UNIVERSITEIT VAN AMSTERDAM

SOFTWARE ENGINEERING

Estimate the post-release Defect Density based on the Test Level Quality

Author:
ing. L. VINKE

Supervisors:
prof. dr. P. KLINT
drs. M. PIL

June 27, 2011

Table of Contents

William A. Foster says *"Quality is never an accident; it is always the result of high intention, sincere effort, intelligent direction and skillful execution; it represents the wise choice of many alternatives."*

Abstract

IT projects carry out all sorts of quality control activities to guarantee a certain quality. Testing is one of them. One goal of testing is to find defects in the developed product. Yet there are many examples of defects found during the operational phase. For example the bug in the Ariane rocket that resulted in the rockets self-destruction. How can the quality of a software product be predicted before it is used? More precisely formulated: how can the post-release Defect Density be estimated based on the Test Level Quality?

First a formal research model is needed that describes the relationships between the different research elements. The Test Level Quality TLQ is a software-metric that indicates the overall quality of the test levels. Defects of a particular defect type should be found in a specific test phase as defined in a test methodology. A test level is a simplified sort of test phase in TMap. When a defect is not found at the test level on which it should be found, the defect is found either too late or too early. This depends on the order of the test levels. The TLQ indicates the extent to which the defects are found too late. The post-release Defect Density is the number of identified defects found during the operational phase per 1000 source lines of code.

We conducted an empirical study about the relationship between the TLQ and the post-release Defect Density. The conclusion for the projects analyzed is that the post-release Defect Density halves when the TLQ increases from Very Low to Low. For all of the projects analyzed the TLQ is classified as either Low or Very Low.

We have analyzed four projects that were executed in the same context. Otherwise it is not possible to draw conclusions; because if they are executed in separate contexts other factors can be the root cause of the changing post-release Defect Density.

Preface

This Master thesis is the final result of the research I have done to conclude my study in the Master in Software Engineering program. I followed this study part-time from September 2008 until June 2011. The quality of software or the quality of processes fascinated me enormously during this period. During the graduation process, I worked within a project team that delivered several releases for a software product. While working on these projects, I noticed that it was a complex process to predict the product quality based on several factors. Due to my interest in software quality, it was an obvious choice to select the research question "how can the post-release Defect Density be estimated based on the Test Level Quality."

Many people have helped me during the graduation process. I received excellent guidance from the University of Amsterdam, and I would especially like to thank Paul Klint, who supported me enthusiastically during the research and gave me good advice. Info Support also give me guidance. I'd like to thank Marco Pil who helped me during the research and gave me positive critical reviews of the delivered artifacts. During the empirical research the following colleagues also helped me either with collecting data or in another way: Wouter Roelofs, Tom Zeinstra, Erik Sackman, Martin Adolfsen, Martin Oldejans, Johan Vink and Xander Buffart. Finally I must mention the support of family. My parents have supported me throughout my entire life and in the education that I have followed. Without them this would not have succeeded. Thanks also to my brothers, sister and sister in law. Last but not least I'd like to thank my grandparents (and grandfather who died last year) who have been following my graduation research and my entire study with great interest. And finally let me thank everyone else who in any way contributed.

Lammert Vinke
Doornspijk
May 2011

Chapter 1

Introduction and motivation

IT specialists and scientists have been trying for years to understand several issues concerning the quality of developed software products. The quality of software products can be expressed with different quality indicators. Quality can be influenced by many different factors during the development and test phase, for example the quality of the code and the quality of the specifications [13]. This research investigates the influence of the quality of the different test phases on the product quality that is expressed as the number of detected defects per 1000 lines of source code.

1.1 Context

This research has been conducted at Info Support. Since its founding in 1986, Info Support has grown from a sole proprietorship to a full-fledged IT services company with over 300 employees.

Info Support carries out the work from offices in Netherlands and Belgium. The core service of Info Support is to develop and manage innovative and robust software solutions. The experience of the employees of Info Support are incorporated into the trainings. But this experience is also incorporated into its software factory Endeavour.

Info Support has made the strategic choice of supporting the following technologies for all the services they offer: Microsoft (.NET, C#, SQL Server) and Oracle (Fusion middleware, Weblogic, Suite, Java).

Endeavour

The research project analyzes a number of IT projects; all of these IT projects work with Info Supports' software factory Endeavour. Endeavour offers many

different elements that enhance the software development process. The purpose of the software development line at Endeavour is threefold, namely:

- Increase in productivity;
- Delivery of adequate quality;
- Increasing transparency of the course of development and testing .

Endeavour focuses on three aspects the People, the Process and the Tools (see figure 1.1). These include for example, process standards, training for users, and tools to support the development process (e.g. defect registration systems) etc.

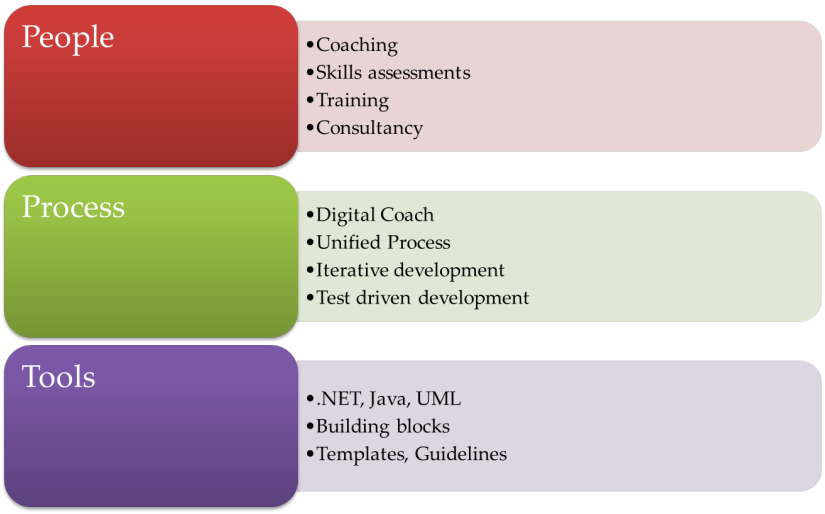

Figure 1.1: People, processes and tools

Endeavour Test

A part of Endeavour that contributes to the achievement of adequate quality is Endeavour Test. Endeavour doesn't aim to achieve the highest product quality possible, but its target is to achieve adequate quality. This is because not every product or module needs the same level of quality. For example a premium calculation module needs a higher quality than a marketing site.

Endeavour uses the methodology TMap for testing the developed software. TMap is based on the so called V-Model (See figure 1.2). The V-Model contains the specification, development and testing stages of a software development project. The V-Model ensures that each step taken during a product's specification and development is tested. This is because in every step a mistake can be made. An example: the product is tested based on the functional design in the System Test and System Integration Test. Every test depicted in the picture is called a test level in TMap.

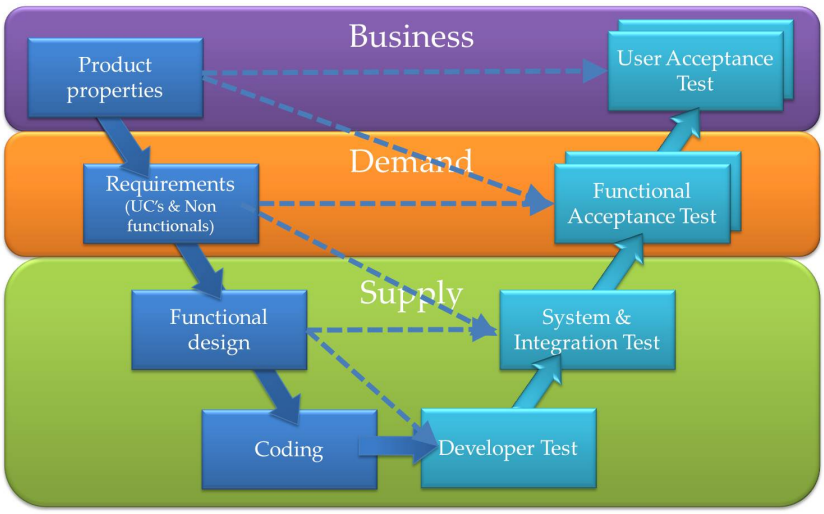

Figure 1.2: V-model

Contribution of research to Endeavour

This research project contributes to the optimization of Endeavour by conducting research into possible relationships between Product Quality and the Test Level Quality. The Product Quality will be indicated with the number of identified defects per 1000 lines of code. When it is possible to estimate the Product Quality before the operational phase; more information is known about the risks on facing defects in the production environment and the quality of the delivered product.

1.2 Problem definition

How many defects will be found in the product after a product is delivered to the production environment? And can the number of the yet unfound defects be determined based on the quality of the various test phases?

Software projects are characterized generally by specification, build and testing processes. At the end of a project, a product will be delivered to the production environment. In the production environment the user can work with the product. During this operational phase the end users will also find previously undetected defects. Is it possible to estimate the number of defects that will be detected during the operational phase based on information from the specification, build and testing processes?

The test discipline makes use of different phases. (See section 1.1 and section 2.3) The quality of a test phase can vary from phase to phase. But what is a good metric to express the quality of a test phase? And can the number of post-release defects be estimated based on the Test Level Quality? So the question

from the previous paragraph can be strengthened: how can the number of post-release defects be estimated based on the Test Level Quality?

During the operational phase defects can also be found in the product. These defects can have negative consequences. Two well-known examples: (a) The defect in the Ariane rocket. The result of that defect was that the rocket destroyed itself and the estimated costs were 370 million U.S. dollars. (b) The year 2000 bug. How many lines of code have been rewritten to correct that problem? Another problem is that sometimes too much is tested. This must be avoided, because it can lead to high costs during the test phase [21].

In addition, the number of post-release defects also depends on several other factors. A good example is the influence of the process quality on the number of post-release defects found. Or what the influence is of the complexity of the product? These elements are also taken into account during this research. This is necessary to properly draw conclusions.

The core of the problem is that it is difficult to estimate how many defects there are to be expected in the operational phase. However, this information is critical to estimate how much risk a company is facing when a product is used in the production environment. Within this research a contribution is made by exploring how the post-release defects can be predicted based on the Test Level Quality.

1.2.1 Research questions

Based on the problem description that is presented in the previous section the following main research question is derived:

How can the post-release Defect Density be estimated based on the Test Level Quality?

To answer this main question several sub-questions are formulated. Each sub-question gives a part of the answer to the main research question. The answers and a more detailed description of these questions can be found in the next chapters.

1. How are the Test Level Quality and the post-release Defect Density defined?
2. What is the relation between the Test Level Quality and the post-release Defect Density?
3. Which other factors have an influence on the results of the research project?

1.2.2 Hypotheses

The hypothesis is that post-release Defect Density is decreasing when the Test Level Quality increases in the manner shown in the figure below (See figure 1.3).

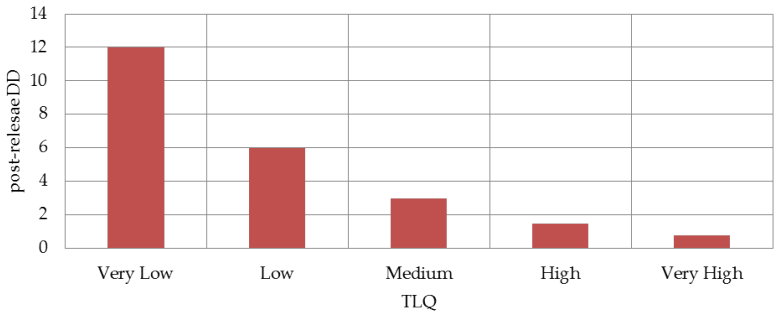

Figure 1.3: Possible relation between post-release Defect Density and Test Level Quality

This hypothesis is made based on literature. McConnel describes several studies concerning the number of found defects per SLOC.

- Software contains an average of 1 to 25 defects per 1000 lines of code. This is based on studies done by Boehm, Geremillion, Yourdon, Jones and Weber;
- Microsoft experiences approximately 10 to 20 defects per 1000 SLOC during the development/testing process and 0.5 post-release defects per 1000 SLOC;
- When using formal methods, the number of defects can be reduced to 3 pre-release defects per 1000 SLOC and 0.1 post-release defects per 1000 SLOC;
- Humphrey indicates that projects that are using TSP (Team Software Process) experience an average of about 0.06 defects per 1000 SLOC.

Based on the studies above it is expected that the projects that will be analyzed in this research contain approximately 15 pre-release defects per 1000 SLOC. These projects strive for a Defect Removal Rate of 95% during the development and testing process. Most projects realize only a Defect Removal Rate of 85%. Based on the 15 defects per 1000 SLOC and the Defect Removal Rate of 85%, the projects that will be analyzed contain 2.25 post-release defects per 1000 SLOC. Additionally it is assumed the Test Level Quality average is 50%. When the Test Level Quality is 100% the project will still contain a few post-release defects. And when the Test Level Quality is 0% the Product Quality decreases enormously.

1.2.3 Scope

Due to time constraints, there was a need to limit the scope of the research. The research was conducted for only one company and one product. The main reason for this choice is that in this way all the other factors stay constant for different projects. When the other factors are constant, it is easier to draw conclusions (See chapter 5). This is because fewer factors play a role and can influence the results. The consequence is that conclusions can only be drawn for this company and product. These findings could be a strong indication however, as to the existence of the same relationship for other products / companies. To finally confirm this, more research is needed. This study provides the models to further investigate those relationships.

1.3 Outline

Chapter 2 briefly describes the literature that was needed to carry out this study. The third chapter describes the research method.

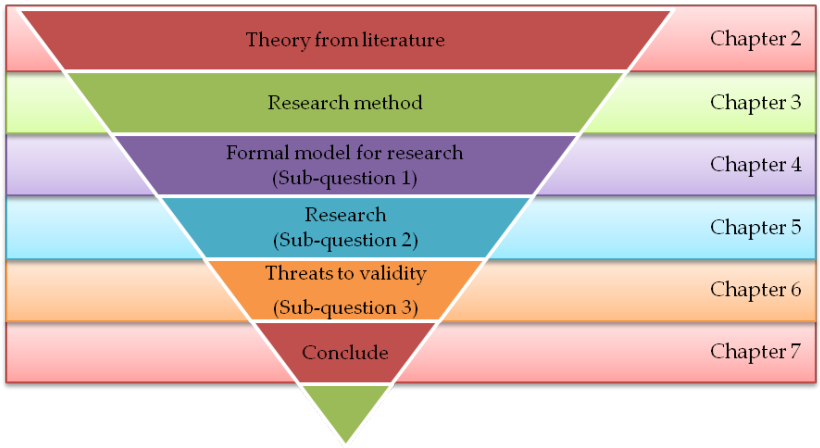

Figure 1.4: Chapter structure

Then in the subsequent chapters the various sub-questions are addressed. In Chapter 4 the first sub-question is answered by a formal model that is used during this research. In Chapter 5 the results from the conducted empirical research are described. The results of this empirical research give the answers to sub-question 2. Possible threats to validity and the influence of other factors (Sub-question 3) are described in Chapter 6. The last chapter describes the conclusions and gives advice for further research.

Thereafter, the appendices include additional information. Appendix A describes step by step how the research was conducted. Appendix B contains the questionnaire that has been used to get more information about the other fac-

tors. Finally, Appendix C contains the data that has been collected during the empirical investigation.

Chapter 2

Background and context

In recent years many scientific research projects have investigated how the quality of a software product can be determined by many factors. For example (a) what is the influence of the code quality on the software product [22] or what is the influence of the process quality on the software product [14]? The knowledge gained during these investigations can be used in this study. The information used is described in this chapter.

In Section 2.1 the definition of quality that is used in this thesis is explained. The second section gives the definition of defects (see Section 2.2). There are several ways to define the test phases. This research uses the definitions that are given by TMap. See Section 2.3 for a more comprehensive description. During the investigation the quality of the test level among other things will be measured, therefore more background information on measurement is given in Section 2.4. The quality of the data used in these measurements can be classified as high or low. Thus, this data can be judged as more or less suited to use. Therefore, Section 2.5 gives more information about data quality and especially data quality optimizations.

2.1 Quality

There are different ideas about quality; therefore several models have emerged that (a) indicate what quality is and (b) provide viewpoints from which quality can be examined [19][26]. For this study we use the following definition of quality [21].

Definition 2.1.1. Quality is the totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs.

Garvin [19] distinguishes a number of different viewpoints of software quality. Below are the different viewpoints described together with the reasoning why this viewpoint was used for this study or not:

- According to the **transcendent based** viewpoint, quality cannot be described precisely. Quality can only be recognized through experience and with a comparative analysis. This viewpoint was not chosen for this research; because it is not possible to measure the quality of an individual product;
- According to the **user based viewpoint**, the quality is high when a product meets the individual needs of users. This viewpoint was not chosen for this research; because it was not possible to interview the users of the product;
- According to the **manufacturing based** viewpoint, quality is expressed as to what extent a product complies with predetermined specifications. This viewpoint is not chosen for this research, because during this research also the defects are addressed that will leads to rewriting the specifications. For example defects that are found by end users during the User Acceptance Test;
- According to the **value based** viewpoint, the quality is not an absolute fact, but the quality must be examined in relation to time, cost and effort. This viewpoint is not chosen for this research, because these data are in many cases not available;
- According to the **product based** viewpoint, the quality is determined by a number of clearly defined characteristics of a product. These characteristics can be determined in an objective way by measurement and audits. This viewpoint is chosen for this research; the characteristic that will be measured is the total number of identified defects (See definition 2.2.1).

This study uses the product-based viewpoint; the other viewpoints were more or less unsuitable for this research.

2.2 Defects

Like quality there are also different definitions of a defect in the scientific world. Therefore the literature is studied to answer this question: what is the definition of a defect?

In the first place there is a distinction between a defect (also called an error) and a failure. A defect is the cause of failure and a failure is the result of a defect. Additionally a defect can lead to zero or more failures. Defects that have not triggered a failure cannot be identified.

Scientifically it is still unknown what the relationship is between the number of identified defects (defects that have led to at least one failure and have been registered) and the number of the residual defects (defects that have not registered or have not triggered any failures) [13]. In other words it is unknown what the total number of defects is. This research focuses only on the identified defects.

There is also a distinction between the identified pre-release defects and identified post-release defects (also called operational defects). Identified pre-release defects are the defects that have been found during development and test process. The operational defects are the defects found after delivery to the production environment, these defects have triggered a failure in the production environment.

Definition 2.2.1. A defect or a fault is a flaw in a component or system that can cause the component or system to fail to perform its required function. A defect, if encountered during execution, may cause a failure of the component or system. Pre-release defects are identified during the development and test process. Post-release defects are identified after the delivery to the production environment.

Defect classification

For each identified defect it can be determined what type of defect it is. In the literature several models describe how defects can be classified [7][8] [9][18][20]. A combination of these models is used during the defect classification process. Defect Causal Analysis (DCA), Orthogonal Defect Classification (ODC) and Personal Software Process (PSP) are three examples.

- With DCA [7] an expert team performs an analysis based on software problem reports that help to identify the root causes of the defects. Based on this information, an action team takes preventative actions to ensure that these kind of problems cannot occur again. For example by carrying out process optimizations.
- ODC [9] supports developers by extracting signatures on the development process from defects. Together with the registration of a defect in the defect registration system a team member determines the type of defect. Based on the distribution of different identified defect types the project team can estimate the maturity level of the developed product. ODC recognizes eight different defect types.
- PSP [20] also has a defect classification schema. Developers use this schema to classify all the identified defects. Based on this classification the developers can improve their code quality. An example: the developer knows better what type of defects happen more often, when writing the code he can avoid these kinds of defects by improved code checking techniques.

2.3 TMap

TMap Next [21] is a structured test approach and is well suited for defining several elements for this research. TMap is widely used by the testing discipline in software developments projects. At the start of the software development

project, the Test Manager creates a Master Test Plan (MTP); this MTP includes the different testing phases. An example of a test phase is the System Test phase where a System Test tester checks if the product is built according to its specifications. Another example is the User Acceptance Phase in which a user checks if the application works according to his or her expectations. TMap calls the testing phases test levels. An important aspect of a test level is that each test level is defined to find a certain type of defects. For example: a System Test focuses on testing if the product meets its specifications, a User Acceptance Test focuses on testing if the product meets to the end-user expectations. The test level definition [21] used by TMap is used intensively during this research

Definition 2.3.1. A Master Test Plan (MTP) is a test plan by which the various test levels are geared to one another.

Definition 2.3.2. A test level is a group of test activities that are managed and executed collectively.

2.4 Software Measurement

During the empirical study different elements will be measured; therefore this section gives some background information about measurement. Nowadays many quality models uses measurements extensively, for example ISO-9000 or CMMI. Therefore much literature is available about this subject [2][1][22][12]. During this study the definition of measurement that is given by Fenton [12] is used.

Definition 2.4.1. Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules.

For software metrics multiple definitions are found. Van Vliet [26] mentions the following combination of properties of a metric:

- An attribute of an entity,
- the function which assigns a value to that attribute,
- the unit which this value is expressed, and
- its scale type.

Goal Question Metric

Basili [1] conceived and described the Goal Question Metric approach to determining valuable metrics. GQM starts with defining a purpose of why something is measured, this purpose it the so named goal. This level is called the conceptual level. To achieve that goal a number of questions are formulated whose answers together achieve the goal. This level is called the operational level.

The set of data which provides answers to the questions are expressed in software metrics. These metrics exist to answer the question in a quantitative way. Hence, this level is called the quantitative level.

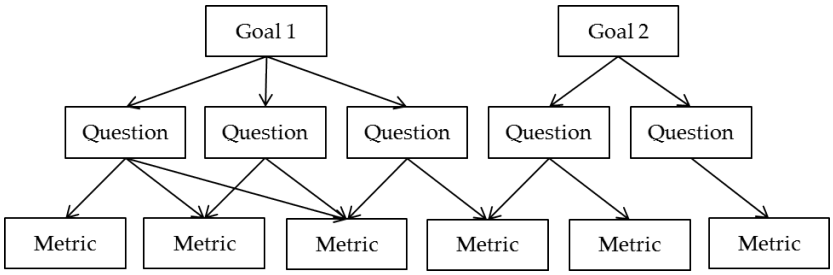

Figure 2.1: Goal Question Metric

2.5 Data Quality Optimizations

One consequence of poor data quality is that no valid conclusions can be drawn. Hence this is also an important aspect of the research. During this research it is possible that the data quality of the registered defects is poor.

Various specialties have already examined several aspects of data quality, including within the world of statistics, management and computer science. Batini and Scannapiezo [4] describe in the book "Data quality, concepts, methodologies and techniques" a comprehensive and systematic introduction to data quality. For this research it is important to get an answer to the following questions: (a) What is data quality? (b) How can data quality be measured? (c) How can Data Quality Improvement Methods improve the data quality?

2.5.1 Data Quality Dimensions

According to Batini the data quality can be expressed in six different dimensions, namely accuracy, completeness, consistency, currency, timeliness and volatility.

- **Accuracy** indicates what the closeness is between a value v and the value v_1. There is a difference between syntactic and semantic accuracy. Syntactic accuracy is the closeness of the value v to the elements of the corresponding definition domain D. An example: a definition domain can be a set with all the defect types (Assignment, Checking, Environment etc.). So, if $v =$ "Assignment", but $v_1 =$ "Checking", then v is considered syntactically correct. In contrast, semantic accuracy is the closeness of the value v to true value of v_1. In that case if $v =$ "Assignment", but $v_1 =$ "Checking", then v is considered semantically incorrect.

- **Completeness** is the extent to which data are of sufficient breadth, depth and scope for the task at hand. This data quality dimension also consists of

different sub-dimensions, namely schema completeness, column completeness and population completeness. Schema completes indicates to which extent the schema used is complete. Column completeness indicates to which extent values are missing for specific properties or columns. Finally population completeness compares all the missing values in set s with a reference set s_1, the difference between the missing values in set s compared to set s_1 indicates the population completeness.

- **Consistency** is the extent to which different sets of data items are consistent. Accuracy is about only two different values. Consistency is on the other hand the comparison of whole sets with data items.
- **Currency, timeliness and volatility** are data quality dimensions that are dependent on time. For example how available certain data is at any given time. These dimensions are not important for this research because research is carried out only after the projects are completed.

2.5.2 Data Quality Improvement

It is important that the data quality is good enough to draw accurate conclusions. If the data is not of sufficient quality Data Quality Improvement methods can be used. An example is Complete Data Quality Methodology, abbreviated CDQM [4]. CDQM consists of three phases, namely (a) state reconstruction, (b) assessment and (c) choice of optimal improvement process.

During the first step the researcher determines how the data was achieved. Questions that can be asked: who enters the data, which quality steps are performed and what processes are followed?

In the second step the existing data is analyzed. During this step the underlying problems of erroneous data are identified. In addition new data quality standards are defined. These new data quality standards must be accomplished after the optimization step.

Finally the data quality optimization is executed. By executing this step different techniques and tools can be used. These steps are documented.

2.6 Related investigations

There have been many studies on the causal or statistical relationship between various factors and the Product Quality in general, see [3][10][11][13] [14][15][16][17][18]. The research methods and information gathered by these studies can be used to answer the sub-question concerning which other factors have an influence on the results of the research project. The conclusions of all previously executed studies are that many factors have an influence on the defect insertion and defect discovery. Many studies categories categorize the factors that have an influence in six areas (See Figure 2.2) So each area depicted in that figure can have ans influence on the conclusions drawn.

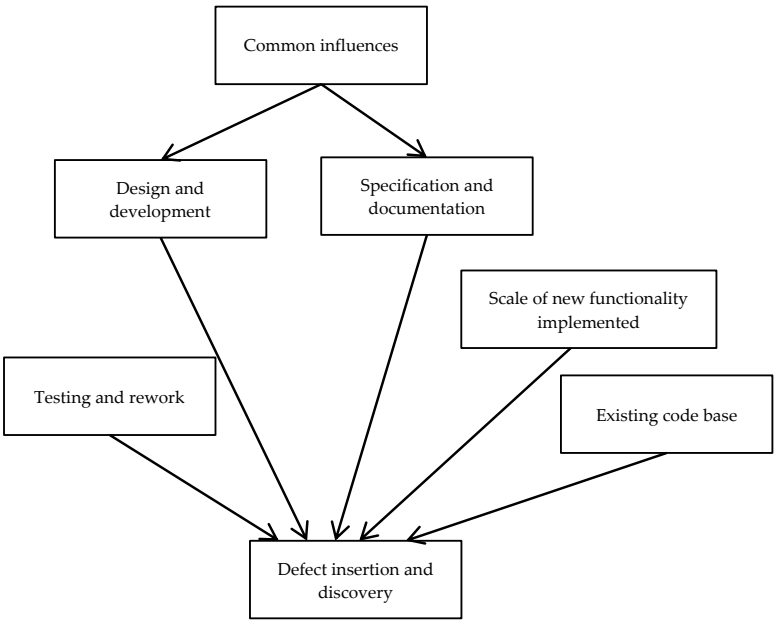

Figure 2.2: Causal relationships to defect insertion and defect discovery

Chapter 3

Research Method and Approach

To answer the main research question "how the post-release Defect Density can be estimated based on the Test Level Quality" an empirical investigation has been executed. To execute the research in a structured scientific manner two methods have been used, namely Goal Question Metric and the Empirical Cycle. Section 3.1 describes the Empirical Cycle and Section 3.2 describes a comprehensive description of the research approach.

3.1 Empirical cycle

De Groot [24] describes the empirical cycle for supporting empirical investigations. This model is not only used in the computer science related field, but also in many other fields. The empirical cycle consists of five stages that follow each other iteratively (see Figure 3.1).

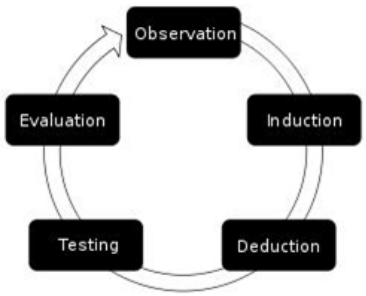

Figure 3.1: The empirical cycle

During the observation phase information is collected and restructured. Based

on this information the hypothesis is formulated during the induction phase. The next step involves the acquisition of empirical data. Based on this empirical data gathered different conclusions can be deduced. This stage is called the deduction phase. After this phase the hypothesis is tested during the testing stage. Finally the last phase evaluates the previous stages of the cycle. This stage is called the evaluation stage. If desired, a new cycle can be started.

3.2 Research method

From a methodological viewpoint the research structure is based on a combination of Goal Question Metric and Empirical Cycle methods. The different steps of these methods are integrated in various places in the survey (see figure 3.2 for an overview). The structure of the chapters maps to the structure of the research.

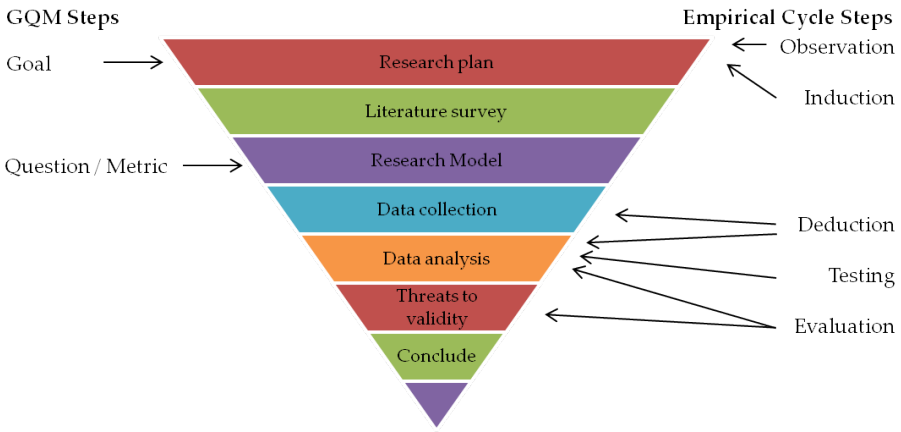

Figure 3.2: Research approach

Research plan

The basis for the research is described in the research plan. This research plan includes (a) the research questions (See Section 1.2.1), (b) a hypothesis (See Section 1.2.2) and (c) a proof-of-concept of the research. From the perspective of GQM the goal is the answer to the main research question, namely the answer to "how can the post-release Defect Density be estimated based on the Test Level Quality? The first two steps (observation and induction) of the empirical cycle were also performed at the beginning of the study. Namely, by writing the research questions, hypothesis and performing a proof-of-concept of the research. This proof-of-concept was a feasibility study for the research.

Literature study

After completing the research plan a literature study has been executed (See Chapter 2). The purpose of this survey was twofold. Namely (a) to obtain more information about previously conducted studies, but also (b) to obtain information used for the construction of the formal model.

Formal model for the research

Based on the information obtained in the previously executed steps, a formal research model has been established (See Chapter 4). By the start of the specification of this model, questions for the GQM method are formulated. The model also incorporates the software metrics. These metrics are also part of the GQM method. Based on these steps the data collection and data analysis are performed.

Data collection and analysis

The data collection and data analysis steps are part of the empirical cycle (See Chapter 5). The collection of empirical data is called the deduction step. The validation to see if the results match the hypothesis (that is described in the research plan) is called the testing step.

Threats to validity

In the first cycle in the evaluation step it has been noted that the data quality of the first project analyzed was not good enough (See Chapter 6). On that basis the data collection model has been adjusted, namely additional information has been gathered to learn more about data quality. Based on this additional gathered information conclusions can be drawn more precisely. The empirical cycle is reiterated for every project that is analyzed.

Conclusions

Finally the conclusions are drawn (see Chapter 7), but the advice for further research is also described.

Chapter 4

Formal model for research

This chapter gives an answer to the first sub question, namely *"How are the Test Level Quality and the post-release Defect Density defined?"* Using the Goal Question Metric approach sub-questions are formulated and related metrics (Test Level Quality and Post-Release Defect Density) are established.

4.1 Application of GQM approach

The purpose of using the Goal Question Metric approach (See Section 2.4) is to answer the main research question. After formulating the problem description the goal is formulated. In short, the goal is to estimate the quality of the product (expressed as the number of defects per 1000 lines of code) based on the quality of the test levels. In this way the risks of going live with the product can be estimated.

The two sub-questions posed to help to answer this question are formulated. First, how can the quality of the test levels be determined? The metric defined that is used to answer this question is the Test Level Quality. And second, how can the quality of the product determined? The post-release Defect Density is used as a metric for expressing the quality of the product.

The following sections give the formal definitions of these software metrics.

4.2 Test Level Quality

The calculation of the Test Level Quality is closely related to the definitions of a defect, a test level and the relation between them. Therefore, first a definition is given of these three elements before the definition of the Test Level Quality is described.

4.2.1 Defects

During a development and test process zero or more pre-release defects will be found. We call this set of pre-release defects $D_{pre-release}$. There are also defects that are found in the production environment. This set of defects is called the post-release defects (or operational defects) $D_{post-release}$. (See also definition 4.2.1).

Definition 4.2.1. Let set $D_i = \langle d_1, ..., d_n \rangle$ where

- *defect* d is a flaw in a component or system that can cause the component or system to fail to perform its required function. A defect, if encountered during execution, may cause a failure of the component or system. Pre-release defects are identified during the development and test process. Post-release defects are identified after the delivery to the production environment.
- i indicates in what stage the defects in the set are identified, $i \in \langle pre - release, post - release, both \rangle$

4.2.2 Test levels

During the development and test phase there are several TMap test levels distinguished. These test levels can be determined based on the Master Test Plan. A test level has some characteristics: (a) each test level has a goal to find a specific type of pre-release defects and (b) each test level is normally carried out by a specific tester (or team of testers) for that test level.

Definition 4.2.2. Let set $TL = \langle l_1, ..., l_n \rangle$ where

- *test level* l is a group of test activities that are managed and executed collectively
- n is the sequence number of the test level (used by data collection)

4.2.3 Relation between defects and test levels

Every pre-release defect will be found in a specific test level, formally defined in the following definition.

Definition 4.2.3. Let predicate $F(d_n, l_n)$: defect d_n is found at test level l_n where $d_n \in D_{Pre-release}$, $l_n \in TL$, where $D_{Pre-release} = \langle d_1, ..., d_n \rangle$, $TL = \langle l_1, ..., l_n \rangle$

For example: an end user has found a null pointer exception (defect d_{21}) during the user acceptance test (l_3).

Each test level has a goal (as described in the Master Test Plan) to find a certain type of pre-release defects. Each pre-release defect can be classified at which test level it should have been found. Below is the formal definition. Section 4.4 further elaborates the process that determines in which test level a pre-release defect should be found.

Definition 4.2.4. Let predicate $M(d_n, l_n)$: defect d_n must be found at test level l_n where $d_n \in D_{Pre-release}$, $l_n \in TL$, where $D_{Pre-release} = \langle d_1, ..., d_n \rangle$, $TL = \langle l_1, ..., l_n \rangle$

Test levels can be organized as subsequent stages, but can also be organized as parallel stages. This research investigates if a pre-release defect is identified too late based on how the test levels are organized, therefore it is required to know how the test levels are organized.

Definition 4.2.5. Let predicate $T(l_n, l_m)$: test level l_n is executed before test level l_m where $l_n \in TL$ and $l_m \in TL$, where $TL = \langle l_1, ..., l_n \rangle$,

If a pre-release defect d_n is found at test level n ($F(d_n, l_n)$ is true) and that pre-release defect should have been found at test level m ($M(d_n, l_m)$ is true), than it is possible to determine if a pre-release defect is found too late. This can be determined as follows: when $T(l_m, l_n)$ is true then the defect is found too late. For every defect in the Defect Registration System the research determines if a defect is found too late or not.

There are two special cases of pre-release defects:

- There are pre-release defects that are found too early. An example: there are pre-release defects related to the integration of components that are found in the System Test test level. According to the goals of the test levels in the Master Test Plan that type of pre-release defects should be found in the Integration Test. This kind of defects is not considered during our research. The research takes into account only the defects that are found too late. This is because it is expected (hypothetically) that the post-release Defect Density does not increases or decreases when there are more pre-release defects found earlier. Hypothetically the efficiency of the test process increases when more defects are found in the correct test level.

- There are also pre-release defects that are identified in a test level parallel to the test level where it should have been found. Each test level l has a sequence number. The sequence number indicates the sequence of the test levels in the set TL. Based on these sequence numbers it can be determined whether a defect is found too late. The sequence numbers assigned are validated by the project members.

4.2.4 Test Level Quality

The Test Level Quality per Test Level TLQ_i is the extent to which the pre-release defects are found at a later test level beyond the test level where it should be found. The TLQ_i is expressed using an ordinal scale. The TLQ_i can have the following values 5 = Very good, 4 = Good, 3 = Moderate, 2 = Bad, 1 = Very bad. In terms of significance it is better to choose an ordinal scale instead of an other scale. Based on the gathered data during the data collection steps, the Test Level Quality TLQ_i per test level l can be calculated.

Test Level	$tooLate(l_i)$	$shouldBeFoundIn(l_i)$	TLQ_i
Developer test	10	20	3 (Moderate)
System test	5	22	4 (Good)
Factory Acceptance test	2	20	5 (Very good)
User Acceptance test	0	3	5 (Very good)

Table 4.1: Example of Test Level Quality calculation

Definition 4.2.6. Let function $TLQ_i = \left(1 - \frac{tooLate(l_i)}{shouldBeFoundIn(l_i)}\right) \cdot 5$ where

- i is the sequence number of the test level
- $tooLate(l_i)$ is a function that calculates all the pre-release defects that should be found at test level l_i, but are found at a later test level
- $shouldBeFoundIn(l_i)$ is a function that calculates all the pre-release defects that should be found at test level l_i
- The TLQ_i can only be calculated when the function $shouldBeFoundIn(l_i)$ has a result greater than zero

An example: A software development project distinguishes four different test levels, namely the Developer Test, the System Test, the Factory Acceptance Test and the User Acceptance Test. For every test level $tooLate(l_i)$ is calculated and $shouldBeFoundIn(l_i)$ is calculated, and then based on that two values the TLQ_i is calculated (See table 4.2.4).

The overall Test Level Quality TLQ can also be calculated. The value of the overall TLQ is expressed with the same ordinal scale.

Definition 4.2.7. Let function $TLQ = \left(1 - \frac{\sum_{i=1}^{n} tooLate(l_i)}{\sum_{i=1}^{n} shouldBeFoundIn(l_i)}\right) \cdot 5$

where

- i is the sequence number of the test level
- n is the maximum sequence number of the test level
- $tooLate(l_i)$ is a function that calculates all the pre-release defects that should be found at test level l_i, but are found at a later test level
- $shouldBeFoundIn(l_i)$ is a function that calculates all the pre-release defects that should be found at test level l_i
- Test levels with $shouldBeFound(l_i) = 0$ should be skipped in this calculation.

4.3 Post-release Defect Density

The post-release Defect Density is a metric that is comparable across different projects. The post-release Defect Density is a quality indicator for the Product Quality that is described in Section 2.1.

Why has Defect Density been chosen? It is possible to measure only the identified post-release defects. But it will be not correct to compare only this number of post-release defects. For example, a product with 5 million source lines of code (SLOC) will probably contain more defects than a product with 100 source lines of code (SLOC). It can also be argued that the complexity may affect the number of identified post-release defects [22]. This research only analyses administrative systems that are developed within the software factory Endeavour. It is therefore assumed that the complexity of these systems is equal. In this study the defect density is used as the indicator for the Product Quality.

Only the total number of added or modified source lines of code are used by the calculation of the post-release Defect Density. When maintenance projects are analyzed; it is not realistic to use the total number of source lines of code for the whole product. When using the KSLOC for the whole product the $DD_{Post-release}$ will be very low. Therefore the research uses the number of added or modified SLOC.

Definition 4.3.1. Let Defect Density $DD_{Post-release} = \frac{|D_{Post-release}|}{KSLOC}$ where

- $|D_{Post-release}|$: is the total number of post-release defects, where $D_{Post-release}$ is the total set of found post-release defects in the first month after delivery into the production environment.
- $KSLOC$: are the total number of added or modified $SLOC$, where $SLOC$ are the total number of 1000 source lines of code.

During this research the number of post-release defects is only measured for the first month after delivery into the production environment.

When the Defect Density is zero, it indicates the best possible Product Quality based on the quality indicator Defect Density. This is because the product contains no defects. If the Defect Density increases, the Product Quality decreases, namely there are more defects per 1000 source lines of code.

4.4 Defect classification

During this research the defect types are classified based on a combination of Orthogonal Defect Classification [9] and Personal Software Process [20] (See Section 2.2). This choice has been established by doing some research. Based on the methods ODC and PSP (DCA doesnt have a list with defect types), twenty random defects are classified independently by two people. The classification by ODC and PSP lead to the same results, namely 70% of the defects were classified

the same by both people . Through a combination of the defect types of both methods the percentage of defects that get the same defect type classification by both people increased to 90%. Therefore during this research a combination of PSP and ODC is used. See Appendix A.1 for a comprehensive description of the defect types used.

In which test level should a defect of a specific defect type be found

The complexity now moves to the establishment of the classification in which test level a defect type should be found, i.e. the function $M(d, l)$ as defined in the preceding paragraph (see 4.2). Based on a MTP a table as specified below can be made. Based on this table it can be determined which types of defects should be found in which phase. For example checking defect types must be found in the Developer Test (test level l_1). When these types of defects are found in the FAT test level (l_2), then these defects are found too late.

		Defect types										
		Build / package	Documentation	Environment	Incorrect requirements	Syntax / Static	Checking	Assignment	Data	Interface	Timing	Incorrect functionality
Test Levels	Developer Test	•				•	•	•				
	System Test		•	•					•			•
	Functional Acceptance Test				•							
	System Integration Test									•	•	
	User Acceptance Test				•							

Figure 4.1: Example table for classifying defects in which test level they must be found ($M(d, l)$)

In the above table there are also a number of defect types that can be found in multiple test levels, for example, the defect type Build / Package. More formally the argument l in the function $M(d, l)$ can get two different values, according to the figure above. In most cases it is clear which test level should be chosen. In the case when it is unclear which test level should be chosen the first test level is chosen [10].

4.5 Concluding remarks

The formal model for the research gives the formal definitions of the Test Level Quality and the post-release Defect Density. This provides the base for carrying out the empirical research. Besides the definitions this formal model also gives information on how the metrics can be calculated.

Chapter 5

Research

Using the formal model for the research described in the previous chapter, the following sub-question can be answered: *What is the relationship between the Test Level Quality and the post-release Defect Density?* To answer this question an empirical study was conducted. And the data from four different projects is analyzed. The results of this empirical research and preliminary findings are described in this chapter.

5.1 Projects analyzed

This research will analyze four different projects. These four projects are all conducted with the software development line Endeavour (See Section 1.1). During the development phase of each project the same team was involved and they are creating a new version of the same product in each release.

The product that is developed is a portal that is used to sell insurances via a portal and a service layer. This service layer communicates with several other components, for example a mainframe or external hosted services. The system supports the primary business. Currently the portal is used by approximately 3000 employees. And the product consists of approximately 3.8 million lines of code (including test code and generated code) and about 50 major or minor components.

The team that worked on the project consists of a project leader, an information analyst, four developers and a system tester. The principal of the various releases is the functional management department. This team consists of about 10 people. This team is also responsible for the execution of acceptance tests. In addition, the user acceptance tests are carried out by the end users of the system. There is also a service desk within the organization where production disruptions (or incidents) are reported by end users.

5.2 Research considerations

For all these projects different research tasks are carried out (See Appendix A for a detailed description). In subsequent sections these steps are described. In these sections the focus is on describing the choices made.

5.2.1 Determine test levels

During this step the test levels are determined based on the Master Test Plan. More formally: the set TL is determined (See Section 4.2.2). Additionally it is determined which types should be found in which test level. This is documented in the decision table defect types versus test levels (See Figure 4.1). Both the set TL and the decision table defect types versus test levels are reviewed by a team member.

5.2.2 Classify defects

When the set TL and the decision table defect types versus test levels are determined every defect can be classified. For each defect five aspects are determined by the investigator, namely:

1. The defect type (See Section 4.4). 10% of the defects is doubly classified by an independent researcher (See Section 6.3);
2. Determine in which test level a defect of a specific defect type should be found. More formally stated: determine the $M(d, l)$, based on the decision table defect type versus test levels;
3. Determine whether a defect is found too late;
4. Determine whether code was assigned for resolving the defect
5. Determine whether the defect registration system is provided with understandable comments for resolving the defect.

During the second iteration of the empirical cycle it is determined that the fourth and fifth aspect mentioned above should also be determined for every defect. In this way the researcher gets more insight into the data quality of the defect registration system. Thereby, the researcher also gets more insight into the risk of misclassification.

5.2.3 Source lines of code

During a test level new defects can be introduced by resolving a defect or by adding, modifying or deleting code. This could impact the Test Level Quality of previous test levels negatively. Example: During the User Acceptance Test a defect is fixed that leads to a new System Test defect. This newly introduced System Test defect is found during the User Acceptance Test. This new defect

is then classified as being too late. This can eventually lead to problems in the results of this research. So there is a need to get more insight into the introduction of new defects.

Adding, changing or removing the code may lead to the introduction of new defects. Therefore, during the development and testing process it is also monitored weekly (a) how much code has been added, changed or deleted, and (b) how many defects are solved during a week. This information can be used to reveal how many new defects are possibly introduced.

The number of added, changed or deleted lines of code can be determined from the data warehouse. By default the number of added lines of a branch operation is for example also part of this summation. The same is true for generated lines of code. This may lead to biased results; this is because it seems that in one week much new code is written, but the real reason is that a developer executes a branch operation. Therefore, some types of change sets are not included in this summation of the number of lines of code, namely:

- Change Sets;
- Generated code;
- Data files;
- Executable.

These exclusions are determined for each project based on a change set analysis. All change sets with more than 500 added, modified or deleted lines of code are analyzed. In this way the filters are calibrated.

5.2.4 Analysis of classified defects

Based on the steps above the software metrics can be calculated as described in the previous chapter. The following metrics are calculated:

- Test Level Quality per test level;
- The overall Test Level Quality;
- The post-release Defect Density;
- Total number of modified source lines of code.

5.2.5 Total number of post-release incidents

Based on the incident registration system the total number of reported post-release incidents (or failures, See Section 2.2) can be calculated per week. These post-release incidents are reported to the Service Desk by the end users of the product. It is possible that a post-release incident is reported several times by different users. Section 5.2.6 describes how to determine the defects that are introduced by the project. For over one year the total number of reported

post-release incidents is calculated per week. (See the blue line in the chart in Figure 5.1) This chart also displays the release times of the four projects with the letters A, B, C and D. The red line indicates the average number of reported post-release incidents.

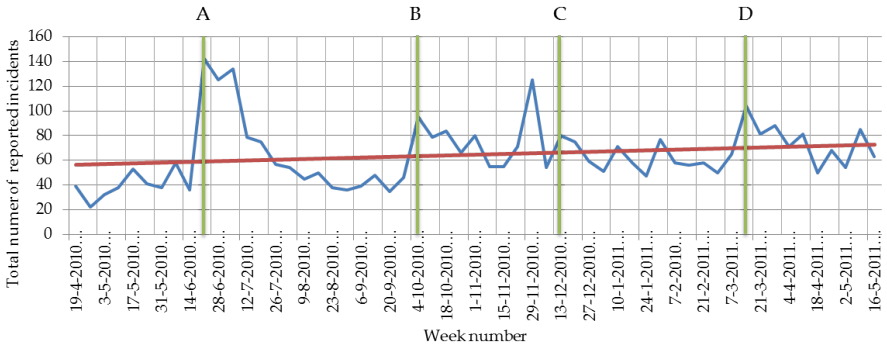

Figure 5.1: Total number of reported post-release incidents per week versus release moments

Based on this chart the following conclusions can be formulated:

- When a release is installed in production, an increase in the number of reported post-release incidents is visible;
- The degree of elevation (height of the peak minus the average that is displayed on the red line) in the total number of reported incidents varies from release to release;
- After the release is taken into production it takes 2 till 4 weeks before the number of reported post-release incidents per week is back to the average level (indicated with the red line);
- The average number of post-release incidents is increasing. The portal and service layer contains more functionality and will therefore contain more errors.

5.2.6 Post-release defects

Based on the incident registration system it is manually determined which defects were introduced by the release. This is done only for the first month after the release is taken into production. The choice to monitor only one month is based on the observed findings in the previous section, namely *after the release is taken into production it takes 2 till 4 weeks before the number of reported incidents per week is back to the average level.*

Finally the post-release Defect Density can be calculated.

Test Level	Goal	Startdate	Enddate
Developer Test	Find problems in the package to deliver. During this level a developer find mistakes that have to do with the robustness / correctness of the code.	1-3-2010	26-3-2010
System Test	Verifies if the product is built according its specifications. These specifications are described in the functional design.	15-3-2010	3-4-2010
Functional Acceptance Test	A functional manager validates if the product is built according its expectations.	19-4-2010	10-5-2010
System Integration Test	Verifies if the product integrates correctly with other systems, for example the Siemens mainframe or the premium calculation engine.	10-5-2010	24-5-2010
User Acceptance Test	An end user validates if the product is built according its expectations.	24-5-2010	19-6-2010

Table 5.1: The test levels used for project A

5.3 Research results of project A

In this section the results of the analyzed project A are described. See appendix C for a detailed description of all four projects.

5.3.1 Test levels

Within this project there were five different test levels distinguished. These test levels were determined based on the Master Test Plan. The different test levels are displayed in table 5.1.

After the test levels are determined, a decision table is constructed which shows which defects of one particular type of defect should be found in a given test level (See Figure 5.2)

5.3.2 Analysis of defects

First, all the defects are analyzed as described in section 5.2.2. Appendix C contains all of the research data used. Also the code churn is weekly collected. The chart in Figure 5.3 gives an overview of these results, and it provides more information about the following aspects:

		Defect types										
		Build / package	Documentation	Environment	Incorrect requirements	Syntax / Static	Checking	Assignment	Data	Interface	Timing	Incorrect functionality
Test Levels	Developer Test	•				•	•	•				
	System Test		•	•					•			•
	Functional Acceptance Test				•							
	System Integration Test									•	•	
	User Acceptance Test				•							

Figure 5.2: Decision table defect types versus test levels

- The defects that are not resolved yet per week. The defects have then been broken down by defect type. See legend for the different defect types;
- The number of source lines of code that are added, changed or removed per week;
- The test levels;
- The date on which the product is delivered to production (indicated with the green line).

Based on this chart the following conclusions can be formulated:

- At the beginning of the project more code is changed. This is in conformance with the expectations; because then the functionality of the product is built;
- After delivery to production there are still errors in the product;
- During the test phase there were minimal code changes;
- There are many defects with the defect type incorrect functionality and environment. There still exist defects of this type at the end of test phase.

This will probably lead to a poor Test Level Quality; this is because these defects are found too late. This is based on the decision table test level versus defect types.

After that, the chart can be created, which is displayed in Figure 5.4. The chart is similar to the chart in Figure 5.3, only this chart indicates at what stage a defect of a particular defect type should be found. Based on this chart it is possible to see how many defects that should have been found in the System Test phase are still open after the System Test phase is closed. This also applies to other phases.

5.3.3 Post-release defects

A total of eight identified post-release defects is the result of this release. These are determined by analyzing all the incidents that have been registered in the defect registration system in the first month of production. Based on the set of post-release incidents the set with post-release defects is identified. This is validated by a team member. An observation is that the number of reported incidents rose to around 140 incidents per week in the first week of production (See figure 5.5).

5.3.4 TLQ versus post-release Defect Density

For these results see the next section.

5.4 Comparison between projects

The study has analyzed four different projects. The results of these projects are compared to find a relationship between the TLQ and the post-release Defect Density. The results of all the projects are listed in Table 5.2 below.

Based on this table the following conclusions can be formulated:

- The post-release Defect Density halves when the Test Level Quality increases from Very Low to Low;
- The Test Level Quality of projects B, C and D are almost equal, in all three projects the Test Level Quality is classified as Low. The post-release defect density is also almost equal. The Test Level Quality of project A is much lower than the other projects. This also leads to higher post-release Defect Density;
- The cause of the lower Test Level Quality of project A is known. After project A there was a new team member, namely a new System Tester.
- As the post-release Defect Density increases, the number of reported incidents increases also when this is compared to the average number of reported incidents;

Property	**Project A**	**Project B**	**Project C**	**Project D**
TLQ	Very Low (85% too late)	Low (59% found too late)	Low (69% found too late)	Low (55% found too late)
Modified source lines of code	3100	3600	4500	4000
Total number of pre-release defects	41	83	85	93
Number of pre-release defects found too late	35	49	59	52
Pre-release Defect Density	13 defects / 1000 SLOC	22,6 defects / 1000 SLOC	19 defects / 1000 SLOC	23 defects / 1000 SLOC
The average number of reported incidents is increased by:	80	30	20	35
Total number of post-release defects	8	6	5	7
Post-release Defect Density	3,0 defects / 1000 SLOC	1,6 defects / 1000 SLOC	1,1 defects / 1000 SLOC	1,7 defects / 1000 SLOC

Table 5.2: The results for project A, B, C and D

- As the pre-release Defect Density decreases, the post-release Defect Density increases;
- The Test Level Quality is pretty low for the four projects. The main reason for this is that many Developer Test and System Test defects are found too late.

5.5 Concluding remarks

In the context of the projects analyzed the post-release Defect Density halves when the Test Level Quality increases from Very Low to Low. At present there were no projects with a Test Level Quality classified as Medium, High or Very High. Therefore, on that end of the Test Level Quality spectrum, no conclusion can be drawn. Based on the results so far described in section 5.4 there is an indication that the post-release Defect Density can be estimated based on the Test Level Quality. There is need for further empirical research, on this and also on other projects, to further explore this relationship (See Section 7.2).

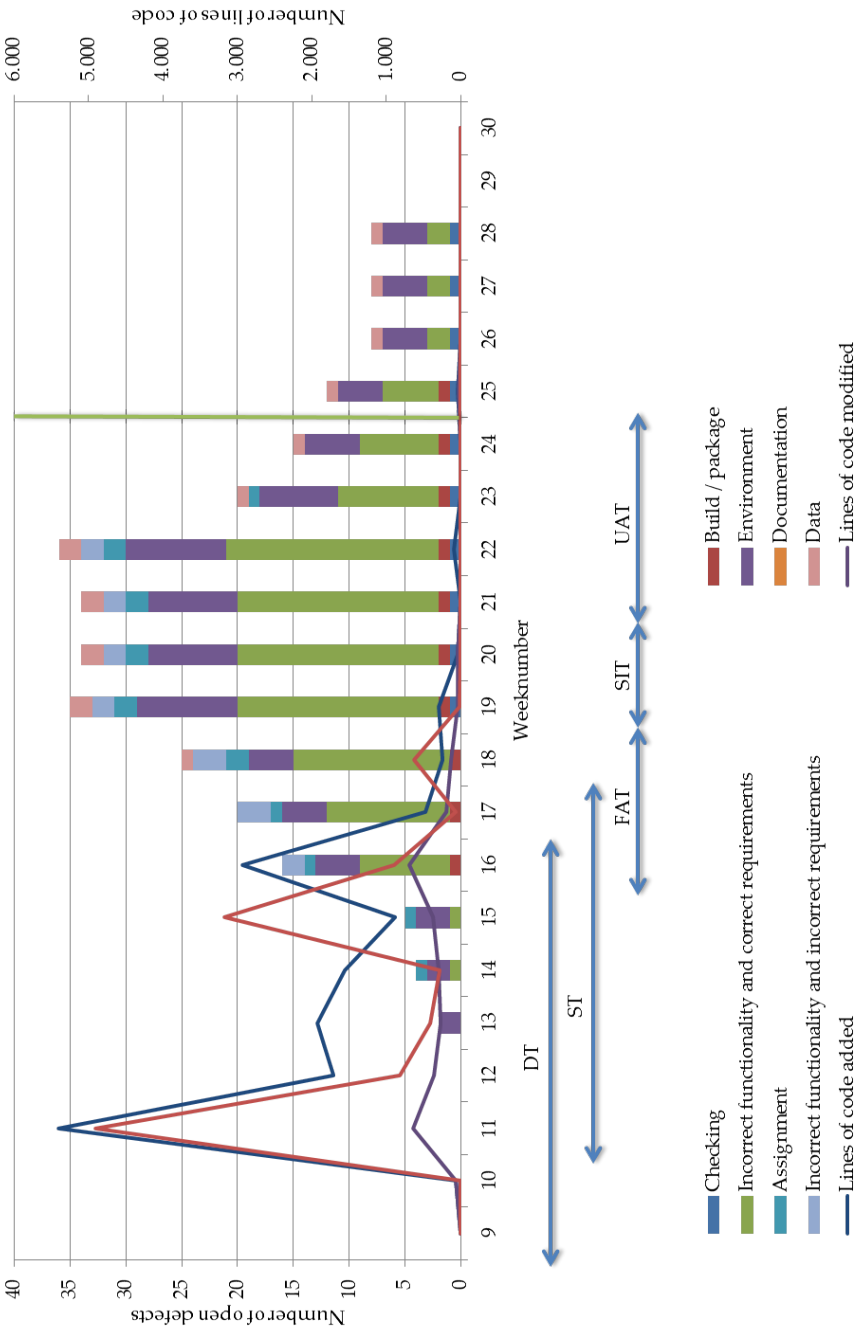

Figure 5.3: Possible relation between post-release Defect Density and Test Level Quality

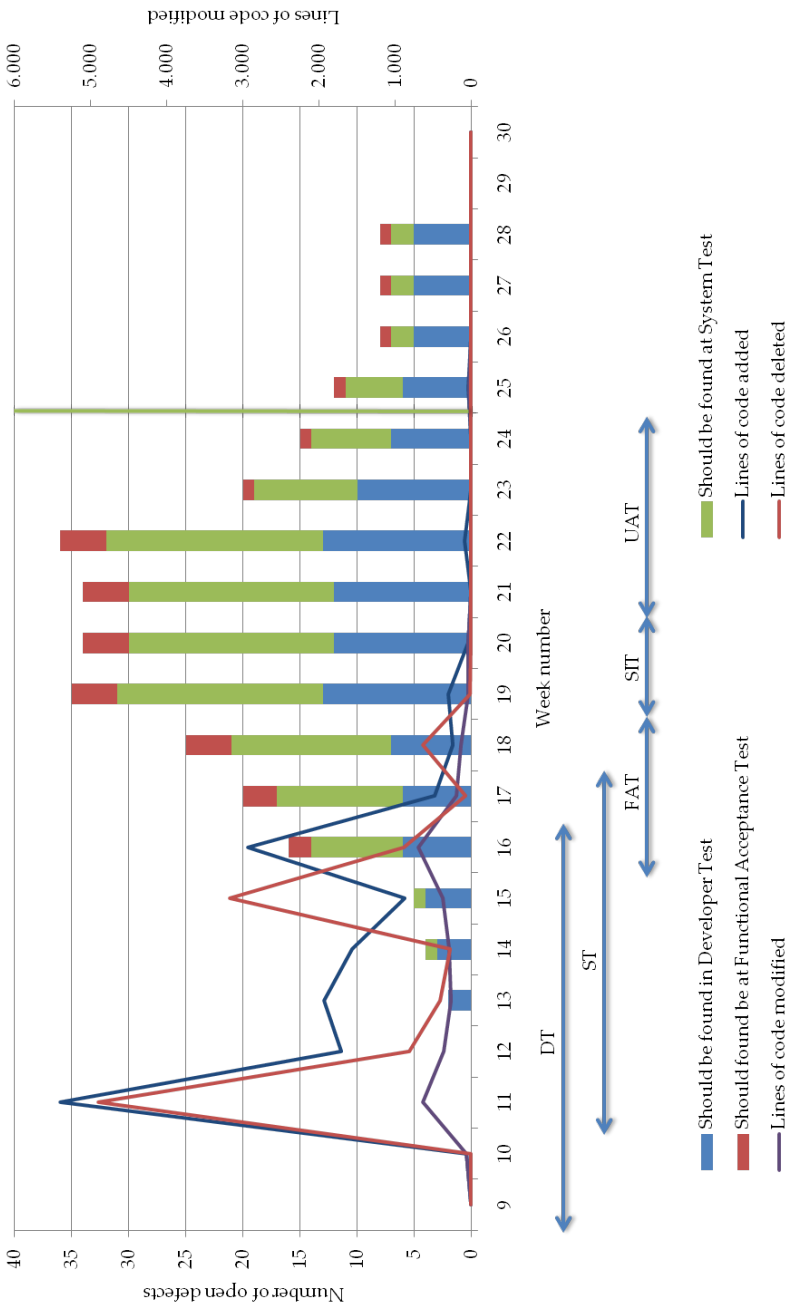

Figure 5.4: Possible relation between post-release Defect Density and Test Level Quality

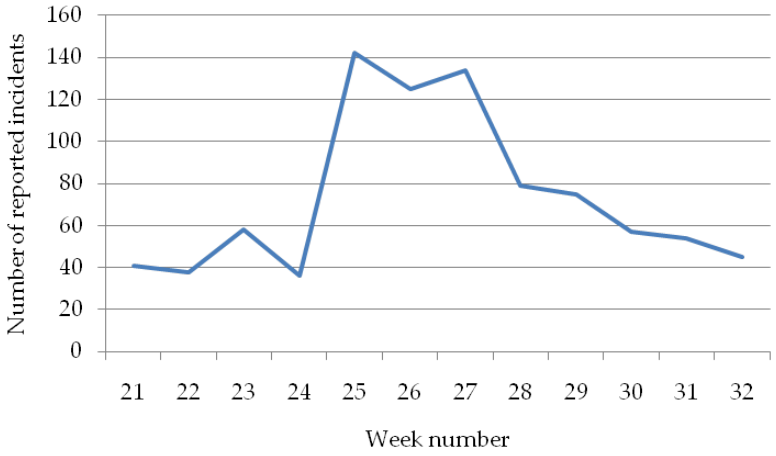

Figure 5.5: Number of reported incidents for project A

Chapter 6

Threats to validity

But which other factors can have an influence on the results of the research project? In order to draw meaningful conclusions an answer to this question is necessary. Also a clear understanding of the quality of the data is required. There is also a possible risk of the occurrence of research bias. What measures are taken to control these aspects is described in this chapter.

6.1 Influence of other factors

What is the influence of all the factors that were identified during the literature survey (See Section 2.6) during this research? Through interviews data has been collected about the other factors within the project. The results of these interviews are used to indicate to what extent the other factors can affect the conclusions drawn. The data collection forms of Freimut [18] are used during this research. Freimut creates a questionnaire (See appendix B) that is used to gather information about the other factors. Due to time constraints the interviews are limited to interviewing two team members per project.

The research analyzes four projects that are executed in the same context, namely four maintenance projects done by the same team for the same product (See also section 5.1). The assumption is that the context of all of the four projects is equivalent; this can be confirmed by the data collected during the interviews (See section appendix B).

Research data

Based on interviews the following can be concluded with regard to the different areas:

- *Specification and documentation process*: During the four projects, this aspect stayed constant. There were no changes in the team or the product. In addition, there were no improvements which had a significant influence on this area;

- *New functionality*: During the four projects, this aspect stayed constant. There were no changes in the team or the product. In addition, there were no improvements which had a significant influence on this area;
- *Design and development process*: During the four projects, this aspect stayed constant. There were no changes in the team or the product. In addition, there were no improvements which had a significant influence on this area;;
- *Testing and rework*: Only project A has lower testing quality according to the questionnaire. The other projects have the same testing quality;
- *Project Management*: During the four projects, this aspect stayed constant. There were no changes in the team or the product. In addition, there were no improvements which had a significant influence on this area;

It can be concluded that only the factor testing and rework deviates during project A. The root cause of this was that a new employee took on the role of System Tester while another employee left the role of System Tester.

Additionally, during the research the data from a completely different project is also analyzed. The formal model for the research described in section 4 is also applicable for this project without any modifications. However, many factors differ in this project compared to the other four projects. Examples of the differences were (a) the method of capturing the requirements, (b) working with an iterative project management approach instead of the waterfall approach. In an iterative approach the defects must be found in the same iteration as they were introduced. It was not possible to reproduce this information based on the Defect Registration System because this data was not captured in the registration system. By taking this information as a part of the template in the defect registration system it would be easy to do this analysis. Within the scope of this study it was not realistic to take this analysis into account.

6.2 Data quality

During the preparation of the defect classification model it has been found that the cause of erroneous defect classification for a large part (three quarters) was that the defects in the Defect Registration System had not been properly updated. Erroneous defect classification means that the researcher and the project member classify the defect differently. If the Defect Registration System has not been properly updated it either means that (a) no Code Check-in was linked by resolving the defect or (b) where a Code Check-in was assigned the check-in comments are missing.

Based on this information it is possible to determine the risk of misclassification for each defect. So it is an option to remove the defects with a higher risk of misclassification. But what does this mean for the representativeness of the used set of defects?

	Provisioned with meaningful comment	**Not provisioned with meaningful comment**
Code check-in assigned	25 defects	1 defects
No code check-in assigned	6 defects	9 (21%)

Table 6.1: Data Quality for project A

	Provisioned with meaningful comment	**Not provisioned with meaningful comment**
Code check-in assigned	37 defects	2 defects
No code check-in assigned	37 defects	7 (8%)

Table 6.2: Data Quality for project B

Research data

Table 6.1, 6.2, 6.3 and 6.4 provide more insight into the Data Quality. The risk of misclassification is higher when resolved defects do not have their Code Check-in linked and do not have meaningful comments. This applies to 14% of all the defects in all four of the projects. One possibility would be to eliminate these defects from of the set of defects to analyze. But what would that say about the representativeness of the set to be analyzed? The next section describes results in which a percentage of 85% of the defects are classified the same by the researcher and an independent team member. Based on this information the defects with a higher risk of misclassification remain a part of the research. For upcoming projects it would be better to register the defect type when resolving the defect, then more is known about the background of that defect.

	Provisioned with meaningful comment	**Not provisioned with meaningful comment**
Code check-in assigned	36 defects	6 defects
No code check-in assigned	24 defects	19 (22%)

Table 6.3: Data Quality for project C

	Provisioned with meaningful comment	**Not provisioned with meaningful comment**
Code check-in assigned	66 defects	0 defects
No code check-in assigned	19 defects	8 (9%)

Table 6.4: Data Quality for project D

6.3 Research bias

During this study the defect classification is performed by the researcher. A possible threat to validity is called research bias. Research bias occurs when the researcher influences the results of an investigation to suggest a certain conclusion. For example, classifying the defects in such a way as to make the hypothesis correct. To prevent this, the following measurement is taken: 10% of the defects are also classified by an independent person. This person was a project member of the project that is analyzed. This 10% percent of the defects can be compared with the results of the researcher. In this way it is possible to measure to what extent the classification of the researcher differs from the classification of the project member.

Research data

Of the 10% of the defects that are classified by both the researcher and the team member, Table 6.5 gives the percentages of the defects that are classified as a defect of the same defect type. The difference in the classification of some of the defects was caused by the fact that the team member remembered information that was not registered , and based on that information he chose another defect type. Therefore it would be better for upcoming projects to register the type of defect when resolving the defect.

Because 15% of the defects are not properly classified, the result of the calculation of the Test Level Quality can differ. This is because the $tooLate(l_i)$ component in the calculation of the Test Level Quality is impacted; this is because this component is based on the classification of the defect types. The $tooLate(l_i)$ component can be 7.5% higher or 7.5% lower. This is based on the 15% of the defects that are not classified the same. It is assumed that when all the defects are doubly classified; the percentage of defects that are classified the same will still be 85%. When this uncertainty is included in the calculation of the Test Level Quality, it is possible to recalculate the TLQ with an error range. The upper range is $tooLate(l_i)$ + 7.5% and lower range is $tooLate(l_i)$ - 7.5%. See for this results also Table 6.5.

Only the lower bound of the error range of Project C will leads to another classification of the Test Level Quality.

Fact	**Project A**	**Project B**	**Project C**	**Project D**
Percentage of defects classified the same by the researcher and the project member	100%	83%	80%	89%
TLQ	Very Low (0,4)	Low (2,0)	Low (1,5)	Low (2,2)
TLQ (-7.5%)	Very Low (0,4)	Low (1,8)	*Very Low (1,3)*	Low (2,0)
TLQ (+7.5%)	Very Low (1,1)	Low (2,3)	Low (1,8)	Low (2,4)

Table 6.5: Percentage misclassifications and TLQ with error range

6.4 Concluding remarks

During the research there were some threats to validity; but they are eliminated. The other factors stay constant for the four projects, except the quality of the system test is different. However, this has an identifiable cause (a new employee). The classification of the defects indicates that at on average 85% of the defects are classified identically by the researcher and the independent project member. The underlying reason for the misclassifications was that the project member could remember information that was not registered. It is therefore important for upcoming projects that this information be registered while resolving defects. In addition, the quality of the data is good enough for the investigation to be carried out; 14% of the defects has a higher risk of misclassification.

Chapter 7

Conclusions and further research

The main question for this study was: *How can the post-release Defect Density be estimated based on the Test Level Quality?* A hypothetical answer to this main question was given in the research plan and repeated in Section 1.2.2. During this research an empirical study validated the hypothesis; the results are described in the conclusions. New research issues are described the suggestions for further research.

7.1 Conclusion

The post-release Defect Density halves when the Test Level Quality increases from Very Low to Low. This is observed for the four projects that are investigated using an empirical study (See Chapter 5). To support this investigation a formal research model is created that defines the various terms used (See Chapter 4). This model is necessary for comparing the results of the four projects with each other. The model contains the definitions of the Test Level Quality and the post-release Defect Density, but also how these metrics for the different projects can be calculated. The formal research model can also be used for other projects.

Other factors can have an influence on the results of the research. These factors have already been described by other researchers, and they have also already done some exploratory research. Within the research it is determined through an interview that these factors for the four projects examined were constant. Only the quality of the System Test differs. This was, from the perspective of this study, desirable, because this resulted in another Test Level Quality for one of the projects. This is reducible in the results.

In addition, the Data Quality of the projects is good enough to carry out the investigation. In 14% of the defects, the risk of misclassification is higher. It is possible to identify research bias by an independent classification. The result

of classification was that on average 85% of the defects by the researcher and the independent project member were classified the same. The reason that the other 15% of the defects were classified incorrectly was that the project member remembered information that had not been recorded in the defect registration system. It is therefore important for subsequent projects to do the defect type classification when resolving the defect, and not after the project is finished (See Chapter 6).

Let us return to the hypothesis (See Section 1.2.2). Indeed it can be concluded that the post-release Defect Density decreases when the Test Level Quality increases. However, the conclusion is different from the hypothesis in two separate aspects:

- The Test Level Quality is Low or Very Low for the four projects. This contrasts with the expected Test Level Quality of Medium;
- When the Test Level Quality is Low or Very Low this leads to lower post-release Defect Density than expected. The possible cause could be that during the first month that the product is in the production environment, only the post-release defects are monitored.

So the post-release Defect Density can be estimated by using the Test Level Quality as defined in the formal research model. Currently this is only studied for the four projects that are executed in the same context. More reference-projects are needed to indicate that a particular Test Level Quality will lead to a certain number of defects per 1000 lines of source code.

7.2 Pointers for further research

Due to time constraints everything could not be investigated; this is why there are still some open issues for further research.

- In this study all of the projects that are examined have a Test Level Quality of Low or Very Low. Within the same context, projects can be examined with a Test Level Quality of Medium, High or Very High. To achieve that, the development and test team should be focused in finding defects of certain defect types in a particular test level. A first step to this can be the inclusion of the decision table defect types versus test level in the Master Test Plan. In addition, during the development and testing process it should continuously be evaluated whether the defects are found at the right stage. When the preliminary conclusion of the evaluation is that defects are still found too late, the process can possibly be adjusted. Defect Causal Analysis is a good method to carry out this evaluation.
- This research is limited to studies of projects in the same context. The same empirical research can also be performed for a project in a different context. The complexity that then arises and must be kept in mind is that the other factors are probably not constant . It is also wise to have

the defect classification done by a project member during the project, so as to make it a part of the defect registration process.

- This investigation is focused on finding defects before it's too late. But what can be concluded when many defects are found too early. It is hypothetically assumed that in that case too much is being tested. The tests are not efficiently executed and this results in higher costs. Again, further follow-up study can be conducted for this question.

References

[1] Basili, V.R., Caldiera, G., Rombach, H.D., The goal question metric approach, `http://www.cs.toronto.edu/~sme/CSC444F/handouts/GQM-paper.pdf` (16 December 2010)

[2] Basili, V.R., Weiss, D.M., A methodology for collecting valid software engineering data, *IEEE Transactions on Software Engineering*, pages 728 - 738, 1984

[3] Basili, V.R., Weiss, D.M., Evaluating software development by analysis of changes: Some data from the Software Engineering Labatory, *IEEE Transaction on Software Engineering*, pages 157 - 168, 1985

[4] Batini, C., Scannapieco, M., Data Quality, concepts, methodologies and techniques, *Springer*, pages 19 - 49, 161 - 200, 2006

[5] Buijs, A, Statistiek om mee te werken, *Wolters Noordhoff*, pages 20 - 61, 2002

[6] Cangussu, J.W., Karchich, R.M., Software Release Control using Defect Based Quality Estimation, *International Symposium on Software Reliability Engineering*, pages 440 - 450, 2004

[7] Card, D.N., Learning from our mistakes with Defect Causal Analysis, *IEEE Software*, pages 56 - 63, 1998

[8] Chillarege, R., Prasad, K.R., Test and development process retrospective a case study using ODC Triggers, *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 669 - 678, 2002

[9] Chillarege, R., Bhandari, I.S., Chaar, J.K., Halliday, M.J., Moebus, D.S., Ray, B.K., Wong, M.Y., Orthogonal Defect Classification - "A Concept for In-Process Measurement", *IEEE Transaction on Software Engineering*, pages 943 - 956, 1992

[10] Damm, L.O., Lundberg, L., Company-wide implementation of metrics for early-software fault detection, *IEEE International Conference of Software Engineering*, pages 560 - 570, 2007

[11] Ekanayake, J., Tappolet, J., Gall, H.C., Bernstein, A., Tracking concept drift of software projects using defect prediction quality, *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*, pages 51 - 60, 2009

[12] Fenton, N.E., Software Metrics, a rigorous approach, *Publisher: Chapman & Hall*, London, pages 1 - 320, 1991

[13] Fenton, N., Neil, M., A critique of software defect prediction models, *IEEE Transactions on Software Engineering, Volume 25 Issue 5*, pages 675 - 689, 1999

[14] Fenton, N., Ohlsson, N., Quantitative analysis of faults and failures in a complex software system, *IEEE Transactions on software engineering, Volume 26 Issue 8*, pages 797 - 814, 2000

[15] Fenton, N., Krause, P., Neil, M., Software Measurement uncertainty and causal measurement, *IEEE Software, July / August*, pages 116 - 122, 2002

[16] Fenton, N., Neil, M., Marsh, W., Hearty, P., Radlinski, L., Project data incorporating qualitative factors for improved software defect prediction, *Third international workshop on predictor models in software engineering*, pages 69 - 79, 2007

[17] Fenton, N.E., Neil, M., Caballero, J.G., Using ranked nodes to model qualitative judgments in Bayesian networks, *IEEE Transactions on knowledge and software engineering, Volume 19 Issue 10*, pages 1420 - 1432, 2007

[18] Freimut, B., Denger, C., Ketterer, M., An industrial case study of implementing and validating defect classification for process improvement and quality management, *IEEE International Software Metrics Symposium*, pages 19 - 30, 2005

[19] Heemstra, F.J., Kusters, R.J., Trienekens, J.J.M., Software kwaliteit, op weg naar beter software (in Dutch), *ten Hagen Stam uitgevers*, pages 3 - 60, 2001

[20] Humprey, W.S., Introduction to the Personal Software Process, *Addison Wesley*, pages 1 - 304, 1997

[21] Koomen, Tim, Aalst, Leo van der, Broekman, Bart, Vroon, Michiel, "TMap Next for result-driven testing". *UTN Publishers*, pages 55 - 148, 2007

[22] McConnel, S., Code Complete, a practical handbook for software construction, *Microsoft*, pages 463 - 534, 649 - 660, 2004

[23] McConnel, S., Professional Software Development, Shorter Schedules, Higher Quality Products, More Successful Projects, Enhanced Careers, *Microsoft*, pages 1 - 272, 2003

[24] Mietes, D.M., Methodological foundation and the research approach, http://dissertations.ub.rug.nl/FILES/faculties/management/1994/d.m.mietus/c3.pdf (12 December 2010), 1994

[25] Universiteit van Amsterdam, template for a Masters project, 2010

[26] Vliet, H. van, Software Engineering, Principles and Practice, *John Wiley and Sons*, pages 101 - 142, 2002

[27] Westfall, L., 12 steps to useful software metrics, http://www.westfallteam.com/Papers/12_steps_paper.pdf (16 January 2011)

[28] Weiss, D.M., Basili, V.R., Evaluating software development by analysis of changes: some data from the software engineering laboratory, *IEEE Transaction on Software Engineering, Volume. SE-11, No. 2*, pages 157 - 168, 1985

Appendix A

Data collection and analysis

For this research project the data of five projects was collected and analyzed. The goal of this appendix is to describe as complete as possible the process of data collection and analysis. By doing this it is transparent how the results of the research are obtained. In the first section (see A.1) the data collection process is described. Section **??** describes how the data of the different projects will be analyzed and compared. For each project the results are documented in Excel sheets.

A.1 Data collection

In section A.1.1 are the main steps for each project described. The subsequent sections are refinements of the main steps:

- Defect classification (see section A.1.2);
- Defect type classification (see section A.1.3);
- Classification if a defect is found too late (see section A.1.4).

A.1.1 Main steps

The steps that are depicted in the figure A.1 below are executed for every project during this research.

A.1.2 Defect classification

All the defects of each project will be classified by the researcher. For every defect the steps are executed that are depicted in figure A.2. Ten percent of the defects will also be independent classified by a team member. The results of both classifications will be compared and reported.

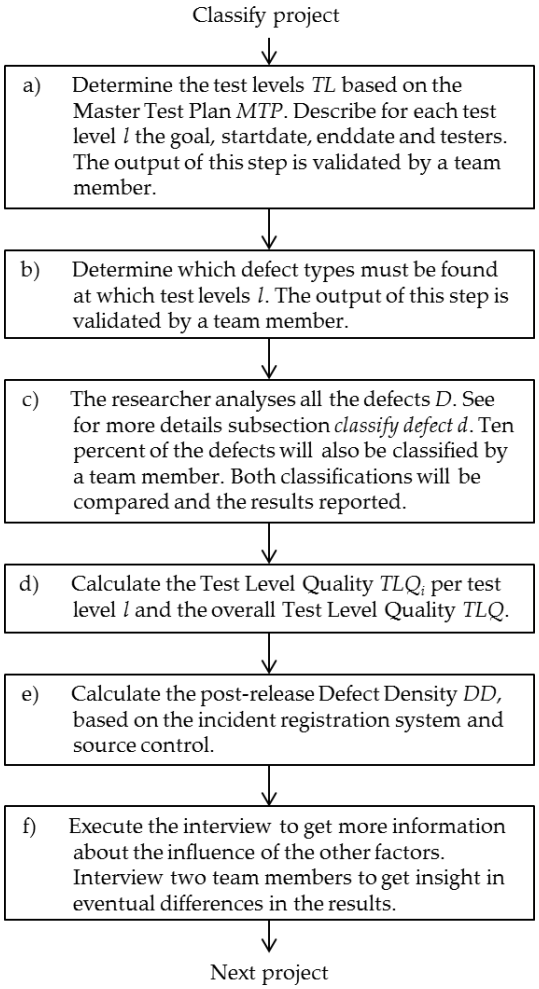

Figure A.1: Steps that are executed for the data collection of a project

A.1.3 Defect type classification

Based on the information in Souce Control and the Defect Registration System the defect type can be determined. For every defect the researcher executes the flow depicted in the picture A.3 below. For more information about the different defect types see table A.1.3 and table A.1.3.

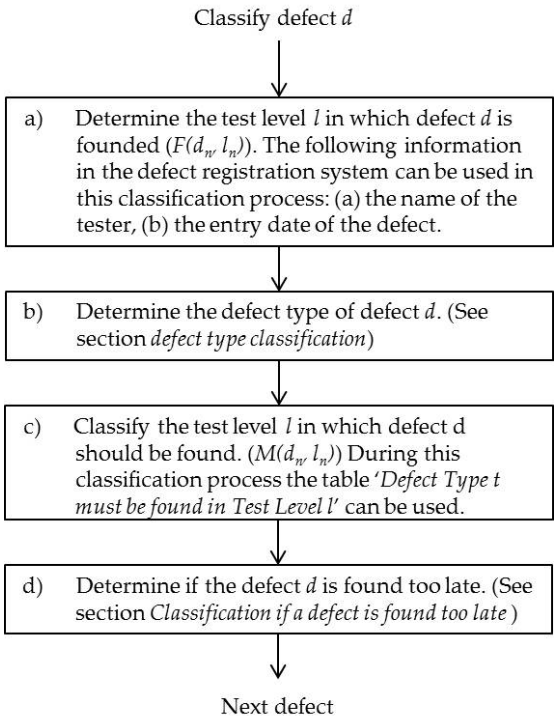

Figure A.2: Steps that are executed for the classification of a defect

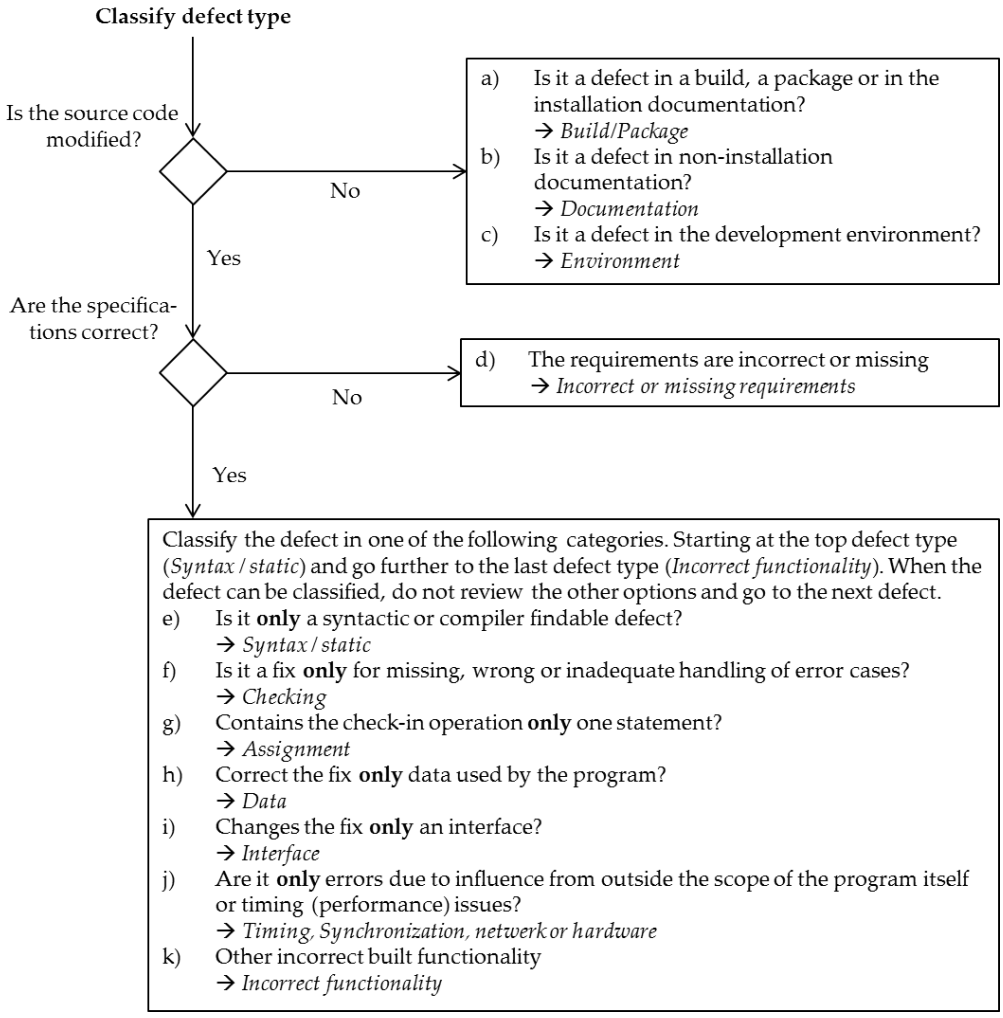

Figure A.3: Steps that are executed for the classification of a defect type

Defect type	**Description**	**Characteristic of the fix**	**Example**
Build/package	Errors in version control, change process or build/packaging system. Also mistakes in installation documentation belongs to this defect type.	Correct problems in version control system, change process, build / packaging system or installation documentation.	For example (a) the wrong versions of sub components are used during the packaging process, (b) fixing an error during the build, (c) wrong labels are set in Source Control.
Documentation	Documents or comments are misunderstandable or wrong. This kind of defects doesnt lead to a code modification.	Only correct the documentation or comments, not source code.	The documentation doesnt describe the functionality that is built correctly.
Environment	Defect in the development environment or support system	Correct the support system or avoid the development environment defect.	For example (a) wrong versions are installed on development or test infrastructure or (b) the wrong mainframe is referenced by a service in the test environment.
Incorrect or missing requirements	Defects in functionality caused by wrong specifications.	Write the missing specifications or correct the wrong specifications. Thereafter implement those specifications.	For example in the specifications it's not described that the application should have the feature that a specific entity can be deleted.

Table A.1: Defect types, part A

Defect type	**Description**	**Characteristic of the fix**	**Example**
Syntax/static	Defects that are usually detected by a compiler.	Correct a syntactic or compiler-findable static semantic defect	For example: the wrong XML syntax is used.
Checking	Missing, wrong or inadequate handling of error cases. A frequently occuring problem is missing input validation of functions.	Add or correct error handling or add or correct function input validation	For example (a) the value of a parameter of a function isnt validated or (b) an error case isnt correctly handled.
Assignment	One statement defects in data management or procedure calls.	Correct one statement.	For example: (a) the use of a wrong operand or operator in expression. (b) A wrong object is assigned to a variable. (c) An assignment is missing or duplicated. (d) A call to wrong procedure. (e) A missing call.
Data	Data that is used by the program is the cause of the defect.	Correct the data.	For example a list with values doesnt contain a specific item.
Interface	Wrong design of interfaces (not conform specifications or programming guidelines).	Change the interface.	For example a service or data contract interface is incomplete, wrong or inappropriate.
Timing, synchronization, network or hardware	Errors due to influence from outside the scope of the program itself or timing issues.	Adapt the program to properly handle these influences.	The performance of the application doesnt meet the specified requirements.
Incorrect functionality (correct specifications)	Defects beyond one statement in functionality. The requirements are correctly specified.	Implement the missing or incorrect functionality.	For example in the specification it's described that the application should have the feature that persons can be deleted. The application doesnt contain that feature.

Table A.2: Defect types, part B

Sequence number	Name	Goal	Startdate	Enddate	Tester
1	Developer Test	Find checking type defects	01-04-2011	01-05-2011	Jim
2	System Test	Find defects related to software that is built not conform to the specifications.	15-04-2011	15-05-2011	Jane
3	Factory Acceptance Test	Find defects related to software that is built not conform to the expectations.	15-05-2011	01-06-2011	Kim

Table A.3: Example of different test levels

A.1.4 Classification of a defect is found too late

How is it determined whether a defect is found too late? For every defect it is known in which test level the defect was found ($F(d_n, l_n)$) and in which test level it should be found ($M(d_n, l_n)$). Each test level l has a sequence number. The sequence number indicates the sequence of the test levels TL. Based on this sequence numbers it can be determined whether a defect is found too late.

Example

Defect d_{21} is found during the System Test level. See table A.1.4 for the different test levels. The defect d_{21} is classified as a Checking defect type (see previous section). Therefor defect d_{21} should have been found during the Developer Test. The sequence number of the test level in which the defect d_{21} is found is 1; the sequence number of the test level in which defect d_{21} should have been found is 2; so the defect d_{21} is found too late.

A.2 Data analysis

The results for the TLQ and post-release Defect Density of each project are plotted into a graph (see Figure A.4). The study expects to find such a trend that the graph is displayed.

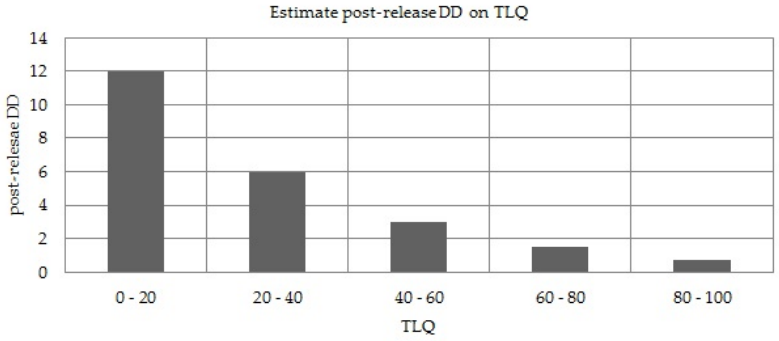

Figure A.4: Possible relation between post-release Defect Density and Test Level Quality

Appendix B

Influence of other factors

The following questionnaire is used to get more information about the influence of different factors (see figure B.1) on the Defect Insertion and Defect Discovery. The list is based on the questionnaire that is described by Freimut [18]. The sections below describe the questions that are asked for the several factors that are depicted in the picture. Any question can be answered with the following values (ordinal scale): Very High, High, Medium, Low, Very Low.

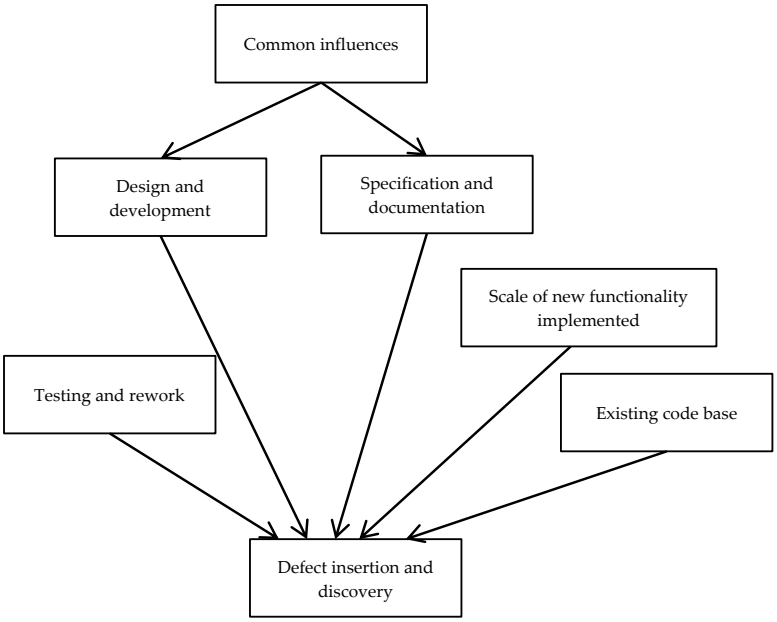

Figure B.1: Causal relationships to defect insertion and defect discovery

B.1 Specification and documenation

1. How would you rate the experience and skill set of your team members for executing this project during the requirements and specifications phase?
2. How would you rate the quality of the requirements given by the client or other groups?
3. Have all the Requirements, Design Documents and Test Specifications been reviewed in the project?
4. In your opinion, how effective was the review procedure?
5. What was the review effectiveness in the project for the requirements phase?
6. In your opinion, is the defect density of spec reviews on the high side?
7. How stable were the requirements in your project?

B.2 New functionality

1. What was the complexity of the new development or new features that happened in your project?
2. How large was the extent of working on new functionality rather than just enhancing the older functionalities in your project?
3. For your product domain, would you rate the total no of outputs/inputs (newly developed / enhanced) as high?

B.3 Design and development process

1. How would you rate the experience and skill set of your team members for executing this project during the design and development phase?
2. On an average, how would you assess the Quality of code produced by the team members?
3. What was the review effectiveness in the project for the Design and Development phase?
4. What is your opinion about the motivation levels of your team members?

B.4 Testing and rework

1. How effective was the testing process adopted by your project?
2. What was the level of software test competence of those performing the unit test?
3. How would you rate the experience and skill set of the independent test engineers (Integration, functional or subsystem testing, Alpha, Beta)?
4. What was the extent of the defects that were found using formal testing against the intuitive / random testing?

B.5 Project management

1. What is the coverage of the identified project / process related trainings as well as trainings identified as per the roles, by the team members?
2. How effective is the projects document management and configuration management?
3. Has the project planning been done adequately?
4. How many sites/groups were involved in the project.
5. To what extent were the key project stakeholders involved?
6. How good was customer interaction in the project?
7. How would you rate the Vendor / Sub-contractor Management (if applicable)?
8. How would you the rate the quality of internal interactions / communication within the team?
9. Whats your opinion about process maturity in the project?