)] Stephen R. Palmer. Feature driven development: Best practices, March 2002. URL <http://www.informit.com/articles/article.aspx?p=26059&seqNum=1>.
- [Pinto(2007)] Pedro Nuno Pinto. Fixed displays framework implementation at cern. Technical report, FEUP, 2007.
- [Royce(1970)] Winston W. Royce. Managing the development of large software systems: Concepts and techniques. *Proceedings of IEEE WESCON Technical Paper*, pages 1–9, August 1970.
- [Savain(2006)] Louis Savain. The silver bullet: Why software is bad and what we can do to fix it, 2006. URL <http://www.rebelscience.org/Cosas/Reliability.htm#WhyBad>.
- [Schwaber and Beedle(2001)] Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall, 2001.
- [Sutherland et al.(2007)Sutherland, Jakobsen, and Johnson] Jeff Sutherland, Carsten Ruseng Jakobsen, and Kent Johnson. Scrum and cmmi level 5: The magic potion for code warriors. *EUROPEAN SEPG 2007*, 2007.
- [Takeuchi and Nonaka(1986)] Hirotaka Takeuchi and Ikujiro Nonaka. *The New New Product Development Game*. Harvard Business Review, 1986.
- [Tom DeMarco(1987)] Timothy Lister Tom DeMarco. *Peopleware - Productive Projects and Teams*. 1987.
- [Vanoli(2006)] Christine Vanoli. The cern accelerator complex. complexe des accélérateurs du cern, June 2006.
- [XPlanner()] XPlanner. Xplanner. URL <http://www.xplanner.org/>.