Data mining analysis of defect data in Software Development Process

by

Joan Rigat

Supervisors

Dr.ir. Jos .J.M. Trienekens
Ir. Jana. Samalikova, TU/e, TM/IS

Eindhoven, February 2009
Overview

In Management the computing is present everywhere. Thanks to computing the range to control the variables of a process is increasing every day. For instance, is possible to extract the data from any process and use the software tools to improve that process.

This study analyzes the data obtained from a Dutch company of Software. We will study those data in order to extract useful information to improve the Software of the company.

This thesis has been realized into the Erasmus exchange program between the Escola Tècnica d'Enginyeria Industrial of Barcelona and the Technische Universiteit of Eindhoven.

This thesis is supported by the Technology Management's department, concretely under the Information System sub-department.

This thesis is directed by Dr.ir. Jos J.M. Trienekens together with Ir. Jana Samalikova.

Dr.ir. Jos J.M. Trienekens is an Associate Professor at TU/e Eindhoven, in the area of software systems development. Ir. Jana Samalikova is nowadays realizing her PhD.
Table of contents

1. **INTRODUCTION** ................................................................. 7
   1.1. General Objective .......................................................... 7
   1.2. Research Outline ........................................................... 8
   1.3. Structure of the report ................................................. 9

2. **LITERATURE STUDY** ..................................................... 10
   2.1. Introduction .............................................................. 10
   2.2. Software Development Process .................................... 10
       2.2.1. V-Shaped Model ...................................................... 12
   2.3. Software Configuration Management .......................... 12
       2.3.1. Change Control Board ............................................. 13
   2.4. What is quality? ......................................................... 14
       2.4.1. The Capability Maturity Model .............................. 16
   2.5. Process Mining ............................................................ 17

3. **SITUATION IN PRACTICE** ............................................... 19
   3.1. Introduction .............................................................. 19
   3.2. Software Development Process .................................... 19
   3.3. Source of the data ...................................................... 23
       3.3.1. Snapshot to event-log ............................................. 24

4. **PROCESS MINING** .......................................................... 26
   4.1. Introduction .............................................................. 26
   4.2. Achievement with PM tools ....................................... 26
       4.2.1. ProM ................................................................. 26
   4.3. Source of analysis ...................................................... 29

5. **ANALYSIS** ................................................................. 32
   5.1. Introduction .............................................................. 32
   5.2. Analysis of the full CCB process ................................. 32
   5.3. Analysis according to the attributes ............................ 35
       5.3.1. Analysis according to attribute Discovered During - Patterns ............................ 35
       5.3.2. Analysis according to attribute Discoverd during -Duration of the task ............. 44
       5.3.3. Analysis according to attribute Severity ......................................................... 45
       5.3.4. Analysis according to attribute Priority .......................................................... 48
       5.3.5. Analysis according to various attributes simultaneously .................................. 49
   5.4. Performance Analysis .................................................. 51
5.4.1. Findings of the Performance Analysis ............................................................. 54

6. CONCLUSIONS .................................................................................................... 56

7. REFERENCES ..................................................................................................... 59
   7.1. Figures ........................................................................................................ 61
   7.2. Tables ........................................................................................................... 62

8. APPENDICES .................................................................................................... 63
1. Introduction

Computer software or just software is a general term used to describe a collection of computer programs, procedures and documentation that perform some tasks on a computer system. The term includes application software such as word processors which perform productive tasks for users, system software such as operating systems, which interface with hardware to provide the necessary services for application software, and middleware which controls and co-ordinates distributed systems. Software includes websites, programs, video games etc. that are coded by programming languages like C, C++, etc.

Software engineering is the application of a systematic, disciplined and quantifiable approach to the development, operation, and maintenance of software. (IEEE 1990). It surrounds techniques and procedures, often regulated by a software development process, with the purpose of improving the reliability and maintainability of software systems. The effort is required by the potential complexity of those systems, which may contain millions of lines of code.

Software size and complexity can grow very quickly, only restricted by the productivity of the developers of software and the capacity of the computers which will execute it. Almost all software products are subject to pressures for change. These pressures are present even before the release of the first version of software. Many software components need to be changed along the process, and these changes need to be handled in a disciplined way to control that the software continues to function as intended, this work is carried out by SCM.

In software engineering, software configuration management (SCM) is the task of tracking and controlling changes in the software. Configuration management practices include revision control and the establishment of baselines. The work of the SCM is the management of Configuration Items (CIs).

The CIs is any component of software, including a documentary item or a Request for Change, which is produced during a Software Development Process (SDP). The evolution of a CI from one phase to the subsequent phase in the SDP is consequence of request improvement for the characteristics of the CIs. The CCB is the entity of the SDP in charge to control the evolution of the software components grouped in different levels of Configuration Items (CIs).

1.1. General Objective

The way that the CIs are handled by the CCB is by the processing of the CI's defects, and because the heterogeneity of their characteristics, it is argued that the CCB process does
not have a unique structure. Moreover preliminary investigations discovered that the CCB process followed in the organization differs from the process specified in the quality manual. This process consists of various tasks. The **Analysis** task is one of them and is skipped most of the cases (70%) of handling defects. Further, (Kusters, et al. 2008) revealed that people tend to make shortcuts in process when the schedule is tight.

The goal of this Project is the analysis of the CCB process in order to understand why the “analysis” task is skipped. Moreover, we will know the differences between the real CCB process and the CCB described in the quality manual.

### 1.2. Research Outline

In order to achieve this goal, we will work with real data from a company which is in the business of electronic products amongst others. Because of confidentiality reasons, here it is not authorized to use real names, and then we will refer to the company with the name COMP-X.

The original data were provided by a consultant that works jointly with Technishe Universiteit Eindhoven. To be able to run the techniques of Process Mining a new database was created from that database.

We will apply a Process Mining tool in order to obtain the information necessary to perform the control-flow behind the process take in practice at the COMP-X.

The figure below shows the conceptual framework of this thesis, with the issues emerged during its development. These issues will be developed in the next chapters.

The framework starts with the global point of view of the COMP-X, and advances up to such point that certain questions need to be solved before continuing. These questions are the aim of this project.
The thesis is divided in six chapters. In this first one, we have given the motivations of the realized work and the general objective of the thesis.

In the second chapter we will explain the theoretical concepts found in the literature related with SDP, CCB, quality, and so on. This chapter will be a base of reference for the analysis of the practical situation.

In the third chapter we will present the practical environment of the project, in other words the SDP and the CCB carried out in the company and moreover, we will describe the database used.

Chapter four is dedicated to present the Process Mining tool, the Pro_M. This tool handles the data with specific format, which differs from the data of the practical situation; these data have more quality and is ready to be mined.

The following chapter is the most important for the project. The analysis of the attributes and the patterns found will be the base to extract the information needed, trying to answer the questions of chapter 1.

At the end, in Chapter 6 the conclusion and further directions are given.
2. Literature Study

2.1. Introduction

The goal of this Project is to increase the quality of the software products by means of the analysis of the Change Control Board process as a part of a Software Development Process. This chapter is dedicated to present the theoretical definitions of these processes. Moreover we will also explain all the concepts related with the questions arisen in the Fig. 1-1.

This chapter is organized as follows: first a general description of the SDP and the lifecycle that uses the COMP-X, we will explain its phases from the model described in Fig. 3-1. Next, will be presented a brief description of the Software Configuration Management that arose from the necessity to have under control all the versions and releases, and will be presented the CCB process. This chapter concludes with a variety of concepts of quality, linking our quality concept with the goal of this thesis.

2.2. Software Development Process

Software Development Process (SDP) or Life Cycle describes the phases of Software development Life Cycle and the order in which those phases are tested.

Software life cycle models describe phases of the software cycle and the order in which those phases are executed. The general, basic model is shown below:

![Fig. 2-1 General Life Cycle Model](image)

Each phase produces deliverables required by the next phase in the life cycle. Requirements are translated into design. Code is produced during implementation that is driven by the design. Testing verifies the deliverable of the implementation phase against requirements.
Requirements; Business requirements are gathered in this phase. This phase is the heart of the project managers and stakeholders. Meetings with managers, stakeholders, and users are held in order to determine the requirements.

Design; The software system design is produced from the results of the requirements phase. This is where the details on how the system will work are produced.

Implementation; Code is produced from the deliverables of the design phase during implementation, and this is the longest phase of the software development life cycle. For a developer, this is the main focus of the life cycle because this is where the code is produced.

Testing; During testing, the implementation is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. Unit tests and system/acceptance tests are done during this phase. Unit tests act on a specific component of the system, while system tests act on the system as a whole.

Waterfall Model

This is the most common and classic of life cycle models, also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed in its entirety before the next phase can begin. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. Unlike what I mentioned in the general model, phases do not overlap in a waterfall model. (Hambling, et al. 2007)

![Waterfall Model](image-url)
2.2.1. V-Shaped Model

The Life Cycle Model used by the COMP-X is based on the V-model. The V-model can be presumed to be the extension of the waterfall model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape. The V-Model demonstrates the relationships between each phase, and puts more emphasis on testing (Meyer 2006). We will explain it in detail at next chapter.

2.3. Software Configuration Management

Software configuration management (SCM) is the set of activities that are designed to control the changes by identifying the work products, establishing relationships among them, defining mechanisms for managing different versions of these work products, controlling changes that are imposed, and auditing and reporting on the changes that are made.

The work of the SCM is the management of Configuration Items (CIs). The CIs is an aggregation of hardware, software, or both, that is designated for configuration items that make up a product (IEEE 1990) that is stored, changed, stored again, and so on. (Hall and Fernández-Ramil 2007).

The evolution of a CI from one phase to the subsequent phase in the SDP is consequence of request improvement for the characteristics of the CIs. These requests for improvement are called “CIs defects” in this project. The CCB is the entity of the SDP in charge to control the evolution of the software components grouped in different levels of Configuration Items (CIs).
2.3.1. Change Control Board

The aim of this paper is to study the utilization of real-life data from a particular software development process to identify the 'real' process as it is carried out in practice (which is often not the process as distinguished in the documents like quality manual).

Kelly (Kelly 1996) is referring to the CCB as one of the best opportunities of interaction between the customer, senior management, the commercial departments and Configuration Management.

The U.S. Small Business Administration (SBA 2008) defines the CCB as the control mechanism that evaluates the scope, applicability, and effect of each requested requirement change (RC). The CCB focuses on items that could affect cost, schedules, or compliance with technical requirements. It acts on any requested RC to the system and provides change approval or disapproval based on defined strategic initiatives, program business objectives, and budgetary parameters. The project CCB has the authority to establish project baselines, initiate or change software, accept testing results, and approve the release of software into production.

To sum up, the Change Control board tracks the CIs defects for the purpose of improve them, to plan the remaining phases of the project and to predict the final results of the development process.

2.3.1.1. Defects

Not only on the field of Software Engineering the defect has several definitions, a defect is considered (The Free Dictionary 2008) the lack of something necessary or desirable for completion or perfection; a deficiency. It must be borne in mind also the following definition: a defect is an imperfection in an object or machine.

These definitions have been transferred to the field of software engineering by various authors (DeMarco 1982) (Putman and Myers 1992) Moreover, in the field of electronics a defect is commonly called “bug”. The origins of the term “bug” come from a real bug, a moth. It was coined during the work of pioneers in technology while they were working in a modern computer.

DeMarco (DeMarco 1982) exposes a difference between bugs and defects, he calls a bug as a something that crawls of its own volition into your code and maliciously messes things up, and even he thinks that to a certain extent the bugs are cute for developers. Bugs could happen to anyone, on the other hand a defect is you own damned fault.
In this project we will stick at defect definition (Putman and Myers 1992). Summarizing he says that a defect is a deviation from the required (or desired) output by more than a specified tolerance. In other words any diversion of wished thing, after observing the results of the obtained software is a defect and somehow it will have to be solved.

Putnam also describes a classification of defects in terms of severity, although we think that the classification that is more suited to us is (IEEE 1990) provided by International Institute of Electrical and Electronics Engineer.

2.3.1.2. Severity of defects

The (IEEE 1990), is a standard glossary of software engineering terminology, which identifies terms currently in use in the field of software engineering. Standard definitions for those terms are established. The document defines defect severity as a scale for determining defect critically. The severity framework for assigning defect criticality that has proven most useful in actual testing practice is a five level scale:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical</td>
<td>The defect results in the failure of the complete software system, of a subsystem, or of a software unit (program or module) within the system.</td>
</tr>
<tr>
<td>Major</td>
<td>The defect results in the failure of the complete software system, of a subsystem, or of a software unit (program or module) within the system. There is no way to make the failed component(s), however, there are acceptable processing alternatives which will yield the desired result.</td>
</tr>
<tr>
<td>Average</td>
<td>The defect does not result in a failure, but causes the system to produce incorrect, incomplete, or inconsistent results, or the defect impairs the systems usability.</td>
</tr>
<tr>
<td>Minor</td>
<td>The defect does not cause a failure, does not impair usability, and the desired processing results are easily obtained by working around the defect.</td>
</tr>
<tr>
<td>Exception</td>
<td>The defect is the result of non-conformance to a standard, is related to the aesthetics of the system, or is a request for an enhancement. Defects at this level may be deferred or even ignored.</td>
</tr>
</tbody>
</table>

2.4. What is quality?

Up to now it has been exposed the common tools of the Development of Software in order to detect and erase the defects. Picking up the thread of the thesis, the goal of the COMP-X is
the improvement the quality of consumers’ life, improving the quality of his products. Quality is a concept which everyone knows, but very few can define. In this chapter we shall explain, in our opinion, what “quality” is.

In the context of software engineering, software quality measures how well software is specified and also, how well the software is conformed with the specifications. Although this is a definition that I believe very successful, in the following are some definitions that are also valid although are departing from different viewpoints.

A definition by McConnell (McConnell 2004) similarly divides software into two pieces: internal and external quality characteristics. External quality characteristics are those parts of a product that face its users, where internal quality characteristics are those that do not.

Another definition by Dr. DeMarco says (DeMarco 1999) "a product's quality is a function of how much it changes the world for the better”. This can be interpreted as meaning that user satisfaction is more important than anything in determining software quality.

Another definition, coined by Weinberg in (Weinberg 1992) "Quality is value to some person." This definition stresses that quality is inherently subjective - different people will experience the quality of the same software very differently.

This slight selection of definitions of quality is a disparate sample of what is this concept. With the aim of standardizing the “quality” ISO published in 1991 a standard that synthesizes a number of characteristics that must be met by the programs that are considered quality programs.

They expose the quality as structured set of features that arose to answer six key questions (Meets the needs for which it was designed?, Does the software have bugs?, Is it easy to use?, etc). This set of features is pointed as follows:

- **Functionality**, which covers the functions that a software product provides to satisfy user needs.
- **Reliability**, which relates to the capability of software to maintain its level of performance.
- **Usability**, which relates to the effort needed to use the software.
- **Efficiency**, which relates to the physical resources used when the software is executed.
- **Maintainability**, which relates to the effort needed to the make changes to the software.
- **Portability**, which relates to the ability of the software to be transferred to a different environment.
In practice, each software organization needs to achieve its own and decomposition of quality, using metrics to measure and track the evolution of each of these attributes. For a given application domain some quality attributes can be more important than others.

The CCB is in charge of handling the request of improvement. These requests come from, problems, changes requirements or bad implementations, etc. The huge number of problems that go through the CCB is such amount, that cover all features of ISO 9126. For this reason we cannot concentrate to a specific field.

2.4.1. **The Capability Maturity Model**

An important contribution to the concept of software quality has been made by (Humphrey 1989). He helped designing the Capability Maturity Model for software, or shortly, the CMM. By now it is in use worldwide to improve the way software is built and maintained. Software process can be very complex. “They define how people interact when they build the most complex and intricate of human products” (Humphrey 1997). Like all models, the CMM is abstract, but is based on experience. It can be seen as a progressive standard that leads an organization to continuously improve on its current practices of software.

The CMM is organized into five maturity levels as shown in Table 2-1.

<table>
<thead>
<tr>
<th>CMM level</th>
<th>Major Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Initial</td>
<td>The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics.</td>
</tr>
<tr>
<td>2 Repeatable</td>
<td>Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on project with similar applications.</td>
</tr>
<tr>
<td>3 Defined</td>
<td>The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. Projects use an approved, tailored version of the organization’s standard software process(es) for developing and maintaining software.</td>
</tr>
<tr>
<td>4 Managed</td>
<td>Detailed measures of the software process and product quality are collected. Both the software process and products are measurable, understood and controlled.</td>
</tr>
<tr>
<td>5 Optimizing</td>
<td>Continuous process improvement is facilitated by quantitative feedback from the process and form piloting innovative ideas and technologies.</td>
</tr>
</tbody>
</table>

The latest assessment states that the COMP-X has determined the CMM at level 3. It means that the organization is capable to define the processes and their intermediate activities, where the inputs and outputs for each of these activities are known and understood. This
level of maturity allows the tracking of defects which can be used in order to plan the remaining phases of the project and also obtain prediction of the outcomes of the processes (Fenton and Pfleeger 1997)

The CMM and this thesis share the same goal, which is the improvement of quality of software products. However the development to improve the Software carried out in this thesis will not follow the CMM framework. Instead of this, we will analyze a specific process tracking the real activities that are taking in practice.

CMM focus on the quality of the software development process as a whole and not on specific problems within particular development processes of an organization.

Information on the overall software process is collected during interviews and the study of document, such as quality manuals. The results of such large-scale improvement investigations become clear at regular intervals of 2-3 years. As a consequence the process improvement activities are not based on real and detailed data from operational development processes, but on the 'official' processes as described in quality manuals of the organization.

The aim of this thesis is the process that is being carried out by COMP-X, exactly; we want to know the reasons to not follow the standard process described on the quality manual. We need actual information derived of the actual process, for this reason this thesis will not focus on the application of improvement through CMM.

Process Mining or data mining is a new research domain. At 2.5 is explained the process mining methodology.

2.5. Process Mining

Although from an academic point of view the term data mining is a stage within a greater process called *extraction of knowledge in databases* (*Knowledge Discovery in Databases or KDD*). What in truth data mining makes is to combine the advantages of several areas like the Statistic, the Artificial intelligence, the Graphical Computation, the Databases and the Massive Processing, mainly using as raw material the databases.

A traditional definition is the following one: "A process non-trivial of valid identification, novel, potentially useful and understandable of comprehensible patterns that are hidden in data" (Fenton and Pfleeger 1997).
From our point of view we defined process mining like the techniques that allow to extract information of event logs. For example, the audit trails of a workflow management system or the transaction logs of an enterprise resource planning system can be used to discover models describing processes, organizations, and products. Moreover, it is possible to use process mining to monitor deviations. (Process Mining Group 2008) The next figure points out the framework of the process mining from our point of view.

![Process Mining Diagram](image)

Fig. 2-4 Process Mining development

The parameters of mining, their mechanisms, the used data as well as the computer program will be explained in detail to chapter 4.
3. Situation in practice

3.1. Introduction

The concepts commented until now will be analyzed in the study case of the COMP-X. The structure of the SDP and concretely the management of the defects will be studied in this chapter. When we put the theoretical concepts into practice, the practical situation becomes much complex as expected.

This chapter is the introduction of the study case, in other words is the “bridge” between the external knowledge of software and the study case of the COMP-X.

This chapter is structured as follows: first is explained the SDP model carried out in the company, this model differs from the one explained at previous chapter, because is more complex and because of this, the CCB is a model to control this complexity. Secondly, the CCB records the path of the CIs in a database in a format of snapshots. These snapshots were not suitable to be analyzed, then (Kusters, et al. 2008) and (Ureña 2008) worked on these data to improve its quality, therefore at the end of this second section we will point out the changes done to improve the quality of those data.

3.2. Software Development Process

As we said at 2.2.1 the Life Cycle Model used by the COMP-X is based on the V-model with few differences. This section is dedicated to present the description of the Software Development Process, defined as an ordered set of activities that, after being completed, result in a software product that is delivered to a customer.

The V-model used in the COMP-X can be seen in Fig. 3-1. The main frame of the model is similar to that commented in chapter 2, though this presents some differences. First, the integration phase and validation phase are subdivided each one in three parts. Also, there is a new phase at the end of the cycle, which is called Rework, and in addition there is a common link between all the phases which is called Architectural support. The description of these phases can be found below the Fig. 3-1.
In the Requirements analysis phase, the requirements of the proposed system are collected by analyzing the needs of the users. This phase is concerned about establishing what the ideal system has to perform. However it does not determine how the software will be designed or built. Usually, the users are interviewed and a document called the user requirements document is generated.

The second phase is the architectural design. This phase is concerned to the selecting the right architecture. This architecture is in charge to realize all, which typically consists of listing of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, details of technology etc. This phase starts simultaneously with the first phase; however, it starts progressively according to what is defined in the first phase.

The third phase is Implementation or coding. This is the process of writing, reviewing and testing the source code. This source code is written in a programming language. The process of writing source code requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms and formal logic. This phase starts in about the middle of progress of the second phase.

The integration consists of three sections: (1) integration test specification (ITS) where the programmers have to develop the specifications of the components, (2) integration test implementation (ITI), to write, review and, debug the test scripts up to being ready for integration testing and, (3) integration test execution (ITE), to perform the integration test
report at system. This phase starts when the architectural design has been completed, and the coding phase is about the middle of progress.

The fifth phase is the execution of verifications and validations about the functionality of the integrated components. It also have three sections, each one will check each integration phase respectively in order to validate them. (VTS, VTI, VTE). This phase starts about halfway Integration.

The rework phase is referring to all corrections made to documents after approval.

In addition there is a common link between all the phases. This link is called Architectural support and manages the necessary information between the phases.

Additionally the SDP is also influenced by two elements that increase the complexity of the processes executed.

1. Releases, (Hoek, et al. 2002) consists of a Configuration Item (most of the time a new version of a program) ready to be exported to the operations environment, due the modifications required after performing tests in the phases of coding, integration, or verification and validation.

2. Increments or Versions, which are the updates made to the whole product. The increments take different CI's releases in order to carry out the update of the whole product.

The management of the changes, releases and versions is carried out in parallel by the COMP-X. In order to manage all these complexities and keep a good organization of elements that flow within the project, a management discipline is necessary to control everything.

The CI's are produced during the SDP and the evolution of CI's from phase to phase in the SDP is consequence of request improvement, then as we said before (2.3.1), the CCB is the entity in charge to control this evolution. The purpose of having a standard, documented change control processes is to ensure that changes are made within the project in a consistent manner.

3.2.1.1. Change Control Board at practice.

The objective of the CCB process is to avoid the introduction of errors. The definition of Change Control Board process in our project expressed as in the Report (Kusters, et al. 2008) is:
The Change Control Board (CCB) process coordinates changes made to the CIs. The CCB tracks and records the path of each CI’s defect from its entry until its exit of the CCB process.

The responsible divisions of requirements engineering and programming carry out the tasks of this process. The role of the CCB is to distribute tasks related to the required change of the CIs, and later evaluate the outcomes of the executed task with respect to the request.

The structure of the CCB process is linear with possible rework in case of failing a task. The process has a sequential manner, the tasks are not executed in parallel, and each task is completed before the next task starts.

The flow of tasks of the CCB process is as follows:

**Task 1.** The CI’s defect is detected and submitted. The tester assigns attributes to the CI’s defect (e.g. priority, severity). Based on the importance, the CI’s defect is either:
- A. further evaluated by the CCB board (Task 2),
- B. or the CI’s defect will directly start with the Analysis task (Task 3).

**Task 2.** The CCB board analyzes the CI’s defect and sends it to the required task depending on the need (Analysis, Resolution, Evaluation, or Concluded task), with the following possibilities:
- C. The CI’s defect is redirected to the Concluded task in case the defect is found duplicated, expected to be fixed in next release or out of the scope of the functionality required;
- D. The CI’s defect is redirected to tasks Analysis, Resolution, or Evaluation depending on the need.

**Task 3.** The task, i.e. either Analysis, Resolution, or Evaluation, starts to handle the CI’s. When the task is completed, one of the four possibilities is chosen:
- E. If the task’s execution is successful then an important CI’s defect is directed to the CCB, and it waits to be redirected again to the next task, (it returns to Task 2)
- F. If the task’s execution is successful then a less important CI’s defect continues with the next logical task, for instance after Analysis it can be Resolution.
- G. If the task was not successfully executed then an important CI’s defect is returned to the CCB for a re-evaluation (Task 2).
- H. If the task was not successfully executed then a less important CI’s defect is handled again by the same task (Task 3)

**Task 4.** Once all the tasks of the CCB process have been successfully carried out, the case of the CI’s defect is closed.

The process can be represented by the following figure:
To track the path of the CIs defect through CCB, the information is stored in a database. The data describes the attributes and time stamps of the CIs defect. As the defect progress through the CCB, new data is recorded in the database.

The CCB can be quite complex, and because of this, the data extracted from the CCB need to be structured, it is necessary that they have fewer noise as possible and besides the fact that those data have to have a certain level of quality. These requirements of the data are necessary in order to transform those data to a format ready to be handled by the process mining tool. If this transformation were not carried out, the outgoing set of data of the CCB would be too chaotic to be analyzed. In the next section we will see how this transformation was done.

### 3.3. Source of the data

The Information about CI’s defect and its path through the CCB process is stored in a database. The CI’s defects were discovered during verification and validation activities and each database record describes the CI’s defect by its attributes and timestamps. Each Submit, Analysis, Evaluation and Resolution activity has a timestamp assigned to its Start and Complete event.

At each Start event of a task, the corresponding data field in the database is filled in. When the execution of the task is successful, also timestamp of the Complete event is recorded. In the case of a failure of a task, the corresponding timestamp of the Start event is removed, that means in our example the data field Analysis (start) is set to be empty.

A consultant creates a copy of the status of the database content as it was at a particular point in the time. We call this copy a snapshot of the CCB database. These snapshots are taken on a weekly basis. The snapshots provide the data to create an event-log, which is the input to the process mining. The snapshots follow the evolution of the handling of the CI’s
defect by the CCB. The evolution is captured in the field CrStatus that stores the information of the current status of a particular CI's defect.

Each snapshot contains a record for each CI's defect that is described by four types of data fields. History Date and Subsystem provide the general reference about the snapshot; they describe the date of the snapshot and the subsystem database from which snapshot was taken. Problem_nr together with the Subsystem data field uniquely identifies a CI's defect. The Priority, Severity, Request_type, CrStatus and Team fields describe the attributes of the CI's defect. The dates of the Start and Complete are stored in the corresponding fields (e.g. the Start event of the Analysis is stored in Analysis (start) field), and the Modify time field stores the date of the last change of the CI's status.

The dates of the Start and Complete are stored in the corresponding fields e.g. the Start event of the Analysis is stored in Analysis (start) field, and the Modify time field stores the date of the last change of the CI's status. An example of such record and its changes over three months is shown in Table 3-1.

<table>
<thead>
<tr>
<th>Data field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>History Date</td>
<td>22/08/2007</td>
</tr>
<tr>
<td>Subsystem</td>
<td>SUB1</td>
</tr>
<tr>
<td>Problem_nr</td>
<td>2714</td>
</tr>
<tr>
<td>Priority</td>
<td>Medium</td>
</tr>
<tr>
<td>Severity</td>
<td>B</td>
</tr>
<tr>
<td>Request type</td>
<td>PR</td>
</tr>
<tr>
<td>CrStatus</td>
<td>In_resolution</td>
</tr>
<tr>
<td>Team</td>
<td>TEAM1</td>
</tr>
<tr>
<td>Submit (start)</td>
<td>04/09/2006</td>
</tr>
<tr>
<td>Submit (complete)</td>
<td>04/09/2006</td>
</tr>
<tr>
<td>Analysis (start)</td>
<td>18/01/2007</td>
</tr>
<tr>
<td>Analysis (complete)</td>
<td>27/03/2007</td>
</tr>
<tr>
<td>Resolution (start)</td>
<td>02/04/2007</td>
</tr>
<tr>
<td>Resolution (compl)</td>
<td>02/04/2007</td>
</tr>
<tr>
<td>Evaluation (start)</td>
<td></td>
</tr>
<tr>
<td>Evaluation (compl)</td>
<td></td>
</tr>
<tr>
<td>Modify time</td>
<td>02/05/2007</td>
</tr>
</tbody>
</table>

Table 3-1 An example of snapshot records of the CI's defect nr. 2714 from a) August 22, 2007 b) September 26, 2007, c) October 22, 2007

3.3.1. Snapshot to event-log

The snapshots capture the evolution of the CI's defects (i.e. the changes of the CI's defect status) in time. This information is used as the input for retrieving the model of the underlying
process that handles the CI's defect. The first step is the translation of these data into a so-called event-log.

This conversion of information to data is out of scope of this project but we will explain the main features.

The input - event-log - with suitable characteristics contains following types of data:

- The “case” (or instances) is an item that is handled by the process. One case may consist of a number of events. Here, they consider a CI's defect as a case.
- The “tasks” that are executed when handling the “case” in the process. They derived the tasks from the CrStatus field. They distinguished the Start and End event.
- The “timestamp” of each “task” executed during the handling of a “case” in the process. They extracted the necessary timestamps from the snapshots fields storing the time information.
- The “resources” involved in the execution of each “task” during the handling of a “case” in the process, they find this information in the Team field in their snapshot.
- The case-related attributes that can be analyzed by different process mining perspectives or they can enrich mined models. In the case of CCB process of the project P, the case-related attributes are Priority, Severity and Request_type fields.

All required types of event-log data can be retrieved from the snapshots. However, they, (Kusters, et al. 2008) and (Ureña 2008) discovered missing records in the snapshots. They excluded some defects of the analysis because these cases at the beginning were submitted to the CCB, but after certain time, they removed from the data set, therefore they not considered those cases as defects later in the Project.

They found another type of problems, the incorrect sequences of events due to the incompleteness of data. They solve these problems with different strategies, mainly they introduce artificial events, Start event or Complete event, depending of each case, to complete the incorrect sequence, therefore they also assign artificial timestamps to newly introduced events, based on the average duration of the tasks or taking the CI's defect attributes into account for example.

After cleaning the data, they translate it to an event-log. The event-log is the input for the process mining using the ProM process-mining tool.
4. Process Mining

4.1. Introduction

The starting point for the analysis step is the resulting event-log as described in the section 3.3.1. To analysing the event-log, we used the plug-ins of the ProM 5.1, the process mining tool. ProM allows for the discovery of different process perspectives (e.g., control-flow, time, resources, and data) and supports related techniques such as control-flow mining, performance analysis, resource analysis, conformance checking, verification, etc. This makes ProM a practical and versatile tool for process discovery and analysis. The plug-ins of ProM implements algorithms to mine and to analyze the event-log.

First we will see why is useful the Process Mining, then the informatics tool that will help to carry out the Process Mining are explained, jointly with the settings for mining the data (Weijters, van der Aalst and Alves de Medeiros 2006). These data are those that we explained at chapter 3.3 however at the end of this chapter we present the data in the current format ready to be mined with PRO-M.

4.2. Achievement with PM tools

The Process mining uses the collected data, in our case these data comes from event-log. The main conclusions of the project studied until now after applying PM tools (PRO-M), are the following:

1. The process mining revealed that people tend to make shortcuts in process. They discovered that the CCB process followed in the organization as captured in the snapshots differs from the specified CCB process. The Analysis task is skipped most of the cases (70%) of handling defects.

2. During the event-log construction phase, they needed to filter out essential number of process instances due to the incompleteness of the snapshots. For that reason, data quality metrics and guidelines for collecting the data need to be investigated regarding the process mining in the software development domain.

4.2.1. ProM

The program PRO-m will be the tool we use for the development of our research project ProM is an extensible framework that supports a wide variety of process mining techniques in the form of plug-ins. It is platform independent as it is implemented in Java.
There are mining plugins, such as:

- Plugins supporting control-flow mining techniques (such as the Alpha algorithm, Genetic mining, Multi-phase mining, ...).
- Plugins analyzing the organizational perspective (such as the Social Network miner, the Staff Assignment miner, ...)
- Plugins dealing with the data perspective (such as the Decision miner, ...)
- Plugins for mining less-structured, flexible processes (such as the Fuzzy Miner)
- Elaborate data visualization plugins (such as the Cloud Chamber Miner) (and many more)

Furthermore, there are analysis plugins dealing with:

- The verification of process models (e.g., Woflan analysis)
- Verification of Linear Temporal Logic (LTL) formulas on a log
- Checking the conformance between a given process model and a log
- Performance analysis (Basic statistical analysis, and Performance Analysis with a given process model)

Finally, ProM supports a large array of log filters, which are a valuable tool for cleaning logs from undesired, or unimportant, artefacts.

### 4.2.1.1. Settings for mining

The Heuristics Miner Plug-in is a tool of ProM, which mines the control-flow perspective of a process model. We want to know what happens in the CCB process, how the information flows, therefore, how is the CCB process control.

To do so, the Heuristics Miner only considers the order of the events within a case. In other words, the order of events among cases is not important. The timestamp of an activity is used to calculate these ordering. Because of Heuristics Miner is based on the frequency of patterns it is possible to focus on the main behaviour in the event log.

The starting point of the Heuristics Miner is the construction of a so called dependency graph. A frequency based metric is used to indicate how certain we are that there is truly a dependency relation between two events $A$ and $B$ (notation $\text{W}_w A \Rightarrow B$).

Let $W$ be an event log over $T$, and $a, b \in T$. Then $|a >_w b|$ is the number of times $a >_w b$ occurs in $W$, and

$$a \Rightarrow b = \left( \frac{|a >_w b| - |b >_w a|}{|a >_w b| + |b >_w a| + 1} \right)$$
First, remark that the value of $a \gtw b$ is always between -1 and 1, and secondly, a high $A \Rightarrow B$ value strongly suggests that there is a dependency relation between activity A and B.

What is a high value, what is a good threshold to take the decision that B truly depends on A (i.e. $A \rightarroww B$ holds)? The threshold appears sensitive for the amount of noise, the degree of concurrency in the underlying process, and the frequency of the involved activities.

We know that each non-initial activity must have at least one other activity that is its cause, and each non-final activity must have at least one dependent activity. Using this information in the so called all-activities-connected heuristic, we can take the best candidate. (with the highest $A \Rightarrow B$ score).

To differentiate the noise between a low frequent pattern is really, three threshold parameters are available in the Heuristics Miner:

i. the Dependency threshold,

ii. the Positive observations threshold,

iii. the Relative to best threshold.

With these thresholds we can indicate that we will also accept dependency relations between activities that have (i) a dependency measure above the value of the Dependency threshold, and (ii) have a frequency higher than the value of the Positive observations threshold, and (iii) have a dependency measure for which the difference with the "best" dependency measure is lower than the value of Relative to best threshold.

In a process, it may be possible to execute the same activity multiple times. If this happens, this typically refers to a loop in the corresponding model.

For length-one loops (i.e. traces like ACB, ACCB, ACCCB, ... are possible) and loops of length two (i.e. traces like ACDB, ACDCDB, ACDCDCDB, ... are possible) the value of $C \Rightarrow Cw$ and $C \Rightarrow Dw$ is always very low.

As the same way before, there are two thresholds, for each length of loop, to accept the dependency relation.

It has been used the default parameter setting (Use all-events-connected-heuristic = true), it means that only dependency relations which are really necessary to connect all task into a workflow net are accepted. Those parameters are presented in table Table 4-1.
<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative-to-best threshold 0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>Positive observations</td>
<td>10</td>
</tr>
<tr>
<td>Dependency</td>
<td>0.9</td>
</tr>
<tr>
<td>Length-one-loops-threshold</td>
<td>0.9</td>
</tr>
<tr>
<td>Length-two-loops-threshold</td>
<td>0.9</td>
</tr>
<tr>
<td>Long distance threshold</td>
<td>0.9</td>
</tr>
<tr>
<td>Dependency divisor</td>
<td>1</td>
</tr>
<tr>
<td>And threshold</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4-1 Values for the Heuristic algorithm

For instance, choosing a dependency threshold of 0.9 means that we need more than 10 positive observations of A directly followed by B to accept a dependency between A and B.

If we are interested in more exceptional behaviour we can change these values.

4.3. Source of analysis

The analyzed event-log has 6870 instances. The Fig. 4-1 shows one snapshot of one of the windows of the mining program, the inspector window. It is possible to see how all the instances are listed in his corresponding column though this list does not have correlative numbers as we have commented in the section 3.3, the event-log of the study only has complete instances.

In the Fig. 4-1, the first column (Instances), shows the cases, the second one shows the sequence of the events for the highlighted case of the first column, and the last one shows the attributes of that event. It is possible to see that every instance (or defect) has a list of events, in this way the path of each defect is tracked. Every task has two events representing the beginning (start) and the end of the task (complete). Each event has a timestamp and simultaneously every event also is defined by his attributes. These attributes are the following ones:

**Discovered_during**: this ordinal variable says the phase of the project where the defect was discovered.

**Execution**: it says if the task has been executed successfully or unsuccessfully or if once all the tasks of the CCB process have been carried out the execution is concluded.

**Product_name**: it is useful for the identification the CI.

**Release**: refers to subsequent interactions of the project to create the product incrementally.
**Request type**: It is the type of the requested change at the moment that the CI is submitted to the CCB, three possible types of requests are distinguished:

- **PR**: implies that all the functionalities are well specified but there is a problem because it does not work. Those ones are the real defects.
- **CR**: implies that in case the functionalities need to be change.
- **IR**: implies that in case the functionalities need to be extended.

**Severity**: is the effect that the defect has, or would have, on the operation of the program after release. It has five types as a difference of (Putman and Myers 1992) On the other hand the following classification is the same that the *Glossary of Software Engineering Terminology (GSET)* (IEEE 1990) as we have seen in the section 2.3.

- **S**: in this case the defect is able to stop the process. This type is called *Critical* in the In the GSET.
- **A**: in this case the defect affects the task process but it cannot stop the process. This type is called *Major* in the In the GSET.
- **B**: this case is the same as A affected but with less importance. This type is called *Average* in the In the GSET.
- **C**: in this case the defect has the minor importance because is a superficial defect that can be solved easily. This type is called *Minor* in the In the GSET.
- **D**: this case implies no classification. This type is called *Exception* in the In the GSET.
Priority: This is the assumed priority for correcting the CI. The evaluation depends on the need of the CI by the entities and the remaining components of the software product. Unlike what we can see in the *Glossary of Software Engineering Terminology*, in our project there is no direct relation between (among) severity and priority since only we have three levels of priority and not five. These three levels are the following ones: High, medium, low.

Some of the attributes listed will be analyzed in depth in the next chapter. Others are not helpful in this study.
5. Analysis

5.1. Introduction

Once we have the event-log we will start to analyze it with ProM.

ProM has the ability to split up the content of the event-log through applying advanced filters. The filters are a type of plugins that the ProM has implemented, which will permit to analyze cases according to his attributes. When the event-log is already filtered then we will use the implemented algorithms to extract the information (patterns, sequences, tendencies...).

With the work in this chapter, we expect to come up with enough information to answer the research questions proposed for the present project. The content of this chapter is the following: first will be carried out an analysis of the CCB process with all the CI's defects found in the snapshots. Thereafter, will be analyzed the CCB process by groups of CI's defects according to their attributes values.

5.2. Analysis of the full CCB process

The process model discovered with the Heuristics Miner plugin is showed in Fig. 5-1. The squared boxes are the tasks. The arcs between tasks represent dependency relations. The label of an arc is the result of the parsing of the traces in the log.

In following subsections this model is analyzed accordingly to its structure and performance characteristics.

We will see at next figure that 69,5% cases (4779 of 6870) directly go from “submit” task to “resolution” without being analyzed. From the remaining 30,5%, (2081 cases) , go from “submit” to “analysis” in first instance only 73 cases did not pass through the “resolution” task later on.

Less than 1% of the cases were handled by the “CCB evaluation” task. The percentage of rework in the tasks does not exceed 1%, and the higher value corresponds to the “evaluation” task with 0,9%.
Fig. 5-1 Heuristic Net of the CCB process as a whole.
Table 5-1 presents another way to express the graph, depicting the percentage of the direct successors of the tasks. For instance, at the second row, which corresponds with “analysis” task, we can see that among the analyzed cases, a percentage of 0.9% of them are sent to the “analysis” again, there is 94.9% of the analyzed cases that are sent to Resolution task, a percentage of 2.4% is sent to “evaluation” task and the remaining percentage 1.8% is sent to “CCB evaluation” task.

The main diagonal of the table displays the percentages of rework; however the only tasks that have it are the Analysis, Resolution and Evaluation.

<table>
<thead>
<tr>
<th></th>
<th>Submit</th>
<th>Analysis</th>
<th>Resolution</th>
<th>Evaluation</th>
<th>CCB</th>
<th>Concluded</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submit</td>
<td>0.0</td>
<td>30.3</td>
<td>69.7</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Analysis</td>
<td>0.0</td>
<td>0.9</td>
<td>94.9</td>
<td>2.4</td>
<td>1.8</td>
<td>0.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Resolution</td>
<td>0.0</td>
<td>0.0</td>
<td>0.2</td>
<td>99.8</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Evaluation</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.3</td>
<td>0.0</td>
<td>99.7</td>
<td>100.0</td>
</tr>
<tr>
<td>CCB</td>
<td>0.0</td>
<td>17.0</td>
<td>41.5</td>
<td>41.5</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
</tr>
<tr>
<td>Concluded</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 5-1 Percentage of direct successors for the full CCB process

In order to distinguish different practices within the process, the *Performance Sequence Diagram Analysis* plug-in was applied. It is used to discover the patterns into a process model. Not all the cases follow the same sequence of activities, so using this plug-in we can depict the main sequences or what we also call patterns.

There were two patterns, from 39 of total, that represent 97.5% of the cases, that is 6699 cases. The next table shows the sequences of these patterns:

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Cases</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 Submit→Resolution→Evaluation→Concluded</td>
<td>4742</td>
<td>69.0%</td>
</tr>
<tr>
<td>P2 Submit→Analysis→Resolution→Evaluation→Conclosed</td>
<td>1957</td>
<td>28.5%</td>
</tr>
</tbody>
</table>

In conclusion we see that the cases inserted in the CCB process are following two main patterns. That is because most of the cases submitted are directly sent to the Resolution task, therefore the model loses his linear structure, to become a process with a structure much complex. In order to analyse thoroughly the causes of that complexity we will split up
the amount of cases in small groups of cases. At the first place we will divide the amount of
cases according to the attributes assigned at the defectes.

5.3. Analysis according to the attributes

The objective of this type of analysis is to know the values of the attributes which are more
prone to skip the analysis.

Using the ProM filters, the cases will be divided into small groups whose common feature will
be the assigned attribute’s value. This division has the purpose of highlighting those cases
that because of their features are interesting to study in front of the others that have to be
relegated to a second plane.

In the next sections the structure is the following, firstly we describe the analysis according
to those attributes; secondly the extracted findings are presented.

For each attribute studied will be a section, excepting the attribute Discovered_During because is divided in two parts: in first place we will analyse the patterns of Heruristics Net of
each phase; second, we will compare the Durations of the CCB’s tasks.

5.3.1. Analysis according to attribute Discovered_During - Patterns

In the field of this attribute there is recorded the phase of SDP where the defect was found. It
is interesting to study it, because the actions taken to solve the defect begin when these
defects are discovered and not before, even the defect was already there but no one noticed
it. Therefore, for our study, it does not have sense to analyzing the moment when the defect
was created.

Anyway, in the source of data (raw material) there is an attribute called Caused_during which
is not placed into the event-log. It has to be noted that when the Caused_during is filled out,
the workers are doing a speculation exercise, because they cannot be sure of which is the
phase where the defect was introduced, to a certain extent, it is not possible to base the
analysis on the these values because the extracted conclusions will not be correct.

These attributes are defined by his values, that we will call variables. These variables are
representing the phases of the project though they are not the principal ones as we will see
in the following table.

The table is divided in two columns, the first one samples the principal phases, already
commented in the section 3.2, the second one samples the separation of the principal
phases into smaller phases. These smaller phases constitute the variables of the attributes
Discovered_during and Caused-during.
Principal Phases | All the phases  
--- | ---  
1. Requirements, definitions and specifications | Requirements  
| Specification  
| SCS-SPS (Software component specification)  
2. Architectural design | Architecture  
| Design  
3. Coding | Implementation  
4. Integration | Alpha Testing  
| Beta Testing  
| Component Testing  
| Integration Testing  
| Scenarios  
5. Verification | Costumer Testing  
| Consumer use  
| System Testing  

Table 5-2 Simplified phases in SDP.

The set out phases are arranged in sequence, from the Requirements of the client to the System Testing. In all SDP there are phases that due to his nature do not have the purpose of discovering the defects (for example Requirements, Implementation, Architecture..) as these are phases of development. In the same way there are phases that never can be responsible for the introduction of defects, for example the phase of Specification, or when the workers are carrying out the Testing phases. With the aim to recognize which variable can be assigned to each attribute, the next table is displayed.

| Principal Phases | All the phases | Caused During | Discovered During  
--- | --- | --- | ---  
1. Requirements, definitions and specifications | Requirements  
| Specification  
| SCS-SPS  
2. Architectural design | Architecture  
| Design  
3. Coding | Implementation  
4. Integration | Alpha Testing  
| Beta Testing  
| Component Testing  
| Integration Testing  
| Scenarios  
5. Verification | Costumer Testing  
| Consumer use  
| System Testing  
| Not applicable  

Table 5-3 Simplified phases in SDP with its variable.
Looking at the table we can identify the phases which has a verification process to detect defects. Thus, those are the phases where a defect can be discovered.

The Not Applicable phase appears as a consequence of a wrong detection of a defect. For example, when the CCB is carried out, and the detection of a defect cannot be assumed to one only phase, then instead of assigning the defect to two phases they assign it to this not real phase. This allows us to be able to realize the data analysis.

This analysis has been realised to recognize if some type of pattern exists within the CCB process, according to the phase where the defects were discovered. In section 5.2 we saw that in the CCB process as a whole, there were 69.5% of cases that had skipped the “analysis”. Now we want to know if this proportion of cases is the same in all the phases along the SDP.

The next table show the results. In the first column we see the set of phases, then we see the number of cases corresponding to each phase and the weight that they represent. The column Cases Skip Analysis, is showing the number of cases that skip the “analysis” together with relative percentage of cases. The minimum percentage value of cases is 49.3% and corresponds to the System Testing. The major value corresponding to Not Applicable is 85.99% of cases.

<table>
<thead>
<tr>
<th>Set of Phases</th>
<th>Cases</th>
<th>Cases Skip Analysis</th>
<th>Cases at CCB</th>
<th>Cases at CCB %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification</td>
<td>543</td>
<td>7.90%</td>
<td>412</td>
<td>75.87%</td>
</tr>
<tr>
<td>Design</td>
<td>477</td>
<td>6.94%</td>
<td>368</td>
<td>77.15%</td>
</tr>
<tr>
<td>Implementation</td>
<td>1282</td>
<td>18.66%</td>
<td>993</td>
<td>77.46%</td>
</tr>
<tr>
<td>Component testing</td>
<td>470</td>
<td>6.84%</td>
<td>372</td>
<td>79.15%</td>
</tr>
<tr>
<td>Integration testing</td>
<td>862</td>
<td>12.55%</td>
<td>532</td>
<td>61.72%</td>
</tr>
<tr>
<td>Costumer testing</td>
<td>81</td>
<td>1.18%</td>
<td>49</td>
<td>60.49%</td>
</tr>
<tr>
<td>Consumer use</td>
<td>33</td>
<td>0.48%</td>
<td>20</td>
<td>60.61%</td>
</tr>
<tr>
<td>System testing</td>
<td>1759</td>
<td>25.60%</td>
<td>867</td>
<td>49.29%</td>
</tr>
<tr>
<td>Not Applicable</td>
<td>1363</td>
<td>19.84%</td>
<td>1172</td>
<td>85.99%</td>
</tr>
</tbody>
</table>

Table 5-4 Simplified phases in SDP

Following the same format, the last column shows the number of cases that pass through the “CCB evaluation” task. The percentage does not exceed at any phase the value of 1.2%

At the next pages we shall analyze the phases with high percentages of representation. These phases are: Implementation, Integration Testing, System Testing and Not Applicable. However we will see in first place, the analysis of the merger of Specification with Design as
a single phase. The Heuristics net found from the other phases are depicted in the Appendix A.

**Atribute: Discovered phase**

**Variable: (Specification) U (Design)**

During the Specification and Design phase, 1020 defects were discovered.

The structure shaped by the CI’s defects shows that after the cases were submitted, 76.1% of them go directly for the “resolution” task without being previously analyzed. A percentage 23.5% of the cases submitted go to “analysis” task. The “analysis” task has a rework of 2.5% of all the cases that are unsuccessfully analyzed.

The remaining cases that have been submitted go to “CCB evaluation” and then it sends these cases to “resolution” task.

The task “resolution” receives 100.0% of the cases coming from “submit”, “analysis” or CCB evaluation.

There are two patterns, from 15 of total, that represent 97.4% of cases. See Table 5-6.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submit → Resolution → Evaluation → Concluded</td>
<td>75.5</td>
</tr>
<tr>
<td>Submit → Analysis → Resolution → Evaluation → Concluded</td>
<td>21.9</td>
</tr>
<tr>
<td>Other sequences</td>
<td>2.6</td>
</tr>
</tbody>
</table>

Table 5-5 Main sequences of Implementation

Fig. 5-2 Pattern of defects found in the specification and design phase.
Atribute: Discovered phase       Variable: Implementation

During the Implementation phase, 1282 defects were discovered. In this phase, the task “submit” to “resolution” happens 77.4% (skipping analysis). The remaining 22.6% go to “analysis” task and 1% of analysed cases are redirected to the “CCB evaluation” task.

The task “resolution” receives 100.0% of the cases coming from “submit”, “analysis” or CCB evaluation.

There are two patterns, from 11 of total, that represent 97.9% of cases. See Table 5-6.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submit→Resolution→Evaluation→Concluded</td>
<td>76.7</td>
</tr>
<tr>
<td>Submit→Analysis→Resolution→Evaluation→Concluded</td>
<td>21.2</td>
</tr>
<tr>
<td>Other sequences</td>
<td>2.1</td>
</tr>
</tbody>
</table>

Table 5-6 Main sequences of Implementation

Fig. 5-3 Pattern of defects found in the implementation phase.
The CIs discovered in this phase, represent 12.5% of total.

The structure shaped by the CI’s defects discovered during Integration Testing is different in comparison with the previous model. The main difference among them is the sequence that involves the task "CCB evaluation". Now, the arcs came from the “submit” task and are redirected to “evaluation”.

There is 61.4% that skip the “analysis” and pass from “submit” directly to “resolution”. The “resolution” task receives the CI’s from the “analysis” and “submit”.

The number of patterns remains at eleven. Also there are two significant patterns that represent the behaviour of 98.4% of CI’s discovered in Integration Testing. See Table 5-7.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submit→Resolution→Evaluation→Concluded</td>
<td>61.4</td>
</tr>
<tr>
<td>Submit→Analysis→Resolution→Evaluation→Concluded</td>
<td>37.0</td>
</tr>
<tr>
<td>Other sequences</td>
<td>1.6</td>
</tr>
</tbody>
</table>

Table 5-7 Main sequences of Integration testing
Attribute: Discovered phase      Variable: System Testing

The fourth part of all defects was discovered during this phase, it represents 1759 cases. Moreover, the structure depicted by the process is much more complex.

The structure shaped by the CI’s defects shows that after the case was submitted, 49,3% of the cases go directly for the “resolution” task without being previously analyzed. In addition, it was found that only the cases that are analyzed can be rejected later in the task “CCB evaluation”, the percentage of these cases is 2,2%.

The “analysis” task has a rework of 1,1% of all the cases that are unsuccessfully analyzed. The remaining structure of the processing is linear it is to say that all the cases that will be “resolved” are successfully handled in the remaining tasks of the process.

In this case we have found 29 patterns. There are 2 main patterns representing less percentage of cases than the variables exposed before. See Table 5-8.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submit→Resolution→Evaluation→Concluded</td>
<td>48,5</td>
</tr>
<tr>
<td>Submit→Analysis→Resolution→Evaluation→Concluded</td>
<td>46,7</td>
</tr>
<tr>
<td>Other sequences</td>
<td>4,8</td>
</tr>
</tbody>
</table>

Table 5-8 Main sequences of System testing

![Pattern of defects found in the System testing phase.](image-url)
Attribute: Discovered phase   Variable: Not Applicable

The structure is the same as Implementation variable, and the weight of percentage is also similar. The System Testing handles 1363 cases against 1282 of Implementation.

During the Not Applicable variable the task “submit” to “resolution” happens 85,6%, 9 points above Implementation. The remaining 14,4% go to “analysis” and 3,1% of analysed cases are redirected to the “CCB evaluation”.

The remaining structure of the processing is linear.

We have found 24 patterns, there are two of them that represent 98,5% of total cases. See Table 5-9.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submit→Resolution→Evaluation→Concluded</td>
<td>85,6%</td>
</tr>
<tr>
<td>Submit→Analysis→Resolution→Evaluation→Concluded</td>
<td>12,9%</td>
</tr>
<tr>
<td>Other sequences</td>
<td>1,5%</td>
</tr>
</tbody>
</table>

Table 5-9 Main sequences of Not Applicable

Fig. 5-6 Pattern of defects found in the Not Applicable phase.
5.3.1.1. Findings of the analysis of the Heuristics Net depending on the phase

After the analysis of the most representative phases, based on the number of CIs defects, we found the main patterns are the same that we identified during the analysis of the CCB as a whole, therefore we have not found any new remarkable pattern based on the variables of this attribute.

Although, we can see a positive tendency to skip the “analysis” while the SDP is progressing, until the project reaches the Component Testing phase. Later appears a negative tendency up to find a value of 50% (Half of the cases submitted go through the “analysis”, the other half skipped it).

We also can see that the percentage of defects discovered before and after the Integration testing phase is 40% and 40%, respectively; the remaining 20% corresponds to the category Not Applicable. Then, without considering this category, the number of defects is equal distributed between the lower and upper part of the SDP.

The amount of cases assigned to Not Applicable “phase” is sufficiently numerous as not be despised. They represent a 20% of the cases. The cases are assigned to this phase because the developers do not know when the defect was discovered so they cannot assign it to a specific phase, this fact would have to be studied in future projects to reduce these cases or even better to eliminate this category. In this manner all cases will be sorted in his phase.

In conclusion the developed software during the upper phases is more abstract than in the lower phases, and not all details of software are known. Could be that the developers do not know the significance of the analysis of the components. However when they are into the lower phases, they can see what type of system has as a whole, or what type of components it uses. Therefore, could be that in the beginning of the SDP, they skips the “analysis” task because they apply the CCB process and his activities in rush. It is possible that the deficiency of information or formation causes to skip the “analysis”.

On one hand we have that there is not really a clear opinion of how to analyze that in the beginning; in the other hand, the literature says (O'Neil 1997)(Marek, Dewayne and Dieter 2000), (Lett 2008) the more defects we found earlier the less work you have at the end.

- Why is the analysis more frequently in the lower lifecycle than in the upper lifecycle?
- The more cases through the analysis, the more defects solved?

To answer both questions could be useful to know the type of defects that are taken the analysis, therefore we will work in this topic, analyzing the Severity attribute. However, first
we will do an analysis according to the duration of the tasks, thus we can now which tasks take more time at which phase.

5.3.2. Analysis according to attribute Discovered during -Duration of the task

The following analysis arose from the necessity to know how long was dedicated to the accomplishment of each task. The results are depicted in a graph.

We have realised the analysis to know the average duration of each task based on the phases of the SDP. The next graph shows this relation. The data used to realise it, are the geometric averages of duration of the task.

In order to clarify the graphic, we have decided to eliminate the “phase” Not Applicable since it did not contribute with relevant information. In addition, we have only depicted tasks whose graphic were significant, therefore, the Submit, the CCB evaluation and Concluded tasks have been separated, because these always have the same length of time along the SDP.

![Fig. 5-7 Task's duration geometric mean](image-url)
5.3.2.1. Findings of the analysis of Duration of the task

The Figure 4-4 shows a progressive increase of the duration of “evaluation” and “analysis” tasks from Component Testing until Customer testing phases; later, the duration descend to values similar to the starting phases of the development.

On the other hand the “resolution” task has an opposite tendency. At the Specification phase, the mean time of “resolution” task is 472 hours and later it drops until 91.3 hours at Implementation phase.

In conclusion, we could say that if the “evaluation” task and “analysis” task they do not last much then extends the duration of the “resolution” task. However later, the task of Resolution gets shorter at the same time as “evaluation” and “analysis” tasks extends their duration. More resolution appears in the beginning, and more analysis and evaluation appears after the Implementation phase, which is quite normal because they are testing, so they are really oriented to finding the defects. Whereas in the beginning after some review, they are fixing the problems at “resolution”, but takes more time because they are not doing the “analysis”.

In the other hand, is quite normal to spend more time to fix the defects at the beginning of the project because the workers have to get used with the new software.

5.3.3. Analysis according to attribute Severity

In the same way as "Caused During" variable, the Severity variable has been mined with Heuristics’ Miner plugin.

This section will answer the next questions:

- Is significantly different the way to handle the Defects according to his severity level?
- As more high severity level, fewer cases skip the analysis?

The attribute “Severity” contains five possible values which range from S (showstopper / blocker) to D (not important at all). This classification exposed at section 4.3 with five levels of severity.

In order to accomplish the objective of this chapter we will do the analysis using Heuristics Miner plugin. In this section, it will be carried out an analysis of three levels of severity, in order to find the differences between them.

We will analyze level S for being a very high level of severity. Of the remaining four levels we have chosen to study the level A and the C, of the same form that we could have chosen B and D. These two non analyzed levels are depicted in the Appendix A.
Attribute: Severity

Fig. 5-8 S (Showstopper/blocker)  
Fig. 5-9 A (Major function affected)  
Fig. 5-10 C (Cosmetic)
Each of the types of severity has a different process model. The main difference among them is the sequence that involves the tasks “CCB evaluation” and the percentage of cases that skip the “analysis”.

There are only 6 cases assigned on S severity, they represent less than 1% of total cases. Within this small group, 5 of 6 CIs skipped the “analysis”, which corresponds 83,3% of cases. The “CCB evaluation” task is not carried out.

In relation to A severity, after the cases are submitted, 50,4 % of them go directly for the “resolution” task without being previously analyzed. In addition, it was found that only the cases that are analyzed can be rejected later in the task “CCB evaluation”, the percentage of these cases is 1,3 %, and then are redirected to “evaluation” task later on. The structure also present evidence of rework, the “analysis” task has a rework of 1%.

In the Cosmetic severity we can see that 73,65% (965 cases) cases go directly from “submit” task to “resolution” task. Moreover, similarly to A severity, the “CCB evaluation” is only presented in few cases after the Analysis, however them are redirected to “resolution” task, not “evaluation” task.

In order to know the progression of the percentage of cases that skip the “analysis” depending on his severity, we summarized the results in the table below:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Values of the attribute</th>
<th>Nº Cases</th>
<th>Cases that skip Analysis and its relative percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Severity S (Showstopper/blocker)</td>
<td>6</td>
<td>0.09%</td>
<td>5</td>
</tr>
<tr>
<td>Severity A (Major function affected)</td>
<td>1545</td>
<td>22.49%</td>
<td>778</td>
</tr>
<tr>
<td>Severity B (Minor function affected)</td>
<td>3253</td>
<td>47.35%</td>
<td>2376</td>
</tr>
<tr>
<td>Severity C (Cosmetic)</td>
<td>1311</td>
<td>19.08%</td>
<td>965</td>
</tr>
<tr>
<td>Severity D (All others)</td>
<td>755</td>
<td>10.99%</td>
<td>654</td>
</tr>
</tbody>
</table>

Table 5-10 Results after applying the Heuristics Miner.

We have depicted the percentage values of cases that do not take the “analysis” according to his severity level:
With the exception of severity type S, as severity level decreases, the number of cases that skips the “analysis” increases, to reach 86.6% of cases in severity level D. Regarding to the S type, we could see that 83.3% represents the second most “chaotic” type.

5.3.3.1. Findings of the analysis of Severity

In relation to the first question that we considered, if the “analysis” is carried out significantly different based on the level of Severity, the answer is yes. Two different tendencies have been detected:

1. The defects with severity S, represent less than 1%, however, the appearance of a defect with this severity during the SDP entails the standstill of all the activities until the defect is resolved, nevertheless a percentage of 83.3% of the cases with S severity do not take into account the “analysis” task.
2. Taking the rest of cases as a set, one sees that as long as the severity level increase, the number of analyzed cases is increasing too. So, there is a direct relation between severity and conformance with the CCB of the quality manual.

In conclusion, in our opinion the second tendency is quite acceptable but the first one not at all. What is quite strange is that the defects with S severity are mainly not analyzed, 5 of 6 cases are skipping the analysis. After consulting our experts with this issue, they said that the defects that causing to block the system, does not have to be really complicated to solve, can also be possible that a critical defect is quite easy to solve.

5.3.4. Analysis according to attribute Priority

The attribute “Priority” contains three possible values which are High, Medium and Low. The Priority tells us how soon it is desired to fix the problem. The levels of priority assigned to the
defects are submitted at the beginning of the CCB process. In this section it will be carried out an analysis in order to find the differences of handling in the three groups of cases.

In order to know the progression of the percentage of cases that skip the “analysis” depending on his priority, we summarized the results in the table below:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition This variable explains the impact of the defect</th>
<th>Nº Cases</th>
<th>Cases that skip Analysis and its relative percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>High</td>
<td></td>
<td>2283</td>
<td>1587 69.5%</td>
</tr>
<tr>
<td>Medium</td>
<td></td>
<td>4127</td>
<td>2844 68.9%</td>
</tr>
<tr>
<td>Low</td>
<td></td>
<td>460</td>
<td>345 75.0%</td>
</tr>
</tbody>
</table>

Table 5-11 Results according to attribute Priority

5.3.4.1. Findings of the analysis of Priority

One sees that the cases with priority High and Medium continue skipping in a 70% of the times the analysis task, however the cases regarding the Low priority skip the analysis until a 75% of the cases. The difference is only of a 5% reason why a direct relation cannot be extracted, in addition the number of cases with Low priority only represents the 6.7% of the total cases.

5.3.5. Analysis according to various attributes simultaneously

Once the cases according to their attributes have been analyzed, we propose one more analysis combining those attributes.

This analysis is carried out with the purpose to discover the characteristics of the cases that do execute the “analysis” task. Three filters on three attributes have been applied at the same time, these attributes are: priority, severity and type of request. The question that will be answered is:

- Can be identified the cases with one combination of attributes more prone to execute the analysis?

These three attributes have their own variables:

- The Priority has three variables: High, Medium, Low
• The Severity has five variables, although we have chosen only four to carry out this analysis. These four are, severity A, B, C and D. The number of cases with severity S is too small to draw conclusions.

• The Request_type sorts the cases in three groups according to his origin:
  o Change request: implies that in this case the functionalities need to be change.
  o Implementation request: that in this case the functionalities need to be extended
  o Problem Request: implies that all the functionalities are well specified but there is a problem because the software does not work. This type of defect is the most prolific during the SDP, in addition with an incorrect managing, or his not detection, it could lead to a serious error in the software product

Once all the cases have been filtered, we have taken the same methodology that in the previous cases. The result is pointed out in the next three tables.

The results are separated in three tables, according to each request type. The tables have two entrances: in columns we have the severity level and in the rows we have the priority. In the crossing of row with column we have the fraction of cases that skip the analysis along with the percentage that corresponds to this fraction.

### Problem Request

<table>
<thead>
<tr>
<th>Severity Priority</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>462/795</td>
<td>326/453</td>
<td>120/166</td>
<td>102/115</td>
</tr>
<tr>
<td></td>
<td>58.1%</td>
<td>71.9%</td>
<td>72.3%</td>
<td>88.7%</td>
</tr>
<tr>
<td>Medium</td>
<td>105/446</td>
<td>707/1180</td>
<td>278/414</td>
<td>107/141</td>
</tr>
<tr>
<td></td>
<td>23.5%</td>
<td>59.9%</td>
<td>67.1%</td>
<td>75.8%</td>
</tr>
<tr>
<td>Low</td>
<td>9/15</td>
<td>17/27</td>
<td>80/114</td>
<td>82/105</td>
</tr>
<tr>
<td></td>
<td>60.0%</td>
<td>62.9%</td>
<td>70.2%</td>
<td>78.1%</td>
</tr>
</tbody>
</table>

### Change Request

<table>
<thead>
<tr>
<th>Severity Priority</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>34/77</td>
<td>35/84</td>
<td>15/28</td>
<td>4/6</td>
</tr>
<tr>
<td></td>
<td>44.2%</td>
<td>41.7%</td>
<td>53.6%</td>
<td>66.7%</td>
</tr>
<tr>
<td>Medium</td>
<td>4/22</td>
<td>92/162</td>
<td>48/77</td>
<td>8/10</td>
</tr>
<tr>
<td></td>
<td>18.2%</td>
<td>56.8%</td>
<td>62.3%</td>
<td>80.0%</td>
</tr>
<tr>
<td>Low</td>
<td>0</td>
<td>1/6</td>
<td>8/15</td>
<td>8/8</td>
</tr>
<tr>
<td></td>
<td>-----</td>
<td>16.7%</td>
<td>53.3%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>
Implementation Request

<table>
<thead>
<tr>
<th>Severity</th>
<th>Priority</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>102/116</td>
<td>87.9%</td>
<td>176/205</td>
<td>85.8%</td>
<td>104/131</td>
</tr>
<tr>
<td>Medium</td>
<td>61/74</td>
<td>82.4%</td>
<td>1009/1117</td>
<td>90.3%</td>
<td>274/315</td>
</tr>
<tr>
<td>Low</td>
<td>0</td>
<td>-----</td>
<td>15/20</td>
<td>75.0%</td>
<td>43/51</td>
</tr>
</tbody>
</table>

In general one sees:

- The Implementation request cases are the more prone to skip the “analysis” task, however it is not possible to differentiate behaviour between PR and CR.
- In relation with the level of priority we cannot find a tendency therefore it doesn’t contribute with new information.

The following values stand out:

- In the Problem Request cases with priority Medium and severity A, a percentage of 23.5% of cases skip the “analysis” task, and the remaining 341 execute the task as it was described in the quality manual.
- Other remarkable value is the Change Request cases with priority Medium and Severity A, a percentage of 18.2% of cases skip the “analysis” task.

In conclusion and retaking the question of this section it would seem to glimpse that the cases with A severity and Medium priority are more prone to pass the “analysis” task, although this behaviour fails in the IR cases.

With our point of view we cannot extract the desired information from this analysis but those three tables under the sight of the analyst of the company could be useful for them.

5.4. Performance Analysis

This analysis has the objective of giving the answer to the next questions:
• Is there any significantly difference of the throughput time between the cases that pass through the “analysis” task with those which not? If the developers skip the “analysis” task, the throughput time process is shorter?

The present Performance analysis studies the throughput time of the CCB process, comparing the cases that pass through the “analysis” task with those which not. However, the measurements given here are references of how much time took the handling of the cases in the process. It is important to understand that many CI’s defects are handled simultaneously in a not linear manner and their real handling takes less time than the one that it is nominally found in the data sets.

In order to achieve specific information we divided the cases according to the Discovered During attribute. With those cases; firstly, using LTL Checker plugin we divided the cases in two groups, which are made by those that pass through the “analysis” task with those which not. Then we mined these groups using the Heuristic Miner plugin to find the Heuristic net. Next, this net has been converted to a Petri net with the purpose to apply the Performance Analysis plugins that have it as a input.

The Performance Analysis plugin has as a output, the Petri net of the analyzed process where the bottle necks can be seen depending on a coloured scale. These data has been transferred to the next tables.

At the first column we see the phases of the SDP, each of these phases is divided in the Analyzed cases and Not Analyzed cases. The column called Arrival rate, presents the number of cases of that type arriving per day. The next three columns are displaying the Throughput times.

Next, three tables will be displayed according to the three combinations of the phases.

1.- The first table is composed by all the SDP phases:

<table>
<thead>
<tr>
<th>Phase of SDP</th>
<th>Type of cases</th>
<th>Total cases</th>
<th>Arrival rate (cases/day)</th>
<th>Throughput time (days)</th>
<th>Min Throughput (days)</th>
<th>Max Throughput (days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification</td>
<td>Analyzed</td>
<td>135.00</td>
<td>0.21</td>
<td>99.11</td>
<td>5.01</td>
<td>288.01</td>
</tr>
<tr>
<td></td>
<td>No Analyzed</td>
<td>408.00</td>
<td>0.65</td>
<td>115.91</td>
<td>1.00</td>
<td>303.00</td>
</tr>
<tr>
<td>Design</td>
<td>Analyzed</td>
<td>109.00</td>
<td>0.17</td>
<td>133.30</td>
<td>8.01</td>
<td>502.01</td>
</tr>
<tr>
<td></td>
<td>No Analyzed</td>
<td>368.00</td>
<td>0.60</td>
<td>140.03</td>
<td>5.00</td>
<td>447.00</td>
</tr>
<tr>
<td>Implementation</td>
<td>Analyzed</td>
<td>294.00</td>
<td>0.45</td>
<td>106.12</td>
<td>1.01</td>
<td>495.01</td>
</tr>
<tr>
<td></td>
<td>No Analyzed</td>
<td>988.00</td>
<td>1.55</td>
<td>117.40</td>
<td>1.00</td>
<td>310.00</td>
</tr>
<tr>
<td>Component Testing</td>
<td>Analyzed</td>
<td>99.00</td>
<td>0.15</td>
<td>83.86</td>
<td>2.01</td>
<td>310.01</td>
</tr>
<tr>
<td></td>
<td>No Analyzed</td>
<td>371.00</td>
<td>0.58</td>
<td>106.63</td>
<td>1.00</td>
<td>301.00</td>
</tr>
</tbody>
</table>
Table 5-12 Performance Analysis of SDP phases

<table>
<thead>
<tr>
<th>Phase of SDP</th>
<th>Type of cases</th>
<th>Total cases</th>
<th>Arrival rate (cases/day)</th>
<th>Throughput time (days)</th>
<th>Min Throughput (days)</th>
<th>Max Throughput (days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification</td>
<td>Analyzed</td>
<td>135.00</td>
<td>0.21</td>
<td>99.11</td>
<td>5.01</td>
<td>288.01</td>
</tr>
<tr>
<td></td>
<td>Not Analyzed</td>
<td>408.00</td>
<td>0.65</td>
<td>115.91</td>
<td>1.00</td>
<td>303.00</td>
</tr>
<tr>
<td>Design</td>
<td>Analyzed</td>
<td>109.00</td>
<td>0.17</td>
<td>133.30</td>
<td>8.01</td>
<td>502.01</td>
</tr>
<tr>
<td></td>
<td>Not Analyzed</td>
<td>368.00</td>
<td>0.60</td>
<td>140.03</td>
<td>5.00</td>
<td>447.00</td>
</tr>
<tr>
<td>Implementation</td>
<td>Analyzed</td>
<td>294.00</td>
<td>0.45</td>
<td>106.12</td>
<td>1.01</td>
<td>495.01</td>
</tr>
<tr>
<td></td>
<td>Not Analyzed</td>
<td>988.00</td>
<td>1.55</td>
<td>117.40</td>
<td>1.00</td>
<td>310.00</td>
</tr>
<tr>
<td>Component testing</td>
<td>Analyzed</td>
<td>99.00</td>
<td>0.15</td>
<td>83.86</td>
<td>2.01</td>
<td>310.01</td>
</tr>
<tr>
<td></td>
<td>Not Analyzed</td>
<td>371.00</td>
<td>0.58</td>
<td>106.63</td>
<td>1.00</td>
<td>301.00</td>
</tr>
<tr>
<td>Integration testing</td>
<td>Analyzed</td>
<td>331.00</td>
<td>0.53</td>
<td>82.06</td>
<td>1.01</td>
<td>439.01</td>
</tr>
<tr>
<td></td>
<td>Not Analyzed</td>
<td>531.00</td>
<td>0.84</td>
<td>101.28</td>
<td>1.00</td>
<td>303.00</td>
</tr>
<tr>
<td>Customer testing+</td>
<td>Analyzed</td>
<td>945.00</td>
<td>1.46</td>
<td>59.49</td>
<td>2.01</td>
<td>306.01</td>
</tr>
<tr>
<td>Consumer use+</td>
<td>Not Analyzed</td>
<td>929.00</td>
<td>1.47</td>
<td>55.95</td>
<td>0.004</td>
<td>299.00</td>
</tr>
<tr>
<td>System testing</td>
<td>Analyzed</td>
<td>331.00</td>
<td>0.53</td>
<td>82.06</td>
<td>1.01</td>
<td>439.01</td>
</tr>
<tr>
<td>Not Applicable</td>
<td>Analyzed</td>
<td>190.00</td>
<td>0.29</td>
<td>79.13</td>
<td>9.01</td>
<td>437.01</td>
</tr>
<tr>
<td></td>
<td>Not Analyzed</td>
<td>1173.00</td>
<td>1.83</td>
<td>109.34</td>
<td>0.004</td>
<td>345.00</td>
</tr>
</tbody>
</table>

Table 5-13 Performance Analysis of SDP phases v2

2.- From the above table we notice that the defects discovered in the Customer Testing and Consumer Use phases, have a similar behaviour. There are phases that represent very few cases and in addition, the throughput time of the cases which pass through the analysis task is much more longer than the cases not analyzed. Because of this, at the next table we join these two phases with the System Testing phase, whose relative weight according to the amount of cases is enough bigger to be notable.
The new throughput time of the compiled phase is similar to the value found in the first table in System Testing phase, therefore this compilation has been carried out correctly.

3.- On the third attempt, we split up the defects between those discovered at the upper phases of SDP and those discovered at the lower part. The upper phases are: Specification, Design, Implementation and Component Testing. The lower phases are: Integration Testing, Customer Testing, Consumer use and System Testing. Notice that Not Application phase cases are not included.

<table>
<thead>
<tr>
<th>Phase of SDP</th>
<th>Type of cases</th>
<th>Total cases</th>
<th>Arrival rate (cases/day)</th>
<th>Throughput time (days)</th>
<th>Min Throughput (days)</th>
<th>Max Throughput (days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper</td>
<td>Analyzed</td>
<td>637.00</td>
<td>0.96</td>
<td>106.12</td>
<td>1.01</td>
<td>502.01</td>
</tr>
<tr>
<td></td>
<td>No Analyzed</td>
<td>2135.00</td>
<td>3.32</td>
<td>119.19</td>
<td>1.00</td>
<td>447.00</td>
</tr>
<tr>
<td>Lower</td>
<td>Analyzed</td>
<td>1277.00</td>
<td>1.99</td>
<td>65.37</td>
<td>1.01</td>
<td>439.01</td>
</tr>
<tr>
<td></td>
<td>No Analyzed</td>
<td>1459.00</td>
<td>2.3</td>
<td>72.47</td>
<td>0.004</td>
<td>303.00</td>
</tr>
</tbody>
</table>

Table 5-14 Performance Analysis of Upper and lower phases of SDP

5.4.1. Findings of the Performance Analysis

Respecting with arrival rate, when the number of cases analyzed is bigger than the cases not analyzed, the arrival rate is also bigger and vicevers.

In general the length of the throughput time is shorter in those cases which the analysis task is taken. At the second table, we can see the exception of the phase composed of those three phases, whose throughput time is shorter in the analyzed cases.

The third table stands out due to the fact that the throughput time of CCB is much longer in the cases discovered in upper phases than the others discovered in the lower phases. The throughput time of the defects discovered at the lower phases is approximately 40 days shorter, without taking into account if cases are analyzed or not.

Associating this last finding with the result of the section 5.3.2 we see that dedicating more time to the analysis and evaluation task, the obtained throughput times are shorter. Therefore, the accomplishment of the process described on the quality manual, gives as a result, better throughput times than the cases passing through the reduced CCB process.

Although the not accomplishment of the “analysis” task supposes a less task in the CCB process, the spent time along the process is longer. Consequently, when the analysis task is skipped, the throughput time of the process is longer than if it is not. To corroborate it we have done the same analysis for the whole cases.
<table>
<thead>
<tr>
<th>Cases</th>
<th>Type of cases</th>
<th>Total cases</th>
<th>Arrival rate (cases/day)</th>
<th>Throughput time (days)</th>
<th>Min Throughput (days)</th>
<th>Max Throughput (days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>All cases</td>
<td>Analyzed</td>
<td>2104.00</td>
<td>3.20</td>
<td>79.55</td>
<td>1.01</td>
<td>502.01</td>
</tr>
<tr>
<td></td>
<td>No Analyzed</td>
<td>4766.00</td>
<td>7.41</td>
<td>102.41</td>
<td>0.004</td>
<td>447.00</td>
</tr>
</tbody>
</table>

Table 5-15 Performance Analysis of all the cases.

In conclusion, the throughput time of analyzed defects takes an average of 23 days less than not analyzed cases. Though we do not find a cause of skipping the analysis task, we have found information which corroborates the importance of doing the Analysis otherwise the throughput time of process is longer.
6. Conclusions

This thesis has been focused on the database form the CCB process, which is in charge of solving the defects found during the development of a software process. The CCB process belongs to a Software Development Process of an anonymous holland company.

The CCB has a linear sequence of tasks executed since the CI’s defect is submitted until his resolution. That sequence of tasks is typified in the company, in the quality manual, with the purpose to be carried out for every new defect. This purpose is not achieved because it was discovered that the “analysis” task, which is a task of that process, was approximately not passed a 70% of cases. This study has presented an analysis of the following task, in order to achieve this purpose, the mining techniques has been used.

The data set used for the present project was brought by a external consultant and lately were transferred to the event log by collaborators of Technology Management department. This transformation resulted an understanding of all the possible routes or sequences that a CI’s defect can follow; and at the same time the event-log has been the raw material for this project. The database includes 6870 complete cases; each case is a CI’s defect which is basically characterized by his sequence of tasks, his timestamps and also by his attributes. This information is compiled in the event log.

In order to study that event log with the process mining techniques, it has been employed the software tool called Pro M, which is in charge to automatically read the event-log, generate the process model of the CCB, and provide performance calculations about the tasks found in the process. It is a not commercial tool used in research’s studies which has been developed by the Eindhoven University of Technology.

To achieve that objective, all the complete cases submitted to the CCB process has been taken into account. The process model discovered using the Heuristics Miner plugin demonstrates that a 69,5% of cases directly go from “submit” task to “resolution” without being analyzed, moreover is displayed an unexpected complex structure that differs a lot from the linear process specified in the quality manual. In order to know the causes that make to take the decision of following or not the “analysis” task, the filters implemented in ProM has been used. Those filters allow us to sort the cases in smaller groups.

The first criterion used for creating those groups depends on the phase where the defect was discovered. Nine groups of cases has been created, the same number as phases of SDP.
that take the verification procedure, which allows us identify the CI’s defect. Two facts arise of this analysis:

- There is a phase called Not Applicable that not belongs to any phase of SDP but appears in the data because the developers assign the cases to that phase when the origin is unknown. This group of cases represent a 20% of total, therefore this phase is the second in relation of amount of cases, and is the first in relation of the percentage of cases that skip the “analysis” with a value that corresponds to 85,99%.

- We have found a positive tendency of skipping the “analysis” task while the SDP is progressing until the project reaches the Component Testing phase. Later appears a negative tendency up to find a value of 50% which corresponds to the System Testing phase. The “resolution” task lasts ten times longer in the Specification phase than in the Implementation phase.

Creating the same groups of CI’s defect, we have investigated the duration of the CCB’s task. The “resolution” task spends ten times more time in the Specification phase than Implementation phase. In the other hand, when we look at the “evaluation” an “analysis” tasks we see the opposite tendency. More resolution appears in the beginning of SDP, and more analysis and evaluation appears after the Implementation phase.

A new study resulted of dividing the amount of cases according to the severity level. Five groups of cases have been obtained according with the five levels of severity. We reach the conclusion that the developers are more prone to skip the “analysis” as the severity level decreases. However, it is remarkable that there is a 0,1% of cases with the higher severity and in spite of that, in a 83,33% of times they not follow the analysis task. This fact was exposed to the collaborative experts, and finally we determined that the cases are too few to extract any conclusion.

Following the same process, we have studied the priority attribute. The findings does not give any significant information; thereby we decided to combine all the attributes in solely one analysis. The major finding has been that the cases with A severity and Medium priority are more prone to pass the “analysis” task, although this behaviour fails in the Implementation Request cases.

The last section, where we have implemented the Performance Analysis, has corroborated the fact that the CI’s defects that follow all the CCB’s tasks, has a minor throughput time that those cases whose analysis task is skipped. We also see that dedicating more time to the “analysis” and “evaluation” task, the obtained throughput time is shorter.

To sum up, we have seen that although we have not found directly causes for the not execution of the “analysis” task, we have realized that the correct execution of the “analysis”
is much more important of what we thought at the beginning, because the accomplishment of the task directly affects the throughput time.

We think that following research lines would be suitable to consolidate the results of this thesis and help to find the causes of the not followin go the analysis task.

- The way of assigning the values of Discovered during attribute should be better because a 20% of cases are assigned to a not real phase, which is a problem to improve the accuracy of the analysis.

- The process of developing software is carried out in components and these have different versions. We don’t see this complexity in the data. Thus if the data were more detailed we could see how the components pass through different versions, then maybe we could know where the tasks are taken and where not. That kind of information could be on the attributes.

- It could be interesting to see how the developers take CCB process in situ in the company. We could register the procedures corresponding of each task and also know the opinion of the developers about the process. With that information we could draw all the process and then find the weak points.

Finally I want to emphasize that these analysis and studies could not be carried out without techniques of process Mining, specifically this thesis has been developed using the ProM, which has allowed us to work in many lines.
7. References


7.1. Figures

Fig. 1-1 Framework of the thesis ................................................................. 9
Fig. 1-2 General life cycle model ............................................................... 10
Fig. 2-2 Waterfall model ................................................................. 11
Fig. 2-3 Waterfall model ................................................................. 12
Fig. 2-4 Process mining development ......................................................... 18
Fig. 3-1 V-model used in the COMP-X ......................................................... 20
Fig. 3-2 Sequence of tasks of the CCB process – Theoretical Process Model ......................................................... 23
Fig. 4-1 Inspector window of our event-log .................................................... 31
Fig. 5-1 Heuristic net of the CCB process as a whole ..................................... 33
Fig. 5-2 Pattern of defects found in the specification and design phase .......... 38
Fig. 5-3 Pattern of defects found in the implementation phase ..................... 39
Fig. 5-5 Pattern of defects found in the system testing phase ...................... 41
Fig. 5-6 Pattern of defects found in the Not Applicable phase ................... 42
Fig. 5-7 Task’s duration geometric mean ....................................................... 44
Fig. 5-8 S (Showstopper/blocker) ................................................................. 46
Fig. 5-9 A (Major function affected) ......................................................... 46
Fig. 5-10 C (Cosmetic) ........................................................................ 46
Fig. 5-11 Relative percentage of cases that skip the analysis event ................ 48
Fig. 8-1 Component testing ....................................................................... 63
Fig. 8-2 Consumer testing ........................................................................ 63
Fig. 8-3 Consumer use ........................................................................... 63
Fig. 8-4 B (Minor function affected) ......................................................... 64
Fig. 8-5 D (All others) ........................................................................ 64
Fig. 8-6 High priority ........................................................................... 65
Fig. 8-7 Medium priority ........................................................................ 65
Fig. 8-8 Low priority ........................................................................... 65
7.2. Tables

TABLE 2-1 MAJOR CHARACTERISTICS ................................................................................................................................ 16
TABLE 3-1 AN EXAMPLE OF SNAPSHOT RECORDS OF THE CI’S DEFECT NR. 2714 FROM A), AUGUST 22, 2007 b) SEPTEMBER 26, 2007, c) OCTOBER 22, 2007 ........................................................................................................................................ 24
TABLE 4-1 VALUES FOR THE HEURISTIC ALGORITHM ........................................................................................................ 29
TABLE 5-1 PERCENTAGE OF DIRECT SUCCESSORS FOR THE FULL CCB PROCESS .......................................................................................... 34
TABLE 5-2 SIMPLIFIED PHASES IN SDP .................................................................................................................................. 36
TABLE 5-3 SIMPLIFIED PHASES IN SDP WITH ITS VARIABLE ........................................................................................................ 36
TABLE 5-4 SIMPLIFIED PHASES IN SDP .................................................................................................................................. 37
TABLE 5-5 MAIN SEQUENCES OF IMPLEMENTATION .................................................................................................................. 38
TABLE 5-6 MAIN SEQUENCES OF IMPLEMENTATION .................................................................................................................. 39
TABLE 5-7 MAIN SEQUENCES OF INTEGRATION TESTING ............................................................................................................... 40
TABLE 5-8 MAIN SEQUENCES OF SYSTEM TESTING .................................................................................................................. 41
TABLE 5-9 MAIN SEQUENCES OF NOT APPLICABLE .................................................................................................................... 42
TABLE 5-10 RESULTS AFTER APPLYING THE HEURISTICS MINER ................................................................................................. 47
TABLE 5-11 RESULTS ACCORDING TO ATTRIBUTE PRIORITY ....................................................................................................... 49
TABLE 5-12 PERFORMANCE ANALYSIS OF SDP PHASES .................................................................................................................. 53
TABLE 5-13 PERFORMANCE ANALYSIS OF SDP PHASES v2 ........................................................................................................... 53
TABLE 5-14 PERFORMANCE ANALYSIS OF UPPER AND LOWER PHASES OF SDP ................................................................................... 54
TABLE 5-15 PERFORMANCE ANALYSIS OF ALL THE CASES ......................................................................................................... 55
8. Appendices

Appendix A

The obtained patterns of the heuristic analysis of the studied attributes that have not been analyzed in detail before but which we have made reference to them, are depicted in the following pages.

Attribute: Discovered during

![Diagram showing patterns of heuristic analysis](image-url)
Attribute: Severity

Fig. 8-4 B (Minor function affected)  Fig. 8-5 D (All others)
Attribute: Priority

Fig. 8-6 High priority
Fig. 8-7 Medium priority
Fig. 8-8 Low priority