Ferramenta de Gestão de Processos de Desenvolvimento de Software

Rui Miguel Ferreira Francisco

Dissertação para obtenção do Grau de Mestre em Engenharia Informática e de Computadores

Júri:

Presidente: Prof. Pedro Manuel Moreira Vaz Antunes de Sousa
Orientador: Prof. Miguel Leitão Bignolas Mira da Silva
Vogais: Prof. Ademar Manuel Teixeira de Aguiar

Setembro de 2008
Abstract

Information systems tend to last several years. But they have to be updated over time to meet the new business needs, originating new requirements. Maintenance plays an important role in the software’s life cycle, incorporating the new needs into the existing product, through a controlled, defined and effective development process. Unfortunately this is not an easy task. Most of the projects slipped, exceeding their initial budgets and schedules. Additionally, maintenance costs are rising, with estimated cost from 60% to 80% of the total software budget. Maintenance is an extended subject. This study is focused only in the perfective maintenance process.

A few techniques (such as IEEE Standard 1219, CMM, Scrum and ITIL) are being used in order to reduce the costs and gain control over the process. Besides that it is necessary an excellent knowledge of the perfective maintenance process, as well a management tool with good control mechanisms, aligned with the process.

With this study we propose to elicit the perfective maintenance process and to create a management tool which reflects the process needs and integrates the old systems used by OutSystems.

Working close to OutSystems’ professionals, it was defined and evaluated their main needs during the perfective process. The result was a documented and defined perfective maintenance process. Moreover it was created a software management tool to support the elicited process.

The resulted process was validated by the users and the new tool used to manage new releases. The created tool increased the control over the working items and reduced the total time spent in the process.

Although, innovation, improvements and optimizations continue to be possible in the perfective process and management tool.

Keywords

Maintenance, Perfective Maintenance Process, Management Tool, Control, Visibility
# Table of Contents

1 Introduction

1.1 Research Proposal

1.2 Thesis Organization

2 State of the Art

2.1 Best Practices / Methodologies

2.1.1 IEEE Standard 1219

2.1.1.1 Problem / modification identification, classification, and prioritization

2.1.1.2 Analysis

2.1.1.3 Design

2.1.1.4 Implementation

2.1.1.5 Regression/System Testing

2.1.1.6 Acceptance Testing

2.1.1.7 Delivery

2.1.2 CMM

2.1.2.1 The Software Process Maturity Framework

2.1.2.2 Visibility into the Software Process

2.1.2.3 Key Process Areas

2.1.3 Scrum

2.1.3.1 The Process

2.1.3.2 Roles

2.1.3.3 Ceremonies

2.1.3.4 Artifacts

2.1.4 ITIL

2.1.4.1 Service Support

2.1.5 Summary

2.2 Tools

2.2.1 Target Process

2.2.2 Version One

2.2.3 Rally Software

2.2.4 FogBugz

2.2.5 Summary

3 Case Study / Problem

3.1 Wrong and Ambiguous Terms

3.2 Undocumented and Ineffective Perfective Maintenance Process

3.3 Inappropriate Management Tool

3.4 Integration Between Multiple Systems

3.5 Summary

4 Proposal
## 4.1 Software Engineering Terms

## 4.2 Perfective Maintenance Process

## 4.3 Management tool

## 4.4 Summary

## 5 Implementation / Prototype

## 5.1 Software Engineering Terms

## 5.2 Perfective Maintenance Process

### 5.2.1 Process Mining Approach

### 5.2.2 Interview Approach

## 5.3 Management Tool

### 5.3.1 Methodology

### 5.3.2 OutSystems All-in-One Agile Platform

### 5.3.3 Domain Model

### 5.3.4 Hierarchical Structure

### 5.3.5 Graphical Interface

#### 5.3.5.1 Add and List Releases

#### 5.3.5.2 Add, Associate and List Projects

#### 5.3.5.3 Add, Edit and List Activities and Tasks

#### 5.3.5.4 3-Point Estimate

#### 5.3.5.5 Iterations

#### 5.3.5.6 Priority

#### 5.3.5.7 Resources Allocation and Project's Budget Update

#### 5.3.5.8 Feedback and Reports

## 6 Results

## 6.1 Software Engineering Terms

## 6.2 Perfective Maintenance Process

### 6.2.1 Process Mining Approach

### 6.2.2 Interview Approach

#### 6.2.2.1 Requirements, High Level Architecture and Design Phase

#### 6.2.2.2 Architecture, Design and Implementation Phase

#### 6.2.2.3 Testing and Deployment Phase

#### 6.2.2.4 Perfective Maintenance Process through time

## 7 Evaluation

## 7.1 Add Tasks

## 7.2 Estimations

## 7.3 Testimonials

## 8 Conclusions

## 8.1 Future Work
List of Tables

Table 1: Target Process Pros and Cons.................................................................18
Table 2: VersionOne Pros and Cons.................................................................18
Table 3: Rally Software Pros and Cons.............................................................18
Table 4: FogBugz Pros and Cons....................................................................19
Table 5: Evaluation of the Project Management Tools......................................19
List of Figures

Figure 1: Proportional software maintenance costs................................................................. 1
Figure 2: Percentage of maintenance cost by class................................................................. 2
Figure 3: Perfective Maintenance in the development process................................................ 2
Figure 4: Visibility into the Software Process - Initial Level.................................................. 8
Figure 5: Visibility into the Software Process - Repeatable Level.......................................... 8
Figure 6: Visibility into the Software Process - Defined Level............................................... 8
Figure 7: Visibility into the Software Process - Managed Level............................................ 8
Figure 8: Visibility into the Software Process - Optimizing Level........................................ 8
Figure 9: Scrum Process........................................................................................................... 10
Figure 10: ITIL’s framework.................................................................................................... 13
Figure 11: Relationship between Incidents, Problems, Known Errors and RFCs [26]........... 15
Figure 12: Regular status transitions during a task life cycle................................................. 26
Figure 13: ER diagram for the four Process Mining tables [39]............................................. 27
Figure 14: Process Mining Steps............................................................................................ 27
Figure 15: Interviews Steps...................................................................................................... 28
Figure 16: Simplified Domain Model of the Project Management Tool................................. 30
Figure 17: Example of the New Hierarchal Structure.............................................................. 31
Figure 18: Add a New Release Graphical Interface................................................................. 31
Figure 19: List Releases Graphical Interface........................................................................... 32
Figure 20: Add New Project Graphical Interface..................................................................... 32
Figure 21: Graphical Interface for Associating Older Projects............................................... 33
Figure 22: List Projects Graphical Interface.......................................................................... 33
Figure 23: Add New Task and 3-Point Estimate Graphical Interface.................................... 34
Figure 24: Edit A New Task Graphical Interface...................................................................... 34
Figure 25: List Tasks and Activities Graphical Interface......................................................... 35
Figure 26: Status icons for the Tasks...................................................................................... 35
Figure 27: 3-Point Estimate Graphical Interface...................................................................... 35
Figure 28: Iteration’s Filter Graphical Interface...................................................................... 36
Figure 29: Iteration’s Progress Graphical Interface................................................................. 36
Figure 30: Tasks and Activities’ Priority Graphical Interface................................................ 37
Figure 31: User Role Budget Association Graphical Interface.............................................. 37
Figure 32: User Role Budget Relation Graphical Interface.................................................... 38
Figure 33: Associate a Project with a Google Spreadsheet Graphical Interface..................... 38
Figure 34: Example of an Allocation Sheet........................................................................... 39
Figure 35: Project Allocation and Budget Calculation Steps.................................................. 39
Figure 36: Feedback Messages Example.................................................................................. 40
Figure 37: ProM result using α Algorithm Result.................................................................... 42
Figure 38: Macro view over the Perfective Maintenance Process.......................................... 43
Figure 39: Process and Main Actors of Phase One................................................................. 44
Figure 40: Main Actors and First Process of Phase Two......................................................... 45
Figure 41: Main Actors and Second Process of Phase Two..................................................... 46
Figure 42: Main Actors and Process of Phase Two................................................................. 47
Figure 43: Perfective Maintenance Process Through Time..................................................... 48
Figure 44: Relation between Time Spent and Number of Items........................................... 49
Figure 45: Time Spent to introduce the duration in Add/Edit and 3-Point Interfaces............. 50
Acronyms and Abbreviations

MR - Modification Request
CMM - Capability Maturity Model
CMDB - Configuration Management Database
SLA - Service Level Agreement
RFC - Request for Change
CAB - Change Advisory Board
DSL - Definitive Software Library
DHS - Definitive Hardware Store
CI - Configuration Items
1 Introduction

Nowadays, requirements often change during the product development life cycle to meet shifting business demands. Software companies need to become more and more agile to satisfy all the customers’ needs. In order to achieve higher quality software and on-time deliveries, good control mechanisms are essential, as well as a defined software development process.

During the last few years many authors reported that a lot of organizations do not have a defined process for their software maintenance activities. For this reason software maintenance became one of the most ignored areas in the software engineering field.

Traditionally, maintenance is seen as one of the final activities of the software life cycle. In the waterfall model, proposed by Royce, is the final activity. The IEEE 1074-1997 standard, represents the seventh step of eight software development steps [1]. Only in recent years, the software community has started to recognize software maintenance as a fundamental area of software engineering [2]. In 1998 were developed international standards for software maintenance, the IEEE 1219 and ISO/IEC14764 standards were developed and are still being used today [1].

Software maintenance is defined by the IEEE standard 1219 [3] as: *Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment.*

It is wildly recognized that software maintenance is one of the most expensive phases during the software life cycle, with estimated costs from 60% to 80% of the total software budget [4]. A lot of studies have shown that software costs are increasing, as seen in figure 1 [5].

![Figure 1: Proportional software maintenance costs](image)

In recent years studies about software maintenance became more complete, resulting in a clear categorization for software maintenance. The most accepted categorization was done by Lientz and Swanson in a survey study [6]. They categorized maintenance activities into four classes:

- **Adaptive** – Changes made to a system to evolve its functionality to changing business needs or technologies;
- **Perfective** – Changes made to a system to add new features or to improve performance;
• **Corrective** – Changes made to a system to repair flaws in its design, coding, or implementation;

• **Preventive** – Changes made to a system to avoid possible future problems.

The survey also showed that the larger maintenance effort was on the first two types, around 75%, and error correction consumed only 21%. Perfective maintenance is the activity which consumes the most effort, 50% of the maintenance budget, while corrective adaptive and preventive maintenance consume 20% to 21%, 25%, 4% to 5%, respectively, of the maintenance budget [7][8][9]. Many recent studies show a similar scale of the problem. Another studies show that the core problem for software evolution and maintenance is the incorporation of new user requirements [6]. Figure 2 shows the percentage of maintenance cost by class [9].

![Figure 2: Percentage of maintenance cost by class](image)

As we can see, the most effort is spent on adaptive and perfective classes.

Software maintenance is an extended subject. This study is focused only in Perfective Maintenance. Figure 3 presents perfective maintenance in the development process.

![Figure 3: Perfective Maintenance in the development process](image)

Over the years, information systems researchers spent an enormous effort in this problem, trying to reduce maintenance costs and increase control over the process. New techniques such as structure design, CASE tools, and object oriented programming have been adopted to solve this problem. However, most of them proved to be inappropriate, since the most important factor for high cost of software maintenance is the lack of close and effective management of the maintenance process [10].

Another problem present in maintenance is it deficient process. Most of the time the process is undocumented and undefined, making it less efficient. There is no close and effective management of the maintenance process.

The maintenance process should be precise and completely defined. All the tasks in the process must be fully formalized by procedures. Maintainers should have easy access to this information[11]. Studies show that the knowledge of the maintenance process is essential for professionals. Most of them expressed a need for a better mastering of the maintenance process for both managerial and
Formalization of the process is also essential because its degree of formalization is considered to be the main indicator of how mature organizations are [12].

The use of outdated and unappropriated management tools was another difficulty present in maintenance. Instead of helping these tools made the management task much more difficult, delaying the process and increasing its costs.

The management tools have to be aligned with the process, reflecting their changes over time and helping the users to perform their activities more rapidly and effectively. They are needed, in order to plan the maintenance activity, to allocate resources, to track maintenance tasks, to keep the history of problems and solutions, and to organize and search this history [11].

Other techniques were also used to reduce maintenance costs. The IEEE standard 1219 establishes requirements for process, control, management of the planning, execution, and documentation of software maintenance activities [3]. The Capability Maturity Model for Software provides guidance on how to gain control of processes for developing and maintaining software [13]. The Scrum agile methodology gives the values, practices and rules which are frequently used for the maintenance of existing systems [14]. The Information Technology Infrastructure Library (ITIL) is a well-know standard for IT management, providing a set of good practices for managing IT services. One of its goals is to develop and maintain IT services that are easily developed and enhanced to meet future business needs [15].

### 1.1 Research Proposal

The main objectives for this research are to elicit the perfective maintenance process and to improve the tool that manages it, based on the process and in the good practices that were studied. It is important that the new tool reflects the process, to turn the development process more effectively and controlled.

The new management tool must be compatible with older systems, automate the more recurrent activities and increase the visibility of the working items.

Another objective of this research is to clarify the terms used by the OutSystems professionals during the perfective maintenance process.

### 1.2 Thesis Organization

This thesis is divided into the following chapters:

1. **Introduction** – describes the problem, research objectives and structure.
2. **State of the art** – describes the good practices and methodologies used in maintenance and some management tools.
3. **Case Study / Problem** – the case study problems are describes and related with the research area problems.
4. **Proposal** – shows the proposal for the problems that were found in the case study.
5. **Implementation / Prototype** – presents in detail the implementation of the proposal and the details of the management tool prototype.

6. **Results** – describes the results of the study, in particular the perfective maintenance process.

7. **Evaluation** – shows the results of the performed tests over the management tool and some testimonial of the users.

8. **Conclusions** – presents the final thoughts and few orientations for future work.
2 State of the Art

This chapter describes the state of the art about maintenance methodologies and best practices. The first section covers: IEEE Standard 1219, CMM, Scrum and ITIL. The second section describes a few management tools such as: Target Process, VersionOne, Rally Software and FogBugz.

2.1 Best Practices / Methodologies

This section describes in detail the IEEE Standard 1219, CMM, Scrum and ITIL. These subjects are considered the good practices in software maintenance and development.

2.1.1 IEEE Standard 1219

The IEEE Standard for Software Maintenance [3] describes an iterative process for managing and executing software maintenance activities, including the following phases:

- Problem / modification identification, classification, and prioritization;
- Analysis;
- Design;
- Implementation;
- Regression / system testing;
- Acceptance testing;
- Delivery.

2.1.1.1 Problem / modification identification, classification, and prioritization

In this phase, software modifications are identified and classified. Each modification request (MR) must be evaluated to determine its classification and handling priority. The classification must be made according to the following types:

- Corrective;
- Adaptive;
- Perfective;
- Emergency.

2.1.1.2 Analysis

The analysis phase shall use the repository information and the MR validated in the previous phase, along with system and project documentation, to study the viability and scope of the modification and to create a preliminary plan for design, implementation, test, and delivery [3].

2.1.1.3 Design

In this phase, all current system and project documentation, existing software and databases, and the output of the analysis phase (including detailed analysis, statements of requirements, identification of
elements affected, test strategy, and implementation plan) should be used to design the modification to the system [3].

2.1.1.4 Implementation

During this phase, the results of the design phase, the current source code, along with project and system documentation (i.e., the entire system as updated by the analysis and design phases) must be used to drive the implementation effort [3].

2.1.1.5 Regression/System Testing

System testing shall be performed on the modified system. Regression testing is a part of system testing and shall be performed to validate that the modified code does not introduce faults that did not exist before the maintenance activity [3].

2.1.1.6 Acceptance Testing

Acceptance tests shall be conducted on a fully integrated system, and performed by either the customer, the user of the modification package, or a third party designated by the customer. All the tests shall be performed on the modified system [3].

2.1.1.7 Delivery

This phase describes the requirements for the delivery of a modified software system [3]. In all the seven previous phases metrics/measures and associated factors identified should be collected and reviewed at proper intervals.

2.1.2 CMM

The CMM describes an evolutionary improvement path for software organizations from an ad-hoc, immature process to a mature and disciplined one. This path is encompassed by five levels of maturity. Each maturity level comprises a set of process goals that, when satisfied, stabilize an important component of software process [16].

2.1.2.1 The Software Process Maturity Framework

The Software Process Maturity Framework is composed by five different levels: Initial Level, Repeatable Level, Defined Level, Managed Level and Optimizing Level.

Initial Level

In this initial level organizations normally have difficulty making real commitments originating a series of crises. During a project crisis, the planned procedures are discarded and revert to coding and testing. Success depends on having an exceptional manager and an experienced and effective software team. In spite of this ad-hoc process, organizations in Level 1 usually develop products that work [16].
Repeatable Level
In this second level effective processes are defined, documented, practiced, trained, measured, enforced, and improvable. Realistic project commitments are made based on previous successes. Basic software management controls are installed and planning and tracking of the software project is stable. Level 2 organizations can be summarized as disciplined [16].

Defined Level
In the third level, processes are used (and changed, as appropriate) to help software managers and technical staff perform more effectively. A defined software process is used in the project's activities. Both software engineering and management activities are stable and repeatable. Established product lines, cost, schedule, and functionality are under control, and software quality is tracked. Level 3 organizations can be summarized as standard and consistent [16].

Managed Level
At the Managed Level the organization sets quantitative quality goals for both software products and process. Productivity and quality are measured for important software process activities across all projects. Data collected from the projects' defined software processes is analysed. Process is measured and operates within quantitative limits. Level 4 organizations can be summarized as being quantifiable and predictable [16].

Optimizing Level
In the fifth level, the organization is focused on a continuous process of improvement. The organization has the resources to identify weaknesses and strengths of the process, with the goal of preventing the occurrence of defects. Improvements occur both by incremental advancements in the existing process and by innovations using new technologies and methods. Level 5 organizations can be characterized as continuously improving [16].

2.1.2.2 Visibility into the Software Process
This section illustrates the level of visibility into project status and performance afforded to management at each level of process maturity. Each succeeding maturity level incrementally provides better visibility into the software process [17].

Initial Level
In this level software process is an amorphous entity (black box). It is difficult to track projects' progress and activities. Customers can only evaluate if the product meets the requirements when it is delivered [16].
Repeatable Level

During level 2, management controls allow visibility into the project on defined occasions. The process of building the software can be viewed as a succession of black boxes that allows management visibility at transition points as activity flows between boxes. The customer can review the product at defined checkpoints during the software process [16].

Defined Level

At level 3 the tasks in the projects define the software process visibility. Managers and engineers understand their roles and responsibilities within the process and how their activities interact at the appropriate level of detail. The customers can obtain accurate and rapid status updates [16].

Managed Level

During level 4 defined software processes are instrumented and controlled quantitatively. Managers are able to measure progress and problems. Their ability to predict outcomes grows steadily more precise as the variability in the process grows smaller. The customer can establish a quantitative understanding of process capability and risk before the project begins [16].

Optimizing Level

At level 5 new and improved ways are continually tried, to improve productivity and quality. Managers are able to estimate and track quantitatively the impact and effectiveness of change. The customer and the software organization continue to work together [16].
2.1.2.3 Key Process Areas

Except for Level 1, each maturity level is decomposed into several key process areas that indicate the areas an organization should focus on to improve its software process. Key process areas identify the issues that must be addressed to achieve a maturity level. They can be considered the requirements for achieving a maturity level [16].

Repeatable Level

The key areas for Repeatable Level are: Requirements Management; Software Project Planning; Software Project Tracking and Oversight; Software Subcontract Management; Software Quality Assurance; Software Configuration Management [16].

Defined Level

In the Defined Level the areas are: Organization Process Focus; Organization Process Definition; Training Program; Integrated Software Management; Software Product; Engineering; Intergroup Coordination; Peer Reviews [16].

Managed Level

The key areas for managed level are: Quantitative Process Management and Software Quality Management [16].

Optimizing Level

In the last level we have Defect Prevention, Technology Change Management and Process Change Management as key areas [16].

2.1.3 Scrum

Scrum is an agile methodology for software development, characterized as an iterative and incremental process for developing any product or managing any work [18].

Scrum is known to deliver faster and better software for customers or end users, and can be viewed as a collection of good ideas and best practices [19]. Inherits all the benefits from the agile methodologies, such as response to changes, continuous delivery of software and multi-cultural teams working on the product [20]. Scrum is used from small corporations to big ones, like Yahoo and Google [21] [22]. The term Scrum itself was firstly used by Takeuchi and Nonaka in the study called The New New Product Development Game, which was published in the Harvard Business Review [23].

2.1.3.1 The Process

The Scrum process starts with the sources for the product backlog. These sources can be Product marketing, sales, engineering and customer support. The items that might be good ideas for the product are added to the Product Backlog, which is a prioritized list of all product requirements, such as features, defects, and desires. The prioritization is only done by the Product Owner.
In the *Sprint Planning* meeting, *Scrum Teams* meet with the *Scrum Master* to plan the next sprint, usually with duration of 15 to 30 days. This meeting originates the *Sprint Backlog*, which is a list of tasks to be done during the iteration. During the sprint the team seeks to meet the sprint goal, and meets daily for a short status meeting, called *Daily Scrum*.

At the end of the sprint an executable product increment must be presented and any incomplete work returns to the *Product Backlog*. The team gets together with management at the *Sprint Review* meeting to inspect the product increment. Finally, the *Product Backlog* is rearranged and the process starts again with the *Sprint Planning Meeting* [24] [23].

![Figure 9: Scrum Process](image)

In the Scrum process we can identify three roles, three ceremonies, and two artefacts. The roles are: *Product Owner*, *Scrum Master* and *Scrum Teams*. The ceremonies are: *Sprint Planning*, *Sprint Review Meeting* and *Daily Scrum Meeting*. The two artefacts are *Product Backlog* and *Sprint Backlog* [25]. In the next sections, roles, ceremonies and artefacts are presented in more detail.

### 2.1.3.2 Roles

The Scrum methodology is formed by three main roles: Product Owner, Scrum Master and Scrum Teams.

**Product Owner**

The Product Owner is responsible for managing and controlling the Product Backlog, and also ensuring that it is visible to everyone [23]. He must take all the inputs (customers, end-users, team members and stakeholders) and convert them into a product vision [25]. He is in charge for prioritization of the Product Backlog, which is characterized by ensuring that the most valuable functionality is produced and built first [24] [23].

The Product Owner is the person responsible for the product and everybody in the organization must respect his decisions. He is only one person, not a committee, because without a single Product Owner floundering, spin, contention, and frustration will arise [23].

**Scrum Master**

The Scrum Master is responsible for the success of the Scrum and for ensuring that Scrum values, practices, and rules are enacted and enforced. He conducts the Daily Scrums and promotes the
process by removing impediments, enabling the team to organize and manage itself [25] [23]. He works with several people:

- customers and management to identify and institute a Product Owner;
- management to form the Scrum Teams;
- Product Owner and Scrum teams to create a Product Backlog for the sprint;
- Scrum teams to plan and initiate the Sprint.

This role is often assumed by the Team Leader, Project Leader, or Project Manager [23].

**Scrum Teams**

Scrum Teams are committed to achieve the sprint goal, i.e. turning a Product Backlog into a product increment within iteration. Teams are self-managing, self-organizing, and cross-functional, and have full authority to do whatever they decide is necessary to achieve the goal [23] [24].

The ideal size of the team should be seven people, plus or minus two. Small size teams limits the amount of interactions and reduce the productivity, while big teams don’t work well, productivity decreases and the Scrum’s control mechanisms become heavy. At the end of each sprint the composition of the team can change [23].

**2.1.3.3 Ceremonies**

There are three ceremonies that can be found in Scrum: Sprint Planing Meetings, Sprint Review Meetings and Daily Scrum Meetings.

**Sprint Planning Meeting**

The Sprint Planning meeting is actually two consecutive meeting. During the first part of the meeting, users, management, the Product Owner, with the Scrum Team review the Product Backlog, and determine the next Sprint goal and functionality. During the second meeting the team devises the individual tasks that must be performed to build the product increment. In this part, management and the users should not make decisions for them or say anything that takes the team off his path [25] [23].

**Sprint Review Meeting**

In the Sprint Review meeting the team presents to management, customers, users, Product Owner, and Scrum Master a demo of the product increment that they built during the Sprint. The Scrum Master is responsible for coordinating and conducting this meeting.

Only completed product functionalities can be demonstrated, and anyone present is allowed and encouraged to ask questions and do observations or suggestions to build a good feedback for the team [23] [24] [25].

**Daily Scrum Meeting**

The Daily Scrum meeting is a 15 minutes meeting that happens every workday. The team members synchronize their work and progress, and report any impediments to the Scrum Master. This is the meeting where the team comes to communicate.
In this meeting three main questions are answered by each member:

- What have you done since the last Scrum?
- What will you do between now and the next Scrum?
- What got in your way of doing work?

This meeting provides a lot of benefits: it improves communication, eliminates other meetings, identifies and removes impediments to development, highlight and promote quick decision making, and improves everyone’s level of project knowledge. The Scrum Master should establish a meeting place and time for the Daily Scrum [23] [24] [25].

2.1.3.4 Artifacts

The main artifacts that can be found in the Scrum methodology are: Product Backlog and Sprint Backlog.

**Product Backlog**

The Product Backlog is an evolving, prioritized queue of business and technical functionality with estimated times for each functionality. It is a list of all features, technologies, good ideas, enhancements, and bugs fixes for next releases. The estimations are in days, and include the time it takes to perform all of the requisite architecture, design, construction and testing [23] [24] [25].

**Sprint Backlog**

The Sprint Backlog is a list of tasks to execute during each sprint maintained by the Scrum team. These tasks are complete pieces of work needed to make a product increment. As tasks are worked on or completed, the hours of estimated remaining work for each task is updated.

The Sprint Backlog is a highly visible, real time picture of the work that the team plans to accomplish during the sprint [23].

2.1.4 ITIL

ITIL (the IT Infrastructure Library) is a comprehensive, consistent and coherent set of best practices for IT service management processes, promoting a quality approach to achieving business effectiveness and efficiency in the use of information systems [26].

At the core of the library are two volumes: Service Support and Service Deliver. Together they form the Service Management discipline. The set of books become complete with Planning to Implement Service Management, ICT Infrastructure Management, Applications Management, Security Management and The Business Perspective.
ITIL Service Support includes six disciplines that provide flexibility and stability for delivering IT services to the business. These six subjects are: Service / Help Desk, Change Management, Release Management, Incident Management, Problem Management and Configuration Management.

In ITIL Service Delivery have five disciplines that make available high-quality and cost-effective IT services for the business. The disciplines are: Availability Management, IT Services Continuity Management, Capacity Management, Financial Management and Service Level Management.

For this study the most relevant subjects are the disciplines of the Service Support. In the next section is presented Service Support and his disciplines in more detail.

### 2.1.4.1 Service Support

The Service Support deals with the day-to-day support and maintenance processes of Incident Management, Problem Management, Change Management, Configuration Management and Release Management plus the Service Desk function.

**Service Desk**

The Service Desk is an essential function for Service Management, which provides a single point of contact between the costumer and the IT services, handling all Incidents, queries and requests, and provides an interface for all other ITIL processes.

The main activities of the Service Desk are:

- **Responding to calls** - logs all received call. There are two distinct types of calls, incidents (error reports and service requests) and changes;
- **Providing information** - informs users about current or expected errors and provides information about new and existing services, provisions of the SLA’s (Service Level Agreements) and order procedures and costs;
- **Supplier liaison** - responsible for the contacts with maintenance suppliers;
- **Operational management tasks** – performs all the operational tasks, such as, making back-ups and restores, creating accounts, authorizing and resetting passwords;
- **Infrastructure monitoring** - uses tools to detect faults affecting essential equipment, normally these tools inform Incident Management automatically.
Incident Management

The Incident Management process is responsible for the management of all Incidents, from detection, recording through to resolution and closure. The main objective of Incident Management is to return business to a normal service level, as defined in an SLA, as soon as possible minimizing the adverse impact on business operations [31] [26] [30] [28].

An Incident is defined as: any event which is not part of the standard operation of a service and which causes, or may cause, an interruption to, or a reduction, in the quality of that service [26].

Incident Management deals with big amounts of Incidents. To maintain his efficiency some success factors must be taken into account, such as prioritization, updated CMDB, a good ‘knowledge base’, an efficient automated system, and a close link with SLM [29] [26].

The main activities performed during the Incident Management process are:

- **Incident acceptance and recording** – detects or reports an incident and creates a record for it;
- **Classification and initial support** – codifies the incident by status, type, impact, urgency, priority, SLA, etc. The user may receive information how to solve the incident or a work around for it;
- **Matching** - checks if the incident is known, and if it is related to an existing incident, problem or known error;
- **Investigation and diagnosis** – investigates the incident if there is no solution;
- **Resolution and recovery** – resolves the issue when the solution is found;
- **Closure** - asks the user if he is satisfied with the solution, then if he answers affirmatively the incident can be closed;
- **Progress monitoring and tracking** – monitors the entire cycle, and can occur escalation if necessary [26] [29] [27].

Problem Management

The objective of Problem Management is to reduce the adverse impact of incidents and problems on the business, and to prevent the reoccurrence of incidents related to these errors. To achieve this, Problem Management assists Incident Management by running the major incidents and problems, it also seeks to get to the root cause of the incidents, and then initiate its correction. Problem Management process also has a reactive and proactive side. The reactive side is concerned with solving problems as response to incidents. While the proactive aspect is concerned with identifying and solving problems and known errors before incidents occur [26] [28] [30] [31].

Problem is defined as a condition identified as a result of many incidents that reveal common symptoms. It can also be defined as an unwanted situation, for which the root cause is unknown. Known error is characterized as a condition identified by successful diagnosis of the root cause of the problem, and for which a provisional workaround has been found [26] [30]. The next figure shows the relation between Incidents, Problems, Known Errors and RFCs.
The main activities performed by Problem Management are:

- **Problem control** – responsible to identify the root cause and to provide the Service Desk with information and suggestions on work-around when available.
- **Error control** – accountable for processing known errors until they are eliminated by the successful implementation of a change under the control of Change Management process [30] [26].

**Change Management**

The goal of Change Management process is to ensure that standardized methods and procedures are used for efficient and prompt handling of all changes, to reduce the impact of changes upon service quality, and improving the daily operations. The process must also implement approved changes efficiently and cost-effectively, with minimal risk to the IT infrastructure [30] [26].

For this process we must understand two concepts, Request for Change (RFC) and Change Advisory Board (CAB). A Request for Change is defined as form used to record details of a request for a change to any CI (Configuration Item) within an infrastructure, or to services, procedures, and items associated with the infrastructure [26]. The Change Advisory Board (CAB) is characterized to be a group of people that can give suggestions to Change Management on the implementation of changes. This board is formed by people from IT and business field [26].

The principal activities performed during the Change Management process are:

- **Submission (recording)** - ensures that all sources of change can submit RFC’s and guarantee that they are effectively recorded;
- **Acceptance classification** - filters the RFC’s and accepts them for further consideration;
- **Planning and approval** - sorts the RFC’s by category and priority;
- **Coordination** - consolidates, plans and approves the development and implementation of changes. It is necessary to ensure that the require sources are available and during planning and approval activity the CAB must be involved when necessary.
- **Evaluation** - verifies if the change was successful and what were the lessons that we learned to improve the process [30] [26].

**Release Management**

The main objective of the Release Management process is to supervise the controlled distribution of software and hardware components into the live environment, ensures the successful rollout of releases, including integration, testing and storage. Its main concern is the protection of the live environment, ensuring that only tested and correct versions of certified software and hardware are
available. The Release Management process is closely related to Change Management process [31] [26] [30].

There are a few important terms to understand this process, such as Release, Definitive Software Library (DSL) and Definitive Hardware Store (DHS). The term Release is used for a group of problem fixes and enhancements to the service, that were tested and introduced into the live environment. Releases can be divided into Major software releases and hardware upgrades, Minor software releases and hardware upgrades and Emergency software and hardware fixes.

A Definitive Software Library (DSL) is a library containing all authorized versions of all software CIs. A Definitive Hardware Library (DHL) contains all spares and stocks of hardware [26] [27].

The principal activities of this process are:

- **Release policy and planning** - covers release numbering and frequency of releases;
- **Release design, building and configuration** - compiles and links application modules produced in-house and any bought-in software;
- **Testing and release acceptance** - concerns with functional, operational, performance, integration and acceptance testing;
- **Rollout planning** - extends the release plan adding details of the exact installation process and the agreed implementation plan.
- **Communication, preparation and training** – informs customers and support staff about what is planned and how it might affect them. This is done by training sessions, periods of parallel working and involvement in the release acceptance stage.
- **Release distribution and installation** – designs the distribution so that the integrity of software is maintained during handling, packaging and delivery. Produces back-out plans to document the actions to be taken to restore the service if the rollout of a release fails [26] [30].

**Configuration Management**

Configuration Manager Process provides the foundation for successful IT Service Management [28]. The principal function of this process is to identify, control, manage, define and verify the versions of Configuration Items (CIs) in a system [29] [26] [31].

Configuration Items are defined as IT components or services of an infrastructure.

The CMDB is a database that contains all the information related which CIs, and important relations between them [26] [30].

The major activities performed by Configuration Management are:

- **Planning** - determines the strategy, objectives and policy of the process.
- **Identification** - sets up the process to keep the database up-to-date;
- **Control** - authorizes which CIs are registered;
- **Status accounting** - stores current and historical details about the status of CI’s during their life cycle;
- **Verification** – checks the CMDB and the IT infrastructure to verify the accuracy of the records;
- **Reporting** - reports information to other processes and reports about trends and developments.
2.1.5 Summary

The previous subjects described the good practices and methodologies that are related with this study. These methodologies try to resolve some of the problems found in the Introduction chapter.

It is hard to implement the IEEE standard or ITIL methodology from scratch. The users had to adapt to a new reality, a new process and a new way of working. In spite of not having a documented and a defined process OutSystems has an implemented process. All the users know who to operate and execute their actions.

One of the biggest problems why these subjects for themselves don’t resolve the problems are related to the resistance to change. It is not easy to explain how the users must do their actions, it takes time to understand all the process and incorporate the changes in their habits.

Besides these good practices the maintenance process has to be complemented with a tool that supports it. The problems aren’t resolved only by implementing these good practices.

2.2 Tools

In this chapter some project management tools were analyzed. This analysis has the purpose to study if there was any commercial tool that could met all the requirements and needs of the perfective maintenance process.

Target Process, VersionOne, Rally Software and FogBugz softwares were chosen to be analyzed. This study was guided by the necessities of the perfective maintenance process and the engineering department. The key factors that were taken into consideration were:

- Customization of the process;
- Support for agile methodologies;
- Integration with previous systems;
- Good report mechanisms;
- Hierarchical structure of concepts;
- Web-based application;

At the end of each analysis a table summarizes the strengths and weaknesses of the presented tool.

2.2.1 Target Process

TargetProcess is an agile project management software. Designed with simplicity in mind, TargetProcess helps software development companies reduce the complexity of software project management and simplifies planning, tracking and quality assurance activities. TargetProcess supports SRUM, Extreme Programming and other agile processes [32].

The pros and cons are summarized in table 1.
Table 1: Target Process Pros and Cons

2.2.2 Version One

*VersionOne* is the leading project planning and management tool designed specifically for agile software development. It enables today’s most popular agile methodologies such as Scrum, Extreme Programming, DSDM and Agile UP. *VersionOne* helps teams simplify the process of planning, tracking, and scaling their agile development efforts. It scales to support multiple projects, releases, and teams, and incorporates all key agile and iterative management practices such as release planning, iteration planning and tracking, burndown reporting, velocity, and user story (or backlog item) and task management [33].

Table 2 shows the pros and cons that were found.

Table 2: VersionOne Pros and Cons

2.2.3 Rally Software

Rally unites agile project management with tracking of requirements, tests and defects, so everyone has a real-time picture of the project’s features and quality, priorities, roadblocks and risks. It provides support for all roles and activities in the software development lifecycle. Rally gives agile development organizations a centralized system to manage their product delivery lifecycle and ultimately increase the value they deliver to customers [34].

Table 3 summarizes the strength and weak points of this tool.

Table 3: Rally Software Pros and Cons

2.2.4 FogBugz

*FogBugz* is a complete project management system designed to help software teams communicate. It
helps them work together by tracking, prioritizing, and coordinating the thousands of small tasks a
development team has to do [35].

Table 4 presents the pros and cons of FogBugz.

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Wiki integration;</td>
<td>● Support for agile methodologies;</td>
</tr>
<tr>
<td>● Hierarchical structure;</td>
<td>● Customization according to a specific development process;</td>
</tr>
<tr>
<td>● Evidence-Based Scheduling (EBS) for estimating;</td>
<td>● Integration between existent systems;</td>
</tr>
<tr>
<td>● Planning take into account the holidays;</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: FogBugz Pros and Cons

2.2.5 Summary

By the previous analysis we can conclude that all the tools are web-based, have good report mechanisms, providing information about project and iteration progress, and concepts are organized hierarchically. Unfortunately all these tools don't satisfy two major requirements. None of them support customization of the perfective maintenance process and integration with previous systems.

Because none of the studied tools completely satisfy all the initial requirements we can conclude that they don't meet all the requirements and needs of the perfective maintenance process. Table 5 summarizes the evaluation of all project management tools.

<table>
<thead>
<tr>
<th>Target Process</th>
<th>VersionOne</th>
<th>Rally Software</th>
<th>FogBugz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customization of the development process</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Support for agile methodologies</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Integration with existent systems</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Report mechanisms</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Hierarchical structure of concepts</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Web-based application</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

✔ Supports  ✗ Does not support

Table 5: Evaluation of the Project Management Tools
3 Case Study / Problem

During the past months we studied the perfective maintenance process of OutSystems, a multinational software company which provides the industry leading All-in-One Agile Platform for rapid delivery and management of web business applications that are built for continuous change.

Most of the problems found in the case study are similar to the research area problems. For example, undocumented and ineffective process and the use of inappropriate management tools. The next sections describe the case study problems in more detail.

3.1 Wrong and Ambiguous Terms

The first problem that we found during our project was the ambiguous or wrong definition of some essential terms used in the software industry. One case was Version sometimes was used for talking about projects and other times were used to talk about a specific version of the OutSystems Platform. Another example was the term Release which was not used at all.

Sometimes this ambiguity of terms created communication problems and misunderstands by the participants of the process. In worst cases these difficulties could delay all the perfective process and increase its costs. If we do not have a common knowledge base, shared and understood by all the members of the process, the previous problems will continue to exist.

3.2 Undocumented and Ineffective Perfective Maintenance Process

Another problem that we faced during our investigation was an undocumented and ineffective perfective maintenance process. Most problems that are associated with software maintenance can be traced to deficiencies of the development process [12].

An undocumented process is characterized by a process that exists and works but it is not supported by written evidences. When a process is ineffective, means that it is working but it is not producing the best results. Because of these problems, most of the projects were not delivered in time and the final budget of the project was usually much greater than the initial prediction.

A documented and effective software development and maintenance process is essential in today's businesses. Without it problems will arise. If the maintenance process is not documented communication difficulties may occur. It is hard to explain how the process works without a clearly idea of each step. With a documented process is easier to improve it, eliminating useless and redundant stages. Only with a defined process is possible to achieve the higher levels of the CMM making the process more effective and efficient resulting in lower costs of the total project's budget and shorter development cycles.

In today's word, if a company wants to be competitive must have an effective and documented perfective maintenance process.
3.3 Inappropriate Management Tool

The use of an inappropriate tool to manage the perfective maintenance process was one of the difficulties found during our case study. The system was obsolete and didn't support the user and the process needs.

The main problems revealed by the tool were:

- Lack of visibility over the tasks;
- Weak hierarchal organization;
- Confusing an unclear concepts;
- Few control mechanisms;

Besides these problems, the application was used by all sectors of the company (maintenance, support, help desk, development). This made that its modification extremely complicated and difficult.

The management tool must reflect the perfective process and at the same time be agile enough to incorporate new requirements and changes that the process may suffer over time.

3.4 Integration Between Multiple Systems

Another challenge faced during our investigation was the integration between multiple systems, such as in house systems and commercial systems.

One of the main requirements for the new system was the integration with other systems, Issue Manager and Time Manager (in house systems) and Google Spreadsheets (commercial system).

Issue Manager is an application that stores all the information about defects, requirements, tasks, projects and support cases related with the OutSystems Platform. All this information has to be used by the new system, and all the new information created by it must be compatible with the old system, Issue Manager.

Time Manager is the application responsible for the vacations of the OutSystems' employees; it also has information about weekends and holidays. This information must be used by the new system when the resources' allocation is made.

Google Spreadsheets is used for the allocation of project resources. The new application must collect the information in the spreadsheet and use it to calculate the projects' budget.

This was an obligatory requirement for the new system. With this integration the perfective process become more effective since the total time of the process is reduced.

3.5 Summary

The problems found in the case study are similar to the ones found in the research area problems. Undefined and undocumented process is presented in both situations. Studies have shown that the knowledge of the perfective maintenance process is essential for the professionals to perform more effectively. OutSystems had a process but it was undocumented and undefined. It was hard to improve the perfective maintenance process because they don't have physical evidences about how
they operate.

Another problem found in the case study that is present in the research area problems is the use on inappropriate management tools. Tools that did not reflect the process were being used by OutSystems professionals. This reduced their productivity and delayed the whole process.
Proposal

Based on the problems described in the previous section we divide our proposal in three distinct sub-proposals:

- Creation of a common knowledge base of software engineering terms understood by all the human resources in the engineering department;
- Elicitation of the perfective maintenance process and its formalization;
- Creation of a new management tool that reflects the process and the perfective maintenance needs.

Naturally this proposal is not done at once. We separate the problem into smaller problems. First, we start with the software engineering terms, and then we continue with elicitation of the perfective maintenance process and terminate with the creation of the new tool. The proposal is periodically evaluated during its development by OutSystems’ members. They provide useful feedback which is incorporated in the proposal improving its final result.

4.1 Software Engineering Terms

During the investigation about the state of the art in maintenance it was acquired a wide knowledge about the different approaches and their software terms.

We propose to create a common knowledge base of terms by comparing the terms found in the previous study with the terms used by OutSystems. Then we check which ones are being used in an ambiguous or incorrect way. The terms that are not being used are introduced improving communication.

The terms will be used in the formalization of the process and will be incorporated in the new version of the management tool.

4.2 Perfective Maintenance Process

The elicitation of the perfective maintenance process is one of the critical points in our study. It is extremely important to have a defined process to improve the management visibility into progress, increase the control mechanisms over the process and provide a common language and a shared vision [36].

Our first proposal to approach the problem is to inspect the database logs from the tool (Issue Manager) used by the developers. We use the ProM tool which is an extensible framework that supports a wide variety of process mining techniques [37] to obtain the process.

Our second approach is more traditional. We propose to interview the most important professionals present in the process, asking what they do, what are the main actors in each step of the process and how their vision of it is. Then we gather all the feedback from the interviews and define the perfective maintenance process.
4.3 Management tool

Besides the elicitation of the perfective maintenance process it is necessary to have a tool that reflects it. It is also important that the tool matches all the needs of the software development team, making them more productive and effective.

The proposal for the new system is characterized by the following requirements:

- Compatibility with agile methodologies, in particular SRCUM;
- Adaptability with existent systems;
- Easy to change;
- Good control and report mechanisms;

Another important requirement is that all the best practices studied in the state-of-the-art chapter must be present in the final system.

4.4 Summary

These three proposals are strictly related with each other. The output of each sub proposal is used as input in the other sub proposal.

The result of the definition of Software engineering terms served as input for the Perfective Maintenance Process and for the Management Tool. The Perfective Maintenance Process output provided essential information for the Management Tool.

Joining all the results it is obtained a more complete and correct study.
5 Implementation / Prototype

This chapter explains in more detail the implementation and the prototype of the solution. It starts with the particularities about the engineering software terms’ implementation. Then, the details how the perfective process was elicited are explained and, finally, the specifications and particularities about the implementation of the management tool are revealed.

5.1 Software Engineering Terms

To have an efficient communication it is necessary that the users share the same knowledge about the same terms and concepts used in maintenance. Communication plays an important role during the process.

To understand what was the knowledge about the essential terms used in the software engineering field and to verify if they were being correctly used a few interviews were performed to the engineering professionals.

After completing these interviews the terms were aggregated to form a single definition which reflected all opinions. Then the created definitions were compared with literature terms, to understand which ones were being improperly used or not used at all. Finally it was created a new definitions based on literature terms and in the state-of-the-art studies.

These terms are shared across the engineering department and understood by its professionals, improving their communication methods. The final definitions will be used in the formalization and definition of the process and in the new management tool.

5.2 Perfective Maintenance Process

The knowledge of the processes is essential in today's companies if they want to become more competitive and perform more efficiently. In our case study we followed two different approaches to find a solution for the problem. The first one is characterized by the review of the database logs with ProM tool. In the second approach we interviewed several members of the engineering department to elicit the perfective maintenance process.

5.2.1 Process Mining Approach

In the first approach was used a process mining technique. Unlike classical data mining techniques with this one the focus is on processes. Process mining techniques can extract information from event logs.

It was used one of the most known tools, ProM, to analyze the database logs from Issue Manager. ProM is an extensible framework which offers a wide variety of process mining techniques. It allows importing from and exports to a wide variety of formats and systems (ranging from enterprise information systems and workflow products to social network software and classical mining tools) and provides advanced visualization and verification capabilities [38].
Issue Manager is an internal tool used by OutSystems for managing the software development process. This tool stores all the information about change requests, new requirements, bugs, tasks and support cases. It is also possible to know the actual state of each task, giving us a real picture of the progress of the process.

Since the creation of a task until its closure it suffers a few status changes. Analyzing the logs resulted from that changes we expected as result a definition of the perfective maintenance process.

In Issue Manager the most important status that we have are:

- **New** – when a task was created and was not yet analyzed;
- **Assigned** – when a task was analyzed and was assigned for resolution;
- **Open** – when someone is working on the tasks;
- **Resolved** – when the work is completed and task is resolved.

Figure 12 represents a regular status transition that tasks experience over time.

![Regular status transitions during a task life cycle](image)

Unfortunately the use of ProM is not so straightforward, it only reads a specific file, MXML. We had to convert our logs into that format.

First we had to clean the data from Issue Manager's logs. We deleted irrelevant information such as comments, iteration changes and other unnecessary changes. We kept the task Id, status, first date and last date and other necessary information to create the Microsoft Access database. We only considered tasks that had at least four status changes.

The next step was the creation of the database, according to the specific structure: four tables with a similar structure as the fields in the MXML format. The tables are:

- **Process_Instances** - which needs to be filled with the identifier of a certain process instance and, if available, its accompanying description[39];
- **Data_Attributes_Process_Instances** - needs to be filled with additional information about each process instance, the so called data attributes[39];
- **Audit_Trail_Entries** - needs to be filled with data about tasks that have been performed during the execution of a process instance[39];
- **Data_Attributes_Audit_Trail_Entries** – needs to be filled with additional information about each audit trail entry [39].

Figure 13 shows the ER diagram for the four process mining tables.
After filling the tables with the necessary information we used ProM Import Framework to covert the new access database to the MXML format.

Finally we opened the MXML file with ProM and applied the $\alpha$ algorithm. The $\alpha$ algorithm transforms a log $W$ into a Petri net ($Pw$, $Tw$, $Fw$). It only uses basic mathematics, the relation $>_w$, $\rightarrow_w$, $\parallel_w$ and $\#_w$, and the functions first and last to get the first and last element from a trace [40]. For a more detailed explanation about the $\alpha$ algorithm consult [40].

In conclusion, before using ProM Framework we had to create an Microsoft Access database from the logs and converted it to the MXML format using ProM Import Framework. Figure 14 summarizes all the steps.

5.2.2 Interview Approach

In our second approach we used a totally different method from approach one. We chose the interview method, which belongs to the conversational category [41]. This technique is very popular in requirement elicitation, and can be used for process elicitation.

Conversational category is characterized by spoken exchange of thoughts, opinions, and feelings. Conversation is effective to develop and understand the problems and to elicit generic processes, because is a natural way to express needs and ideas, and ask and answer questions. Methods in this
category are also known as verbal methods. Other methods within this category include focus groups, brainstorming, workshop, etc. However, all these methods are labor intensive. Preparation of meetings, transcript producing and analyzing information from a live interaction can consume a lot of time [41].

In interview methods an experienced analyst with generic knowledge about problem domain discusses the problem with different group of people and builds up an understanding of their explanations. If the interview is conducted with pre-defined agenda and questions, it is called structured interview; otherwise, it is an open-ended interview [41]. In our study we used the structured interview.

First we started with the interview preparation and meeting schedule. We decided what questions were necessary for the interview. We asked two main questions to all the interviewees: how they think how was the process, and what were the main actors in each step of the process.

For the meeting scheduling we had to take a few of factors into account:

- Compatible agendas between the interviewer and the interviewee;
- Rooms' availability, before any interview we had to book a room with a calm environment where the interviewee could concentrate in our questions;

To obtain different points of view and impartial opinions we interviewed different people with different roles within the organization. The selected roles were:

- **Manager** – responsible for managing all the projects in the engineering department;
- **Team leader** – in charge of a development team and responsible for making project decisions with the team;
- **Developer** - person who designs and writes software code.

During interviews if we thought we need more information about a specific subject we asked about it, beside our pre-defined questions.

Finally, when all the interviews were completed all the documents were gathered and a new document with the perfective process was created based on common information collected in the previous interactions. After the document was finalized, it was sent to all participants in the interviews. Then they validated the new documented perfective maintenance process and gave useful feedback about things that must be changed in the final document.

Figure 15 summarizes all the steps in this approach. It starts with the interviews preparation and meeting scheduling, then continues to the interviews and to the creation of the document, ends with the validation of the process.

![Figure 15: Interviews Steps](image)
5.3 Management Tool

Most of today's software development projects slipped, exceeding their initial budgets and schedules. To overcome this problem it is necessary to have a project management tools that match the needs of the maintenance process, specially the perfective maintenance process.

In order to have efficient and usable tool by the users a few requirements had to be elicited. The main necessary points that the application must had were:

- **Customization of the perfective process**, since the process is not a static entity and evolves over time the new system has to have ability to adapt and reflect that changes;
- **Support for agile methodologies**, OutSystems uses an agile methodology, SCRUM, in its development so is essential that the new tool supports it;
- **Integration with previous systems**, it is mandatory that the new system can interact with existent applications and use their information;
- **Good report mechanisms**, the new application must have rich report mechanisms to provide a better control over the process and increase its visibility;
- **Hierarchical structure of concepts**, to have detailed control over the work the concepts must be clearly organized;
- **Web-based application**, it provides a whole picture about the progress of the projects to everyone in the engineering.

Other requirements were taken into account such as easiness of resources allocation and feedback messages. All the basic user interface necessities were considered, mainly simplicity and ease of execution.

5.3.1 Methodology

During the development of our system the Scrum methodology was used. This agile methodology is used by all the OutSystems’ professionals in the engineering division and it is guided by the following practices:

- Iterations of two weeks;
- Daily meetings, with a duration of 15 minutes where were discussed the progresses and difficulties faced during the development;
- Meeting at the end of each iteration, where the stakeholders gave feedback about the product increment.

Before doing the final interfaces a few mock ups were made and validated with the users.

5.3.2 OutSystems All-in-One Agile Platform

The technology used to create the tool was the **OutSystems All-in-One Agile Platform**. The choice was based on the fact that most of the systems were done with this technology and the integration between them will be easier.
This technology is also a good choice for the following points:

- Visual primitives;
- Quickly development;
- Easy to change.

5.3.3 Domain Model

The final domain model for this application is complex. Entities from the previous system were used and necessary ones were created. Joining the old and the new model we have a more complete and rich model, which satisfy the development and process needs. Figure 16 shows a simplified version of it, describing the main entities and attributes involved and their relationships.

The entities Product, Project, Task, and Iteration already existed in the previous application. The necessary attributed were added making them more meaningful. The Release entity was created improving the way projects were organized and planed.

This simplified domain model can be considered as the core structure for the new management tool.

5.3.4 Hierarchical Structure

An organized structure of concepts is essential in a project management tool. It increases the control over the working items (releases, projects or tasks), and also improves the visibility over them.

The previous management tool (Issue Manager) used by OutSystems had a poorly hierarchical structure of concepts, which did not satisfy the perfective maintenance process needs. The new hierarchical structure was based on the state-of-the-art studies, previous analyzed tools, elicited terms and perfective maintenance process.

The final structure is formed by the following concepts: Product, Release, Project, Activity and Tasks. Figure 17 presents an example of the final hierarchy and their relations.
With this structure a Product can have multiple Releases, while a Release can only belong to a single Product. A Release has one or more Projects; however a Project is only present in one Release. A Project must have more than one Tasks, and them are only associated with one Project. Tasks can have child tasks. The parent tasks are called Activities and they group related tasks. These hierarchal organizations of concepts provides the ability to organize and have a better control over the releases, projects, activities and tasks, making them more manageable.

5.3.5 Graphical Interface

The new project management reflects the perfective maintenance process needs as well as the user needs. It is focused only in essential elements and displaying only the necessary fields. A lot of management improvements were made in the new application to increase the control mechanisms and management visibility over the process.

5.3.5.1 Add and List Releases

The application interface provides a simpler way to create Releases. Focusing in essential elements and displaying only the necessary fields. Figure 18 shows the interface for adding a new Release.

As seen, there are only two obligatory fields (Name and Product), marked with a vertical orange bar. The products can be selected from the combo box. These products already existed in the previous application, so any product created in the past are available in the new project management tool. The Release can be complemented with additional information from the non-obligatory fields, such as...
Description, Begin Date, End Date and Active.

Besides the graphical interface for adding a new release it was also created an interface to list all releases. This screen provides a global view over all releases, showing the most relevant information of each one. Figure 19 represents the graphical interface for listing releases.

Releases:
List Releases | Create New Release

Product: <select>
Active Only: [on/off]
Apply Filter

<table>
<thead>
<tr>
<th>Name</th>
<th>Begin Date</th>
<th>End Date</th>
<th>Product</th>
<th>Number of Projects</th>
<th>Actions</th>
</tr>
</thead>
</table>

*Figure 19: List Releases Graphical Interface*

It is also possible to select particular releases, based on the filter criteria. Releases can be filtered by Product and Active status (Releases that are not completed). For each release it is also shown complementary information, like Begin Date and End Date, the Product that the release is associated with, the Number of Projects within the release and Actions that can be executed over it.

5.3.5.2 Add, Associate and List Projects

Adding a project to a release is very simple and effective. The only mandatory field is the Name of the project, visible by the vertical orange bar. After its creation the project becomes associated with the release that was previously selected. If necessary the information about the project can be completed with a Description. Figure 20 shows the graphical interface for adding a new Project.

*Figure 20: Add New Project Graphical Interface*

While adding a new project the context is never lost. All the information that was previously available stills accessible during the creation. Every project created with the new project management tool is compatible with the older system; this means that any new project can be accessed by it.

It is also possible to associate projects that were created with the older system to a specific release. Figure 21 presents how the previous projects can be added to the current release.
The older projects can be filtered by Name and Description and can be associated to a release by clicking in the green plus icon. For each project it is also shown additional information, for instance Description, Budget and Start Date.

Moreover, it was created a graphic interface to display all projects. This screen makes possible to have a general view about all the projects that are related to a release, presenting the most important information of each project. Figure 22 shows the graphical interface for listing all the projects within a release.

The information about the release that is associated with the projects can be found at the top of the interface. The projects can be filtered according to a criterion. Name, Description and other advanced options, such as Start Date, Estimated Close Date and Open status, can be used to filter them. For each project it is possible to know the Budget, Estimated Close Date, the actual progress and if the project is already completed. If a project has description it is shown below its name.

This interface gives a better control over the perfective process. It is possible to verify the actual progress of each project. Enabling the ability to predict with more precision if the release is on time and budget. This was not possible using the old tool since there was no release concept and no clean way to compare multiple projects.

5.3.5.3 Add, Edit and List Activities and Tasks

The graphical interface for adding a new task is simple and it is focused only in the necessary information to create a task. This interface plays a crucial role in the perfective maintenance process.

One of the main requirements was to make the adding process quicker and simpler. There are only three mandatory fields, Name, Assigned to and Kind, marked by the vertical orange bar. It is possible to add additional information about the task Description and its Estimated Duration. It is provided two
different methods to estimate the duration of the task: single duration and 3-Point estimate, based on best, likely and worst case duration.

Figure 23 shows the graphical interface for adding a new task, including the 3-Point estimate.

![Add Task Graphical Interface](image1)

**Figure 23: Add New Task and 3-Point Estimate Graphical Interface**

While adding a new task the context is never lost. It is possible to know the project and their tasks. If the task has a parent (*Activity*) it is shown at the top of the interface. After adding a task it becomes associated with the project that was selected before. All the tasks created with the new application are consistent with the older system, and can be accessed by it.

The edit task screen was also redesigned. This interface was build using all the feedback provided by the users. Only the most frequently changed information is shown, making editing action simpler. Figure 24 shows the graphical interface to edit a task, and the default fields.

![Edit Task Graphical Interface](image2)

**Figure 24: Edit Task Graphical Interface**

The principal fields for editing a task (*Assigned To, Kind, Status, Iteration* and *Comment*) are visible in this screen. It is also possible to do advanced operations, such as changing the priority of the task, by clicking in the link at the top of the interface. The tasks can be added and removed from projects using the link available in the higher area of the screen. When the edit action is finished a yellow background appears around the task in the tree view and all the executed changes are logged.
Additionally it was created an interface to show all the tasks and activities related to a project. This screen shows the most important information about each task. Figure 25 presents the graphical interface for listing all working items within a project.

![Figure 25: List Tasks and Activities Graphical Interface](image)

The items (activities and tasks) can be organized hierarchically. The relevant information, such as Id, Name, Active effort and Actions available (add child, add sibling and edit it with the older tool) for each activity or task are shown by this interface. It is also possible to identify the status of each task by its icon. Figure 26 presents all the possible icons for a task. Activities are identified by the folder icon, and can be created by adding a child task to a task without children. This transforms a task into an activity.

![Figure 26: Status icons for the Tasks](image)

It is also possible to organize the items using drag-and-drop. This provides a better way to arrange them. For example: putting the most important on the top of the stack.

### 3-Point Estimate

Another important screen that was created was the 3-Point Estimate. This interface is especially useful for the estimations of the tasks in the perfective maintenance process. It provides an easy way to insert all the estimations for the tasks, based on the 3-Point technique. Figure 27 shows the graphical interface for the 3-Point estimate.

![Figure 27: 3-Point Estimate Graphic Interface](image)

All the essential information is available to perform this action quickly and correctly. The background color changes if the value in the field is changed. This gives the user the ability to quickly identify all...
the variations. The final estimate is automatically update while the best, likely and worst case are added. The total effort of the parent task (activity) is also updated while the values are introduced.

5.3.5.5 Iterations

Other enhancements in the interface were made to provide a better control over the iterations. It was created a graphical interface where it is possible to filter the tasks and activities according to a criterion. Figure 28 presents a sample of the interface and the available criteria.

The first five options are present in all projects. This first one shows all tasks and activities. The second one filters the working items of the project since the first iteration until the previous iteration, inclusive. The following two choices have a similar behavior, except the tasks and activities are filtered until the current and next iteration respectively. The fifth option only presents items without iteration. The next options vary by project. They show solely the tasks and activities that are assigned to the selected iteration. Another interface was created to solve the difficult of managing different kinds of tasks, development and documentation task. This problem exists because the knowledge transfer team is independent and their effort is shared by all other teams and projects. The iteration view was changed to include two separated progress bars, one for development and quality assurance tasks and another for documentation tasks. Figure 29 presents the graphical interface of the iteration's progress.

This view provides an easy way to control the iteration's progress of both tasks kind. It also gives additional information (Name, Duration and Status) about the tasks assigned to the iteration. These interfaces increase the control over the iterations, facilitating the management of them. They are particularly useful in the iteration planning and implementation phase.
5.3.5.6 Priority

In order to know which tasks and activities had to be implemented first it was created a graphical interface to show all the information related to the priority of the working items within a project. Figure 30 presents an example of that interface.

The priority of the tasks and activities are shown by a green tick in the respective column. The working items can also be filtered according to their priority, using the check boxes below the priority names (Must, Should and Could).

This interface is a helpful resource for the users, especially during the implementation phase. It makes easier to know which task must be made first and give the ability to organize and plan the work more efficiently.

5.3.5.7 Resources Allocation and Project's Budget Update

One of the main breakthroughs that the new tool provided is related with the resources allocation and project budget calculation.

With the old application each project had its own spreadsheet with the available budget for each day and information about the iterations. To fill the spreadsheet it was necessary to check when the weekends, holidays and vacations of each user were. All these steps were made manually. Another big disadvantage of this approach was the difficult to manage resources between different projects and the maintenance effort to keep all the different project spreadsheets updated.

Now with a few configurations this activity is performed automatically. The roles must be associated and a default budget must be set for each user. It is also necessary to relate the project with its name in the Google Spreadsheet and choose a responsible email.

The graphical interface to set a role and a default budget for the user is very simple. The User and the Role are selected from the drop boxes, and the Default Budget (available budget for a work day) is chosen. At the moment it was only necessary two roles (Developer, which includes the quality assurance role, and knowledge transfer), but the systems is done to support more in the future. Figure 31 shows the graphical interface to associate a role and a budget with a user.
Additionally it was created an interface to list all the relations (user-role-budget). This interface provides the ability to delete and edit a relation that was previously created. All the changes are recorded, and the historic can be accessed when necessary. Figure 32 presents the graphic interface for listing the user-role-budget relations.

![Figure 32: User Role Budget Relation Graphical Interface](image)

Then it is essential to relate the project with a spreadsheet. The name that is used in the spreadsheet to refer the project must be written in the configuration field, and it is necessary to select the Google spreadsheet from the drop box. The responsible email is also very important. This email is used to send the reports about new vacations and changes in the project's budget. Figure 33 shows the graphical interface that is used to associate a project with a spreadsheet.

![Figure 33: Associate a Project with a Google Spreadsheet Graphical Interface](image)

Now, a pre-configured Google Spreadsheet (first row has the users' login and first column has the dates) is filled automatically with weekends, vacations and holidays associated with the users. These information is imported from Time Manager, which is an application used by OutSystems' professionals to book their vacations. This tool also has information about the weekends and holidays. Then the spreadsheet can be completed with information about the allocation of resources for each project. The project names were previously configured are written in the columns associated to the user. This means that during that day the user is allocated to the project with a default budget. The spreadsheet also supports exceptions such as different roles and different budgets associated with the user during a particular day.

The knowledge about the duration of the iterations is also present in the document. This is possible by writing the project's name in the first row and filling the columns with information about the end of each iteration.

Figure 34 presents an example of an allocation sheet with weekends, vacations, holidays, resource allocation and iteration information. A few exceptions can also be found in this figure. For instance user#2 has different role and budget in Project1 during 6th of August, user#1 has a different role for
Project3 at 7th of August and user#4 has a different budget for project2 in the same date.

<table>
<thead>
<tr>
<th>Date</th>
<th>user1</th>
<th>user2</th>
<th>user3</th>
<th>user4</th>
<th>Project1</th>
<th>Project2</th>
<th>Project3</th>
</tr>
</thead>
<tbody>
<tr>
<td>30/09/01</td>
<td>Project3</td>
<td>Vacation</td>
<td>Vacation</td>
<td>Project2</td>
<td>End of iteration</td>
<td>End of iteration</td>
<td></td>
</tr>
<tr>
<td>30/09/02</td>
<td>Project3</td>
<td>Vacation</td>
<td>Vacation</td>
<td>Project2</td>
<td>End of iteration</td>
<td>End of iteration</td>
<td></td>
</tr>
<tr>
<td>01/10/01</td>
<td>Vacation</td>
<td>Vacation</td>
<td>Vacation</td>
<td>Project2</td>
<td>End of iteration</td>
<td>End of iteration</td>
<td></td>
</tr>
<tr>
<td>02/11/01</td>
<td>Project3</td>
<td>Project5</td>
<td>Project3</td>
<td>Project1</td>
<td>End of iteration</td>
<td>End of iteration</td>
<td></td>
</tr>
<tr>
<td>03/12/01</td>
<td>Project3</td>
<td>Project3</td>
<td>Project3</td>
<td>Project1</td>
<td>End of iteration</td>
<td>End of iteration</td>
<td></td>
</tr>
<tr>
<td>04/13/01</td>
<td>Project3</td>
<td>Project3</td>
<td>Project3</td>
<td>Project1</td>
<td>End of iteration</td>
<td>End of iteration</td>
<td></td>
</tr>
<tr>
<td>05/14/01</td>
<td>Project3</td>
<td>Project3</td>
<td>Project3</td>
<td>Project1</td>
<td>End of iteration</td>
<td>End of iteration</td>
<td></td>
</tr>
<tr>
<td>06/15/01</td>
<td>Holiday</td>
<td>Holiday</td>
<td>Holiday</td>
<td>Holiday</td>
<td>End of iteration</td>
<td>End of iteration</td>
<td></td>
</tr>
<tr>
<td>07/16/01</td>
<td>Weekend</td>
<td>Weekend</td>
<td>Weekend</td>
<td>Weekend</td>
<td>End of iteration</td>
<td>End of iteration</td>
<td></td>
</tr>
<tr>
<td>08/17/01</td>
<td>Weekend</td>
<td>Weekend</td>
<td>Weekend</td>
<td>Weekend</td>
<td>End of iteration</td>
<td>End of iteration</td>
<td></td>
</tr>
</tbody>
</table>

Figure 35: Example of an Allocation Sheet

With all the necessary information introduced, the budget of each project is calculated based on the data in the spreadsheet. The duration of each iteration is also determined during this action. This operation does not have to be done after filling the allocation sheet; it can be done when necessary.

Figure 35 summarizes all the steps since the import of non-working days to the budget calculation.

Automating these activities has decreased the time spent in the resource allocation activity. Now it is only necessary to set a few configurations and manage a single spreadsheet with all the relevant information. Controlling resources between projects becomes easier since the information is centralized in a single document.

5.3.5.8 Feedback and Reports

The main important operations are preceded by feedback messages about the result of the operation. This increased the control over the performed actions. Figure 36 present an example of the system feedback messages.
The report mechanisms were another crucial point. Every night the system seeks for differences in the budget. If any difference is found an email is sent to the manager of the project. This report has information about changes in the spreadsheet (for instance new vacations) and details about the changes in the budget, such as previous budget and actual budget, new and suppressed iterations, development and documentation budget. The message present in the email is similar to the one in the feedback message shown in figure 36.
6 Results

This chapter describes the results of the Software Engineering Terms and the elicited Perfective Maintenance Process, using both approaches process mining techniques and Interview method.

6.1 Software Engineering Terms

During the study of the engineering terms used by OutSystems it was identified a few ambiguities. For example version and release terms were sometimes confused. That was the principal communication problem found during the interviews. The following definitions show what were the definitions used by the OutSystems professionals:

• **Version** – (1) related with revision, if patch is made a new version is created. (2) Project version, to distinguish projects on a particular context. (3) Is a major marketing version (4.1; 4.2; 5.0), two years of support obligation.

• **Release** – Associated with a major marketing version (4.0;4.1).

It is possible to verify that version had more then on meaning. Version was also used to talk about projects and releases.

Besides these wrong definitions, there were a few other ones that were correctly used. For example:

• **Sprint / Iteration** – fixed period of time, not too long, to show a part of the feature. It has detailed plan and the stakeholders are defined.

• **Team** – group of people, each of them has a budget associated with a role.

• **Role** – role of a person in the team (Manager, QA, Developer, Documentation).

The next step was clarifying the wrong terms based on literature terminology. The result is presented by the following definitions:

• **Release** - a collection of new and/or changed items which are tested and introduced into the live environment together [26].

• **Project** - a concerted effort that is focused on developing or maintaining a specific software product or system. A project has a scope, structure, budget and delivery schedule [42].

• **Task** - the smallest accountable unit of work. A task is the lowest level of work division typically included in the Project Plan and Work Breakdown Structure. Related tasks are usually grouped to form activities [43].

• **Sprint / Iteration** - Fixed period of time, not too long. In the end of each sprint a product increment is presented to the product owner and any other stakeholders that wish to view it. [44].

These redefinitions helped the communication and clarify the hierarchal structure of the terms. Most of the engineering terms obtain with this study were used in the definition of the process and in the new management tool.

6.2 Perfective Maintenance Process

The results obtain from both approaches are completely different. First the details outcome of the first approach (Process Mining) is show. Then the particularities of the results obtain using the interview
technique are presented.

### 6.2.1 Process Mining Approach

The results obtained with *ProM* were not very promising. The outcome didn't present the expected transitions. A lot states did not have any transitions, for example *Not Approved*, *Not Resolved*, *Needs Specification* and *Postponed*. The other ones which had transitions did not represent a valid sequence through the task's life cycle, for instance *Needs Verification*, *Assigned*, *Open* and *Resolved*. Figure 37 shows the result obtained with *ProM* using \( \alpha \) algorithm.

![ProM result using \( \alpha \) Algorithm Result](image)

Unfortunately it was not possible to try other process mining approaches and algorithms because of time constrains and changes in the work orientation.

### 6.2.2 Interview Approach

After validating the final document containing the process with the users, it was obtained a macro process of the perfective maintenance process.

As known, perfective maintenance is characterized to be the changes made to a system to add new features or to improve performance. Because of that the perfective maintenance process covers all the common activities of software development, such as requirement elicitation, architecture and design, implementation, testing and deployment.

The macro process can be divided into three major phases:

- Requirements, High Level Architecture and Design;
- Architecture, Design and Implementation;
- Testing and Deployment.

In the first phase the requirements elicitation and the high level architecture are done. The features to include in the next release are also chosen at this point.

The architecture, design and implementation stage is characterized by the drill down of the features
and where all the implementation steps (coding, documentation and quality assurance) are executed. The iterations of the perfective process are performed during this phase.

The last step of the macro process is where the tests are executed and all the necessary material for the knowledge transfer is produced.

All of these three phases have their own micro process. Figure 38 shows the macro view over the perfective maintenance process used by OutSystems.

**Figure 38: Macro view over the Perfective Maintenance Process**

### 6.2.2.1 Requirements, High Level Architecture and Design Phase

Requirements, high level architecture and design are characterized to be a management phase, where the scope and all the features for the next release are selected. This phase is formed by five steps:

- Choose the subject for the release;
- Guidelines for the release;
- Requirements’ elicitation;
- High level specification of features;
- Choose the features for the scope’s release.

In the first step the subject and the date for the release are chosen. The main actors at this point of the process are top level management and marketing.

The Guidelines for the release are where the goals and set of improvements to be achieved in the next release are defined.
The third step is characterized to be iterative. At this stage the requirements for following release are elicited, and the necessary ones are included.

The fourth step is also iterative, where the high level specification and estimation of the features are performed. Normally these estimations are in days or weeks. When the tasks of this stage are completed is decided which features must be incorporated in the next release.

In the last stage the features that will form the scope of the release are selected. The main actors of the last four steps are top level management, product manager and architect.

Figure 39 presents the process and all the principal actors of the requirements, high level architecture and design phase.

6.2.2.2 Architecture, Design and Implementation Phase

This phase of the macro process is formed by two processes. The first one is more focused in the architecture and design of the features, while the second is more focused in the implementation of the features.

The architecture and design process is formed by five steps:

- Resources allocation;
- Architecture and specification;
- Specifications’ drilldown;
- Estimations;
- Review.

The first step is where the resources are allocated. The teams are formed and the daily allocation
plans for each resource are made.

In the second stage of the process the architecture and specification of the features are detailed, and their impact is analyzed. The main actors at this point are product manager, architect, and client of the feature and team leaders.

During the drilldown of the features the tasks and activities are created and organized into a hierarchical schema. The principal actors of this step are architect, product delivery manager and team leaders.

In the estimations step all the team members get together and estimate the tasks created before in more detail (hours and days). The participants in this stage are the team members.

At the last step all the architecture, design, specifications and estimations are reviewed and if they are detailed enough advance to the implementation phase, otherwise the steps are repeated starting from the architecture and specification stage. The main actors in this step are managers, architect, client and product manager.

Figure 40 shows the principal actors and the first process of the architecture, design and implementation phase.

As seen before, this second process of the architecture, design and implementation phase are focused in the implementation of the features. It is formed by four steps:

- Coding;
- Usability Test;
- QA test / Unit tests;
- Documentation.
The first stage is characterized by the coding of the features that were previously selected. The usability and QA/unit tests are performed over the produced code. These steps start before the end of coding.

The documentation step is done along with the tests. This stage also starts before the ending of the coding phase.

When all these steps are completed the feature is finalized. The participants present in this second process are the development team and the documentation team.

This stage of the macro process includes the iterations that characterize the agile development process.

Figure 41 presents the main actors and summarizes the second process of the architecture, design and implementation phase.

6.2.2.3 Testing and Deployment Phase

Testing and deployment is the final phase of the perfective maintenance meta process. This phase is constituted by seven steps:

- Upgrade tests;
- Scalability tests;
- Knowledge transfer materials;
- Beta version;
- Beta support;
- Release candidate;
- General availability.

During the first two stages upgrade and scalability tests are performed. The major actors of these steps are the developers and the QA team members.

When the previous tests are completed the knowledge transfer material is produced. Videos and tutorials are made to show the features and the potential of the new release. The participants of this step are the R&D members.

At the same time that the knowledge transfer material is being produced a beta version of the release become available for internal use and chosen partners.

During a fixed period of time the development team gives support for the beta version. When the beta
version is stable enough become a release candidate.

After some time without major problems reported in the release candidate, it changes to general availability. At this point the new release is available for everyone and the perfective maintenance process ends.

Figure 42 shows the most important actors and the meta process of the testing and deployment phase.

![Figure 42: Main Actors and Process of Phase Three](image)

### 6.2.2.4 Perfective Maintenance Process through time

The several stages of the perfective maintenance process occur in different periods.

The process begins with the Requirements elicitation, High level Architecture and Design. During this phase it is defined what the next release should do, and some high level planning is done.

The Iteration Planning step includes the architecture, specification and drilldown of the features. At this point the tasks are estimated in more detail. Then, during periods of fifteen days the features are coded. At the same time QA and usability tests are produced and the documentation is written. When the fifteen days are ended an Iteration Review Meeting is scheduled and the product increments are shown to the stakeholders of the feature. These phases, from the Iteration Planning to the Iteration Review Meeting, are repeated over time until the features are completed. The main focus of the management tool is in this phase, Iteration Planning.

The final stage of the process is Testing and Deployment. At this point the final tests are performed and the release becomes available for the public. Simultaneously the material of knowledge transfer is produced.

Figure 43 illustrates the evolution of the software development process over time.
Figure 43: Perfective Maintenance Process Through Time
7 Evaluation

To evaluate the performance and quality of the management tool a few tests were performed. It was used a sample of items to test the speed of adding tasks because this action is recurrent during the perfective maintenance process. It was also tested the time spent during the Estimations phase of the process using two methods, Edit Screen and 3-Point. At the end of this chapter some testimonials are shown.

7.1 Add Tasks

It was created 29 items (4 activities and 25 tasks) in the old systems and the new application, and the time spent to add all the items was recorded.

With the previous system it was necessary 15 minutes and 5 seconds to perform the action. During the tasks' creations a few errors occurred because of it poorly interface and absence of automation's mechanisms. For example the project is not automatically associated to the task.

Using the new management tool it was necessary only 9 minutes and 10 seconds. The new tool needed less time (around 6 minutes) to complete the same task.

Figure 44 presents the relation between time spent and number of items. This graph is based in the previous result.

As seen the time difference between the two systems increase along with the items. Insert 160 items consumes around 1 hour and 24 minutes with the old tool, while with the new one it took less than an hour (around 50 minutes). Using the new tool it is only necessary 61% of the total time spent by the old tool to perform the same action.

7.2 Estimations

Another test that was performed was related to the Estimated phase of the perfective maintenance process. During this phase the duration of tasks is estimated in more detail using the 3-Point method
by team members.

For this test it was recorded the time spent using the Add/Edit Tasks Interface and the 3-Point Interface to introduce the 3-point duration of 25 tasks.

Figure 45 shows the results in seconds to perform these actions, using both interfaces.

Using the Edit Interface it was spent 4 minutes and 30 seconds, while with the 3-Point Interface it was only spent 1 minute and 32 seconds. The results are conclusive. Using the second interface the same action is performed approximately three times faster than with the Edit Interface. This is a reduction of 66% of the total time spent using the first interface.

7.3 Testimonials

Here are a few statements provided by the users (developers, managers and team leaders) about the qualities and enhancements presented in the new project management tool.

“In my opinion, the hierarchical vision of tasks is the main improvement. It is very useful to view the tasks grouped by parent and create sub-tasks is much more simple. Editing tasks is easier and has some interesting defaults, for example the project is implicit in the edited task. Drag-and-drop order is another improvement. It is very useful to create a view of the tasks’ priority or an order view for each it must be implemented.”

Hugo Lourenço

“With the new tool it is easier to create and assigned resources and budget for the projects and provides a better way to view the tasks’ hierarchy.”

João Portela

“The main contribution to my work, provided by the new tool, was the ability to group and prioritize tasks in an easy and more visible way. It gives us a better control of the work, because it is possible to see and manipulate the hierarchy of the working topics.”

João Proença

“The new tool helped me to understand the organization and hierarchy of the tasks, and the relation between them. It also helped me to understand the total time of the activities, because in the old tool the sum of the tasks’ effort were not calculated. In addition, the new system gives a better view over the backlog order.”
Leonardo Fernandes

“The project view is very good. Tasks and activities can be organized and grouped according to its priority, instead of having them displayed in a flat list.”

Ricardo Ferreira

“The tree structure is very useful, especially for ordering children tasks. With the previous tool the effort to the same thing was much bigger.”

João Rosado

“With the new tool the creation of backlog items and its prioritization is easier. The release interface is excellent, in particular the view over all projects. Updating the projects’ budget becomes a much simple task.”

Rodrigo Castelo

“The resources planning and allocation tasks are performed more effectively, due to the integration between multiple systems. The tree view also provides a good visibility and control over the working items.”

Luís Lopes

“The reviewed version of Issue Manager, with its new hierarchical view allow me to have an immediate understanding about the tasks’ dependencies presented in the backlog, this was not possible until now. The extra information that the new structure can have (for instance iterations and priority) provides faster way for re-ordination the tasks.”

Miguel Melo
8 Conclusions

To meet the business demands it is obligatory to have an effective perfective maintenance process. To support the process is necessary to have good communication between team members in way that information flows without misunderstands. It is also very important to have a management tool that satisfy the process needs and helps the users to perform more efficient.

Communication is essential to have an effective process. The definition of the engineering terms clarified the concepts, making the communication easy and reduces the misunderstandings. That common knowledge was incorporated in the elicited process and in the management tools making easier to understand the process and the concepts in the new application.

The elicitation of the perfective maintenance process was a hard task. Two methods were used accomplished that goal. The conversational method proved to be very effective in the process elicitation compared to the process mining approach.

The failure of the process mining can be explained due the following reasons:

- Earlier stage in the investigation and lack of knowledge about process mining techniques;
- Misunderstood information, in particular the document where was described how to convert an Access data base to MXML format;
- Original information, Issue Manager logs, might have noise;
- Wrong algorithm, probably the most correct choice for this case was the sequence clustering.

The results obtained using the conversational method were very satisfactory, reflecting how the process is performed by R&D professionals. Some likenesses with the subjects studied in the state-of-the-art chapter were found in the final outcome.

The perfective maintenance process used by OutSystems has a few similarities with release management of ITIL. Phases such as release design, building and configuration, testing and release acceptance and communication, preparation and training are also present in the elicited process. Both processes are concerned in the protection of the live environment, ensuring that only tested and correct versions of software are available [26].

The process reflects all the good practices of the SRUM methodology. Sprint Planning meetings, Sprint Review meetings and development in a series of iterations (Sprints) characterized this agile methodology. These qualities are part of the process, especially in the Architecture, Design and Implementation phase.

The final result is also extremely important for CMM. With the knowledge of the perfective maintenance process it is possible to achieve higher levels in the software process maturity framework. In spite of the fact that the process was not documented it was well defined, practiced, measured and the project commitments were made based on previous successes. For these reasons the perfective maintenance process used by OutSystems is in Repeatable Level.

The new management tool also gave important improvements to the process. Common tasks became
easier and more efficient as result of a simpler interface focused only in the essential elements. The control over the tasks, activities and releases was increased. A lot of automatisms were made in the new tool, such as projects automatically associated with a release and tasks automatically associated with the projects. These mechanisms reduce the errors and make the actions quicker.

Better control mechanisms over the perfective process were created. It is possible to verify the actual progress of each project. Enabling the ability to predict with more precision if the release is on time and budget. This was not possible using the old tool since there was no release concept and no clean way to compare multiple projects.

The time spent with the resources allocations was greatly reduced, due the integration of the management tool with Time Manager and Google Spreadsheets. This improvement makes the operation of updating the projects’ budget much simpler.

The new management tools is also equipped with feedback and reports. All the main operations gave information about its final result and reports about changes in the budget of the projects are sent to the project manager. These mechanisms also increase the control over the project.

The knowledge of the process combined with an appropriate management tool and clear engineering terms understand by all members offers great benefits. The control over the process is increased and total time of the process is reduced. The predictions became more correct and total cost of the process is reduced.

8.1 Future Work

A lot of work had been made, but still plenty of room to improvements. Here are presented a few ideas that can be followed for future improvements:

- Increase report mechanisms;
- Enhance the Iteration manager;
- Add templates (tasks) support;
- Improve the graphical interface;
- Add Checklists and Tags;
- Improve the perfective process;
- Obtain the operations that are performed often.
Bibliography


[41] Z. Zhang, “Effective Requirements Development–A Comparison of Requirements Elicitation Techniques.”
