AN INVESTIGATION INTO TELECOMMUNICATIONS BILLING SYSTEM TESTING PROCESSES

Vitesh J Jinabhai

A Dissertation submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, in fulfilment of the requirements of the degree of Master of Science in Engineering

Johannesburg 2012

Declaration

I declare that this dissertation is my own, unaided work, except where otherwise acknowledged. It is being submitted for the degree of Master of Science in Engineering at the University of Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

_____

Vitesh J. Jinabhai

Signed this ____ day of _____ 2012

Abstract

Testing is an important part of the software development process, since it ultimately determines the quality of the product or service that the end user is provided. As error correction costs increase exponentially with time, it is important to resolve software defects as early as possible. The same applies to telecommunications billing software, where the level of competitiveness demands that the testing process be both accurate and efficient. The investigation undertaken aimed to evaluate and improve the testing process of a company that develops telecommunications billing software, Nokia Siemens Networks (NSN). The methodology used to perform the study involved the use of the Goal Question Metric (GQM) approach, which has been used extensively for process measurement and improvement. A research model was developed which derived process goals from the key research questions, ensuring that the research questions could be answered from the goal results. Four goals were determined using this method. These goals were to improve defect detection accuracy, defect correction accuracy, defect detection efficiency and defect correction efficiency. This led to 14 questions and 95 metrics in total. Defect detection accuracy was found to be insufficient, defect correction accuracy was determined to be satisfactory, defect detection efficiency was a key goal, and it was found to be unsatisfactory, while defect correction efficiency was acceptable, however there were many cases where error resolution was slow. Several specific proposals for improvement are suggested, as well as general process improvement suggestions. The process can be improved overall by using the agile Scrum approach. Scrum's cross-functional teams coupled with development testing through Test-driven Development will ensure that detection accuracy and efficiency are improved. The study found that because the process is more traditional than agile and separates testing and development, it is not well suited to the size of the projects and their timelines. In order to meet the needs of the industry and release quality services competitively, a more agile approach needs to be taken. The research conducted provides a contribution to a field where research is scarce, and provides evidence of the insufficiency of traditional development processes in small telecommunications projects, while motivating the use of agile methodologies to meet organisational goals.

Acknowledgement

The author wishes to thank Prof Rex van Olst for his guidance and supervision during the course of the research. Thanks are also extended to Herman Kotze, Thandanani Mbanjwa, Vassen Moodley and Umar Dockrat of Nokia Siemens Networks for their assistance with the collection of data, as well as with familiarisation of the organisations processes. Their invaluable assistance ensured the successful completion of the investigation.

Contents

List of Figures

List of Tables

1 Introduction

1.1 The Importance of Software Testing in Telecommunications

Testing is one of the most important phases in the software development life cycle. This is because testing has a big impact on the quality of the product that reaches the end user, more so than development. Since few development processes produce a product free of flaws, the testing process is relied upon to ensure that the resulting product meets the required quality standards. The quality of the product determines the amount of satisfaction the end user has, as well as the amount of resources spent on correcting its flaws.

The testing of software in telecommunications is no different, since the telecommunications industry relies heavily on software, and therefore testing is a key aspect of its development process. In fact, it is more critical to ensure that telecommunications software has as few defects as possible, due to the competitive nature of the industry. This competitiveness also demands that the testing process be as efficient as possible, to ensure that new products and services are released timeously. The consequence of inadequate testing is a loss of revenue on several fronts, in the long and short term. It is a well-known fact that the cost of correcting an error increases considerably as the development cycle progresses. Errors not found by the testing process cost far more to correct once they have reached the customer. There are numerous examples of this in many industries, showing that the consequences of an inadequate testing process are not only costly in terms of revenue, but also in terms of security and customer satisfaction.

The need for adequate testing is even greater in the case of billing software. This is because billing directly affects revenue, and errors in this area have the potential to be extremely costly. The testing process must also be especially efficient at testing billing software, since every new product or service is linked to a billing system. In order to maintain a competitive advantage, these products and services must be released as soon as possible. A balance must therefore be obtained in which the testing process is both efficient and accurate, so that the entire development

process executes in as short a time as possible, while producing a product with as few errors as possible.

1.2 The Focus of the Investigation

The research aims to contribute to the field of software testing in telecommunications in terms of the suitability of the current software testing process for the industry. The research will determine the adequacy of the current process and motivate the use of an improved process. The research carried out specifically examines the software testing process at Nokia Siemens Networks (NSN), a company that develops billing software for telecommunications companies, and aims to propose improvements to the process based on the findings. Figure 1 shows a generalised billing process, including the sub-processes that execute and the resulting billing artefacts they produce. A billing system operates by collecting and aggregating service usage data for a particular user from the network itself, identifying and calculating the charges, compiling the charges, applying taxes, and rendering the bill on the user account [1]. Rating involves assigning a cost to the service usage based on various criteria such as the time of day, type of user account, etc [1]. Charging is the process of applying these costs to the user account [1]. NSN offers billing software that focuses on the rating and charging aspects of billing.

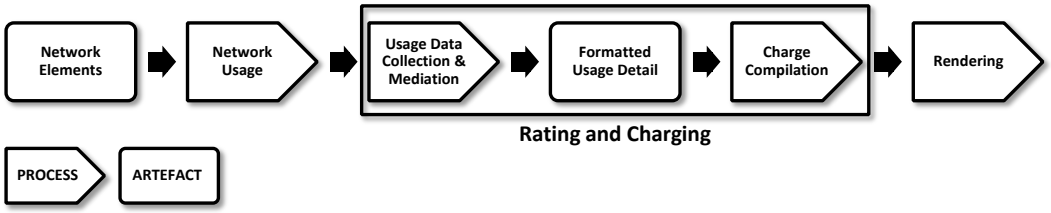

Figure 1: A generalised telecommunications billing process showing the various processes and the artefacts produced, highlighting the sub-processes executed by the billing software of the company (adapted from [1])

1.2.1 The Software Developed

As discussed, the software developed focuses on rating and charging. At this level in the billing system, usage data has been collected and formatted. This usage now needs to be rated based on several criteria. These criteria can include the type of billing package the user is using, the time of day should special rates apply or whether or not any bundle rates apply. Once the usage is rated, charges are calculated based on the criteria

taken into account. The resulting charges are then deducted from the user account.

The actual software that performs these operations is composed of several standard base modules, as well as modifiable modules, which are changed in each project to achieve the required functionality according to the client's preferences [2]. The software is coded in a proprietary language based on other object-oriented languages. The structure of a software package is shown in Figure 2. The core modules remain the same for all projects and are standardised logic modules that provide basic functionality [2]. The customisable modules capture the specific billing requirements of the client [2]. The modifiable modules consist of the client's own billing parameters for the package that is being offered to their users. These billing parameters consist of tariffs for the billing package [3]. Tariffs are related to the packages offered, and can be related to SMS bundles, voice bundles, off-peak discounts, etc [3]. Subroutines are coded in order to execute specific functions based on the tariffs [3]. These functions can include checking the account balance, checking if an account is active, applying charges to an account, etc [3]. The software is developed and tested off site using a simulation package [3]. When software development and testing is complete, it is installed on the client's equipment and tested with the client on site [3]. The software is installed on a carrier-grade server computer.

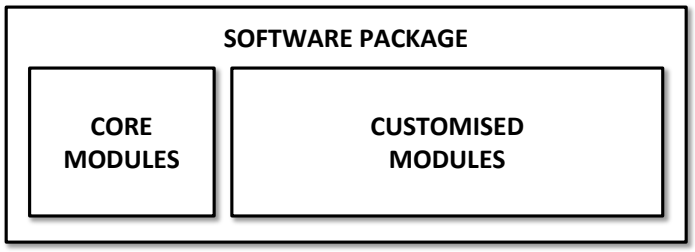

Figure 2: The structure of a software package that is developed (adapted from [2])

1.2.2 The Software Development and Testing Processes

The solution development process is summarised in Figure 3. This process is followed for small projects that aim to introduce minor additions to existing billing products. As such, the projects undertaken usually take three to four months to complete. After consultation with the client and

their requirements being determined, the solution is designed [3]. The design phase involves the design and coding of the required modules for the project [3]. A standard software platform is used for all solutions developed, and as discussed, both core and modifiable modules for billing functionality exist. Modifiable modules are changed to meet the specific requirements of each project [2].

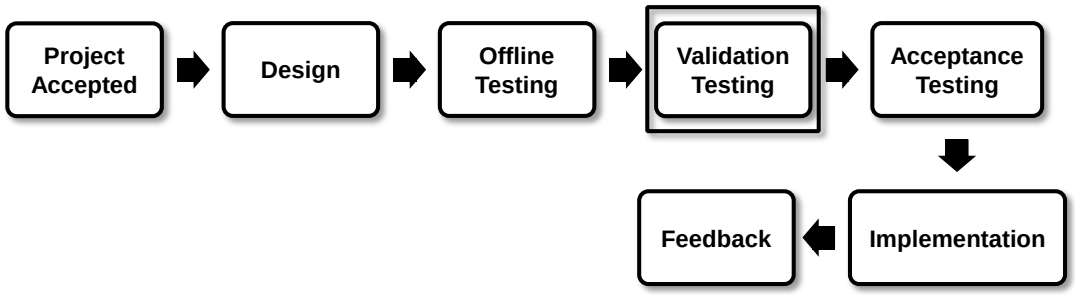

Figure 3: The software development process, highlighting the validation testing phase as the focus of the research [3]

Once development is complete, offline testing is performed by developers, and testers may assist with this process [3]. When offline testing is complete, preparations are made for validation testing [3]. The validation testing phase is the focus of this study. Acceptance testing is performed jointly with the client [3]. Once all testing is complete, the software is installed on the client's systems [3]. After implementation, a feedback session is scheduled with the client in order to evaluate the project, and examine any issues that were encountered during the project [3]. The validation testing process is summarised in Figure 4. The required end-to-end testing is performed based on test cases defined [3]. Additional testing is performed if necessary, and this may include, for example, performance testing [3]. An initial release of the solution is provided to the client for approval, and this forms part of acceptance testing [3]. All the required documentation relating to the operation of the system is then reviewed [3].

Figure 4: The main steps of the validation testing process [3]

The development process followed by the organisation closely follows an iterative and incremental lifecycle model. This is because the process is executed in a fixed sequence of steps for each incremental addition of functionality. The process also borrows elements such as customer communication and customer evaluation from the spiral lifecycle model. Lifecycle models are discussed in further detail in the following chapter.

1.2.3 The Error Correction Process

A Test Management System (TMS) is used to log errors and track the resolution process. The TMS records the following information:

- Steps taken during error resolution
- The dates and times of each resolution step
- The person associated with each step
- Any details regarding each step taken
- The test cases associated with an error
- Project Information such as:
    - Milestones and their dates
    - Test cases for each project

There is therefore a complete log of the entire resolution process followed for each error of every project, as well as information regarding when testing started and ended. A basic error correction process is followed once an error is found. This process is depicted in Figure 5. The error is first logged by a tester, after which it is analysed by a developer. The analysis process may involve examining any symptoms of the error. Once the cause is determined, the error is corrected. It is the developer's responsibility to determine if the implemented solution is adequate. Once the solution is verified, the error is closed by the tester. The solution is then delivered to the client by the customer liaison in the organisation. The client may then verify the solution on their systems.

Figure 5: The Error Correction Process [3]

1.3 The Research Questions and Research Process

The key research questions are focused on determining two things:

1. Is the current testing process adequate in terms of its
    a. Accuracy?
    b. Efficiency?
2. If not, what improvements can be made?

In order to answer these research questions, the Goal Question Metric (GQM) approach was used. This method was developed in the 1980s and has been used extensively in industry since then for process measurement and improvement [4]. The method involves defining high level goals as a starting point, such as the improvement of a process. The second step involves determining questions to ask which characterise the goal. Using these questions, metrics are defined which answer them. In this way, only relevant measurement data is collected and it is associated with a high level goal, which can be evaluated by answering the corresponding questions. The integration of the GQM approach into the investigation is shown in Figure 6.

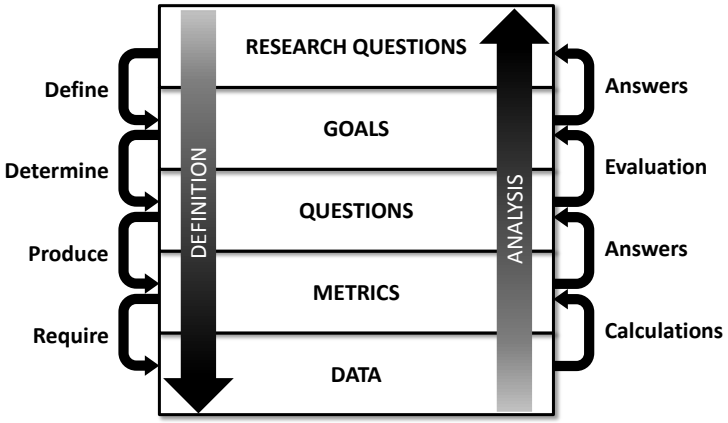

Figure 6: The research model used for the investigation

Definition was done in a top down manner. Using the research questions as a basis, goals were defined. The goals determine the GQM questions that need to be asked for their evaluation. Each question produced a set of metrics that were used to answer it. The metrics then required specific process data in order to be calculated. Analysis was done using a bottom up approach. Once the data was collected, calculations converted it to metrics that are more useful. Analysis of the metrics allowed the GQM questions to be answered. The answers of the questions for each goal

resulted in the goal being evaluated or reached. The examination of each goal then allowed the research questions to be answered. In order to provide a context for the investigation, initial metrics were proposed before the goals were defined. These metrics were based on relevant literature surveyed. The initial metrics were kept in mind while examining the process documentation and project documentation, as well as when determining the extent of the data to collect.

1.3.1 Data Collected

As discussed, most of the test process data was available on a test management system. It was determined that since the project sizes are small, the investigation will require ten projects to examine in order to assess the process accurately. Because the current version of the development process is recent and there were not enough projects for a single client making use of this process, the projects were chosen for two clients. Five projects were chosen for each client. The projects were undertaken in a period of two years. The projects lasted two to six months excluding the time spent correcting errors found after the release of the solutions developed. In addition to information regarding errors and project milestones obtained from the TMS, testing cost data was obtained from testers and the size of each project was obtained from developers.

1.4 Summary and Outline

The problem has been presented, and the research questions have been stated. The aims and focus of the study have also been discussed. The following chapter surveys the literature relevant to the study. The research methodology, which focuses on the GQM process, is subsequently discussed in chapter 3. The next chapter then presents the results of the study, which include the application of the GQM process, and the results thereof. Chapter 5 then discusses improvement proposals based on the results of the previous chapter. The concluding chapter then presents an assessment of the study, highlights its key results, outlines the possibilities of further work, and discusses broader trends observed by the study.

2 Survey of Literature

A survey of the literature related to all the aspects of the study has been carried out. In addition to the background pertaining to software development lifecycles, the use of metrics and the Goal Question Metric (GQM) approach, similar studies that have been undertaken are presented, as well as studies that motivate this investigation. The literature reviewed can be divided into five topics, namely:

- Software Development Models and Methodologies
- The Cost of Inadequate Software Testing
- The Need for Software Process Improvement in the Telecommunications Industry
- The Use of Metrics for Process Measurement
- The GQM Method and its Use in Process Improvement

Although the literature concerning software process improvement is vast, these five topics encompass the main aspects of the investigation carried out, as well as provide motivation for the study. It should be noted that literature regarding both software development and software process improvement in telecommunications billing is scarce, and this is most probably due to competitiveness in the industry, which limits the amount of widely available research, as well as the relative novelty of process improvement in the area of telecommunications billing software.

2.1 Software Development Models and Methodologies

The development of software follows a life cycle of activities that begins with requirements specification and ends with delivery. Milestones are used to manage the progress of development [5]. There are several lifecycle models that have been developed, which are characterised by:

- Team size
- Project size
- Primary objective
- Flexibility
- Level of Assurance

Lifecycle models can be seen as templates for methodologies or processes, since processes are derived from these models. Based on these

characteristics and an organisation's requirements, different methodologies are employed for different types of software development projects. This section briefly discusses several prominent lifecycle models and their characteristics.

2.1.1 The Linear Sequential Model

The linear sequential model was first proposed by Royce in the 1970s, and it is more commonly known as the waterfall model [6]. As seen in Figure 7, this model follows a sequential process of steps that include analysis, design, code and test activities [6].

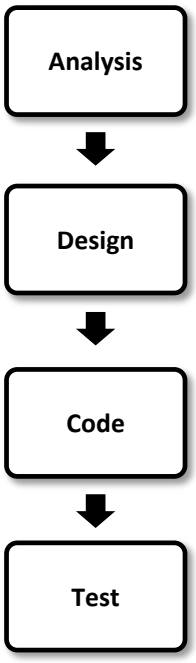

Figure 7: The linear sequential lifecycle model (adapted from [5])

Each activity may be undertaken iteratively, and there is accommodation for feedback in the original model, however, there are still shortcomings with this model, mainly that if requirements are changed during the course of development (as is often the case), the model does not accommodate this well [5]. Additionally, if a phase is delayed for any reason, the subsequent phases all become delayed, with the teams associated with those activities being inactive until the preceding phases are complete [5]. Another issue with this process is that the customer only receives a tangible product at the end of the cycle, which limits the amount of

feedback that they can provide, increasing the impact of requirements stage errors. Although this model is flawed, it is preferable to employing a disorganised approach to development. Later lifecycle models have aimed to address the weaknesses of this model.

2.1.2 Iterative and Incremental Development Methodologies

Iterative and incremental development models and methodologies were proposed to mitigate the shortcomings of the waterfall model. These models follow a similar sequence of activities to the linear sequential lifecycle, but incorporate iterations of activities or groups of activities, as well as development of software in working iterations. Iterative and incremental development models and methodologies include:

- The Spiral Model
- The Rational Unified Process (RUP)
- Agile Methodologies

These methodologies can be seen as evolutionary process models, since they view software as evolving as opposed to static [5]. The evolutionary nature of software is thus modelled in terms of iterations. The methodologies listed above are discussed in further detail.

2.1.3 The Spiral Model

The spiral model was proposed by Boehm in 1988 [5]. The spiral model is based on iterative and incremental development, and rapid development of software is accommodated. This model is illustrated in Figure 8. The spiral model makes use of iterations that cycle through phases known as task regions [5]. These task regions may include customer communication, planning, risk analysis, engineering, and construction and release, and customer evaluation [5]. Each task region is composed of a task set, which can be adapted to the meet the needs of the organisation in terms of formality [5]. The process begins at the centre of the spiral and progresses through each task region. Risk analysis is an important part of the model, and allows the model to be used for large projects. The software process is controlled by identifying and mitigating risks [5]. Due to the iterative nature of the spiral model, it can be used for every subsequent improvement or modification to a software package until the software is retired [5].

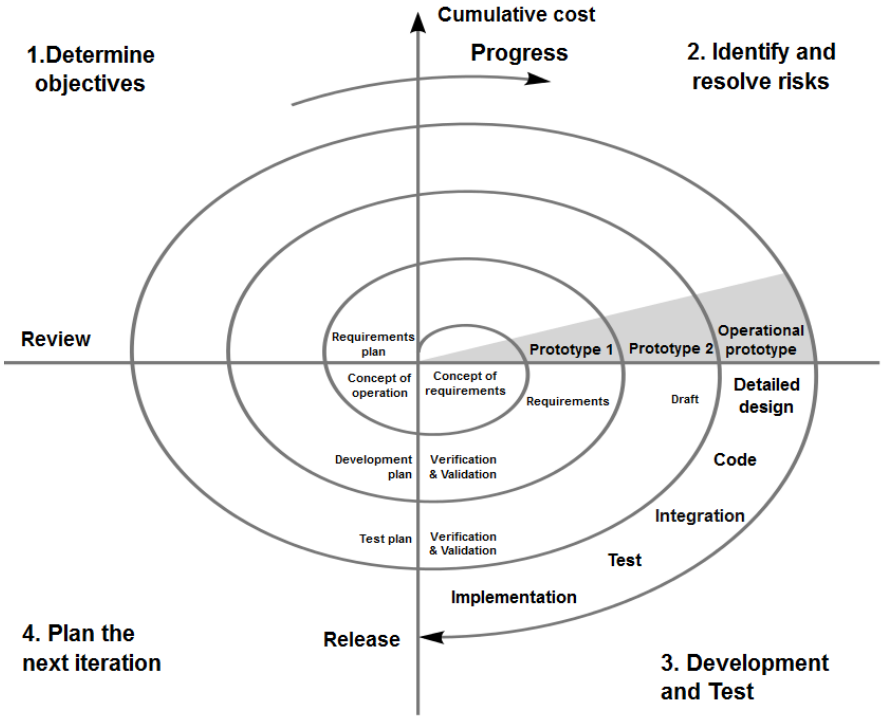

Figure 8: The spiral model [7]

2.1.4 The Rational Unified Process

The Rational Unified Process (RUP) is a process framework that is meant to be adapted to suit the needs of an organisation. The process framework is depicted in Figure 9. RUP is iterative in that there are four phases that place varying focus on different disciplines [8]. The phases are on the horizontal axis of the diagram, while the disciplines are on the vertical axis. The end of each iteration is shown at the bottom, and it produces some kind of deliverable, either external or internal [8].

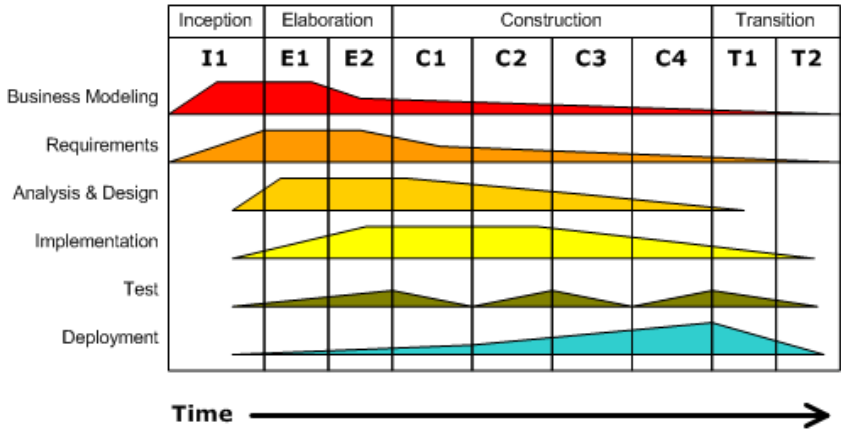

Figure 9: The Rational Unified Process framework [9]

The graphs in each discipline represent the amount of effort spent on each phase of that discipline. In addition to being iterative, RUP is also risk based, and emphasises risk analysis to ensure successful project completion [8]. The inception phase lays the foundation for the following project phases, and involves extensive planning. The elaboration phase aims to analyse any potential risks, determine the system architecture, and to produce a development plan for the project. The product produced by this phase must meet certain criteria to continue development. If these criteria are not met, the project can be redesigned [8]. The construction phase involves the development of a working iteration of the product. The final phase of RUP is the transition phase. The transition phase assesses the product to ensure it meets the user requirements. The transition phase also accommodates additions of minor functionality through additional iterations. Beta testing and user training is also performed during this phase. Although RUP encompasses many processes and activities, an organisation has the freedom to select only the elements of the framework that they require.

2.1.5 Agile Methodologies

Agile software development methodologies were developed in order to manage the dynamic nature of software development projects. Agile methodologies are characterised by their adaptability and short project cycles, and are based on iterative and incremental development. The Manifesto for Agile Software Development was published in 2001, and lists the following twelve principles of agile software development [10]:

- Customer satisfaction
- Welcoming changing requirements
- Frequent delivery of software
- Daily collaboration between business people and developers
- Support and trust motivated individuals to get the job done
- Face-to-face conversation is the best method of communicating
- The primary measure of progress is working software
- Sustainable development and the ability of all parties to work at a constant pace
- Continuous attention to technical excellence and good design

- Simplicity
- Self-organizing teams
- Regular assessment and improvement of team effectiveness

These principles summarise the nature of agile methods. Boehm and Turner summarised the suitability and characteristics of agile methods, and plan-driven methods such as the waterfall model, this is shown in Table 1 [11]. There are many methods and practices that form part of agile methodologies, the most notable are:

- Extreme Programming
- Crystal Clear
- Scrum
- Test-Driven Development

Although agile methods are promising and have clear benefits, they lack the structure and assurance that plan-driven methods have. Their suitability for projects should be examined based on the characteristics listed in Table 1.

Table 1: The suitability and characteristics of agile and plan-driven methods [11]

Characteristics	Agile	Plan-Driven
Primary Goals	Rapid Value Flexibility	Predictability Stability High assurance
Size	Small teams and projects	Large teams and projects
Environment	Chaotic and dynamic	Stable
Requirements	Informal and Unpredictable	Predictable evolution
Development	Simple design Short Increments	Complex design Long increments
Testing	Executable test cases	Documented test plans

2.1.6 Overview of Methodologies

The models discussed above can be divided into agile and plan-driven approaches. Newer plan-driven and agile methodologies have come to terms with the evolutionary nature of software and deal with this in different ways. Plan-driven methods have aimed to be flexible while still maintaining a high level of assurance, but are still more appropriate for large teams working on large projects. One of the key ways that agile methodologies differ is the use of cross-functional teams and short

timelines, which would not be feasible with large projects and teams, but is effective with small ones. It is important for an organisation to use the most suitable methodology by taking into account the factors discussed.

2.2 The Cost of Inadequate Software Testing

A study commissioned by the NIST (National Institute of Science and Technology) found that errors in software cost the United States economy $59.5 billion per year [12]. Of these costs, over 50% are borne by the end users and the remainder by the software developers. The report additionally states that over a third of these costs can be avoided by improving the testing infrastructure, which in turn will allow:

- More errors to be removed before the release of the software
- Error detection earlier in the development process
- More efficient error detection and resolution

The NIST study supports the notion that software defect removal costs increase with subsequent development stages. Therefore, the main advantage of improved testing is error detection and correction earlier in the development cycle.

There are many incidents that demonstrate this fact, such as the August 2003 blackout in the United States, which was caused partly due to a software error [13]. The total cost of the blackout was estimated to be up to $10 billion [14]. Although this is an extreme case, it demonstrates the impact of an inadequate testing process. Another notable case of software failure due to insufficient testing are the 1999 crashes of the unmanned NASA Mars Polar Lander and Mars Climate Orbiter. The cause of the crashes was a simple calculation error in which English units were not converted into SI units [15]. A NASA study admitted that software testing was not adequate [16]. The total cost of the projects was roughly $775 million, including the cost of both the lander and orbiter, spacecraft development, launch and mission operations [17][18]. If an organisation such as NASA, which follows such rigorous testing processes, is susceptible to errors due to flawed testing, then it is apparent that any organisation is likely to experience the same. Similar incidents that lead to revenue loss occur often in every industry, since software has become so ubiquitous [19].

The actual costs in some cases may possibly be higher than those reported, as it is often difficult to understand and hence quantify the cost of failure [20].

The increase in the cost of error correction as development progresses has been demonstrated by incidents such as those discussed, and observed by respondents of the NIST study, as well as several authors. Boehm suggested that the cost of error correction increases exponentially with each phase of development that the error remains uncorrected [21]. Later studies have also confirmed that a link exists between error resolution costs and the number of failures that subsequently occur [22].

The fact that so many studies, such as [23] by Westland, have been undertaken to quantify the cost of software failure and the cost failure prevention indicates that software errors are a major problem. This is because software failure costs have been shown to be potentially high. Additionally, unlike other engineering products, software does not provide a satisfactory level of quality assurance [24]. Due to these reasons, it is important to ensure that errors are detected and resolved as early as possible in the development process.

In order to reduce the probability of failures occurring, there has been much focus on process measurement and improvement, and there have been many methodologies developed to assess processes. The Capability Maturity Model Integration (CMMI) approach to process measurement and improvement is one of the most widely used methods of assessing and improving an organisation's performance in several areas. CMMI for development defines 22 process areas of an organisation that can be assessed [25]. Each process area belongs to a certain maturity level, which ranges from 1 to 5 [25]. An organisation can be appraised with a Standard CMMI Appraisal Method for Process Improvement (SCAMPI), and a maturity level is awarded to the organisation (or an organisational unit) based on the state of each of its process areas [25]. Each process area is composed of generic and specific goals and practices [25]. In order for a process area to be satisfied, all generic and specific goals and practices must be covered by an organisational process for that particular process area [25]. Generic goals and practices exist for all process areas, while

specific goals and practices only apply to individual process areas [25]. Maturity levels are related to the staged representation of CMMI, in which an organisation aims to improve several process areas together [25]. Should an organisation only aim to improve an individual process area, the continuous representation of CMMI is used [25]. In this case, a capability level, which ranges from 0 to 3, is awarded to each process area that an organisation selects for appraisal [25].

In order to apply the principles of CMMI, the Personal Software Process (PSP) was developed by Watts Humphrey for use by individual developers in order to improve their personal software development processes [26]. PSP aims to assist engineers in identifying areas in which improvement is needed. PSP is composed of methods, forms and scripts that guide the engineer in executing the development process. The PSP process entails the following steps: Planning, Design, Design Review, Code, Code Review, Compile, Test and Postmortem. Scripts are used to guide the engineer with each step. In addition to scripts, PSP logs and forms are used to provide templates for storing data. PSP standards are used to guide the actual work done, in terms of coding, LOC (Lines of Code) counting, and defects. PSP scripts are grouped into the following levels [27]:

- PSP0 and PSP0.1
- PSP1 and PSP1.1
- PSP2 and PSP2.1
- PSP3

Each level is associated with different tasks. As with CMMI, the levels can be seen as a progression of process maturity. PSP0 and PSP0.1 focus on planning, development and a postmortem [26]. PSP1 and PSP1.1 focus on estimation and planning [26]. PSP2 and PSP2.1 tasks include those of the previous level, and include design and code reviews, and design templates [26]. PSP3 includes the previous levels' tasks, as well as cyclic development, for larger scale projects. PSP3 has been superseded by the Team Software Process (TSP).

The Team Software Process was developed for large-scale projects undertaken by teams of software engineers practicing PSP [28]. As PSP is

aimed at guiding individuals towards improving their development processes, TSP provides a framework that guides teams towards achieving their goals. TSP has two main components, a team-building component and a team-working component [29]. A TSP launch is part of the team-building process. The launch consists of a series of nine meetings carried out over a four-day period [29]. The meetings are carried out for the purpose of developing goals, plans and strategies [29]. TSP consists of a measurement framework, which is based on the same measures within PSP [29]. TSP combines all the individual PSP data in order to manage the project [29]. PSP and TSP provide a structured, well-defined framework for the improvement of processes from a small individual scale, through to team level improvement.

The Test Maturity Model Integration (TMMI) framework is based on CMMI and has been developed specifically for test process improvement. As with CMMI, TMMI is composed of process areas and maturity levels [30]. Each maturity level has process areas associated with it, and like CMMI, each process area involves generic and specific goals and practices [30]. TMMI process areas are focused on different areas in testing. Unlike CMMI, TMMI currently only has a staged representation, which means that in order to achieve a rating of a particular maturity level, all specific and generic goals and practices of process areas up to and including that maturity level must be satisfied [30]. These frameworks for process improvement, as well as many others, rely on extensive documentation and place many constraints on the processes they examine in order to achieve good ratings [11].

2.3 The Need for Process Improvement in the Telecommunications Industry

It has been stated that software is no longer making its way into every aspect of society – it has already made its way there [19]. The same is true for general industry, and the telecommunications industry in particular. Software is no longer simply a tool that is used to execute a company's functions; it is the core of a company's functions, entrenched in every aspect of service and product provision.

Due to the role that software plays in the telecommunications industry, the process of developing and testing software must be sufficient to meet the needs of the industry. There are two main reasons that process improvement is necessary in telecommunications. Firstly, the rapid development of technology continues to promise the user new products and services. In addition to this, advancements usually bring about price reductions for older products and services. It is apparent that the operator that provides these advancements first and offers favourable pricing for users will have a competitive edge. Secondly, if these products and services are not of standard, the operator risks not only losing revenue in the short term, but in the long term as well due to customer dissatisfaction. Noting these two points, and the fact that a software development process is followed to create new products and services, it is evident that the more efficient the development process, the faster and more reliably the product can be launched.

Many countries have competitive telecommunications industries. South Africa in particular, has seen large changes to its industry in the past decade. There has been a significant decrease in the use of fixed line communications in favour of mobile offerings [31]. In addition to Vodacom and MTN, the launch of Cell C and, more recently, 8ta, has led to increased choices for the user, and hence increased competition in a country where choices were once severely limited. This has resulted in these operators competing aggressively by constantly offering various new promotions [31]. Hence, it is clear that the first operator to offer new products and services will have the greatest advantage. It therefore makes sense that the operator that has the most efficient and reliable development process will be able to meet the needs of the market most timeously.

A case study published in 2004 outlines how a medium-sized telecommunications company delivered a defective product late due to the lack of a rigorous testing program, resulting in customer dissatisfaction [32]. The product was a voicemail offering, which was delivered to the company's major customers. There were many faults with the product that were visible to the customer. Most of these faults were major. The installation of the product also had many issues, which had an

impact on service. In addition to this, the product was delivered late by several weeks. This resulted in significant customer dissatisfaction. Due to this, a metrics program was instituted by the research, development and quality groups of the company for the following version of the software due in a year. Metric profiles were used as a basis for the measurements taken. Four areas were focused on, namely: quality, functionality, time and cost. The Goal Question Metric (GQM) approach, which is discussed in a subsequent section, was followed in order to answer key organisational questions. Once the measures were defined, continuous metric reporting was performed as part of the program. The metric reports were given to management, and this allowed them to track the process in detail, as well as provide customers with information regarding the status of the project. The metrics reported included:

- Pre-test-stage defects
- Test-stage defects
- Tests scheduled, executed, passed, failed, and blocked
- Code length and changes
- Build, schedule, and cost reports

The metrics evolved and became broader over time. The metrics program resulted in substantial improvements. The newer version was delivered on time and had fewer defects than the previous version. The number of customer-visible major defects decreased by a factor of approximately ten and minor defects by a factor of five. Additionally, the newer version did not have as many installation issues as the previous one. The key reasons listed for the success of the metrics program were:

- A wide range of useful metrics provided rich information about the state of the process
- Any major issues were identified and corrected before escalating to critical levels
- Management could easily focus attention on aspects of the project that required it, based on information provided by the metrics

The case study shows that formal software testing and test measurement, emphasising early defect detection and correction, is an important cost and time-saving practice and crucial for customer satisfaction.

The points raised are also valid in the case of billing software for two main reasons. Firstly, billing directly affects revenue, and therefore billing errors are likely to be more costly. Secondly, all new offerings are linked to a billing system, and therefore billing is an important aspect of any new service. Software errors have the potential to be extremely costly, and in the case of billing, the cost of errors is likely to be higher since billing directly involves revenue. Additionally, the nature of the telecommunications industry demands that the process of developing and testing telecommunications billing software is as efficient and accurate as possible.

2.4 The Use of Metrics for Process Measurement

A metric is a quantitative method of measuring or predicting a specific attribute of a product or process [33][34]. Software metrics have been in use for over 40 years, beginning with the use of the lines of code metric in the sixties [33]. Metrics have since become a standardised method of measuring products and processes, and are useful for examining software testing processes [35]. Additionally, metrics are an essential part of any structured measurement program [34]. Despite this, metrics have not been implemented suitably, if at all, in many cases [32].

Respondents in the NIST study stated that their ability to obtain further testing resources was limited in part by a lack a historic tracking data [12]. Because the performance of the test process is inadequately tracked, estimating costs of fault detection and correction is difficult. Furthermore, accounting systems do not accommodate separate costs associated with error correction [23], which increases the difficulty of quantifying these costs. According to Capers Jones, of the companies that collect metrics, less than 10% include defect statistics or record errors [23]. Generally, there is a significant lack of process measurement in industry, and this is the initial step towards process improvement. It is therefore vital that an organisation have, at the very least, a rudimentary system in place to track the testing process and produce data on demand.

Pusala outlines several benefits of good metrics, which include [34]:

- Allowing the prediction of long term performance, and identifying high level goals

- Providing a basis for estimation, and enabling planning for better performance
- Offering a method of reporting the status of a process
- Identifying process areas which require attention, and areas which can be improved
- Providing information which enables faster and better decision making
- Allowing the entire process to be evaluated in terms of effectiveness and efficiency

Although the benefits of using metrics are clear, it is essential to focus on metrics that will be of use, and that identify key aspects of the process being measured. Therefore, identifying appropriate metrics is a crucial task when implementing a metrics program. One of the most reliable methods of identifying metrics and relating them to high level objectives is the Goal Question Metric (GQM) approach, which is discussed in subsequent sections.

2.5 The GQM Approach and its Extensions

The Goal Question Metric approach was developed by David Weiss under the supervision of Victor Basili in the 1980s [36]. Since then it has become one of the most widely used methods of defining metrics for evaluating processes and products [4]. The GQM method entails a goal-driven approach to measurement. This approach circumvents one of the main challenges of applying metrics, determining which are the most useful, by relating the measurements taken to specific goals associated with products or processes.

There are three levels to the GQM measurement model: a conceptual level, an operational level, and a quantitative level [37]. At the conceptual level is a goal, which is defined for a process, product or resource. A goal may be related to quality and/or productivity. This ensures that any measurements taken are focused on a specific purpose. At the operational level is a set of questions. These questions are aimed at defining the assessment of a particular goal. They examine the quality of the object being measured in terms of the defined goal, from a particular viewpoint.

Metrics reside at the quantitative level of the model. Each question has a set of data associated with it, i.e. specific metrics are selected to answer the questions. This allows the question to be answered quantitatively.

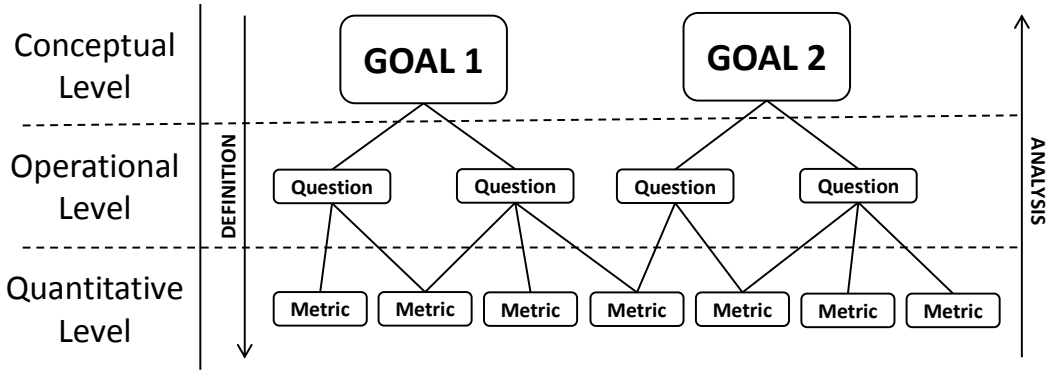

Figure 10: The GQM Model, adapted from [37]

This model is depicted in Figure 10, and it is hierarchical. In some cases, a single metric may be used to answer multiple questions. In this case, the metric is examined from different viewpoints.

Basili et al [37] describe a goal as being composed of three coordinates: an issue, an object and a viewpoint, in addition to a purpose. An issue may be user friendliness, efficiency, timeliness, etc. An object may be a process, product or resource. A viewpoint is the person or department whose point of view the goal is examined from, such as a project manager. The purpose of a goal could be improvement or measurement. An example of a goal would be to improve the user friendliness of a mobile word processor from the viewpoint of a user. In this example, the purpose is improvement, the product a mobile application and the viewpoint that of a user. Possible questions are, "How easy is the product to install?" and "How user friendly is the product?" Possible metrics may include ratings of the ease of installation, the number of issues encountered with the installation and a rating of the overall user friendliness.

Analysis is then performed in a bottom up manner, with the results of the metrics explicitly answering the associated questions. The answers then allow the quantification of the goal. At this point, any issues with the product or process examined are revealed, as well as the magnitude of the

issues. Once the issues are discovered, strategies for improvement to the product or process can be developed.

Several advantages of the GQM approach can be noted from the above, namely:

- Appropriate metrics are defined which are linked to well defined goals, preventing resources being expended on unnecessary metrics
- Since metrics are connected to goals, a context is provided for the analysis of the data collected
- The questions asked allow the goal to be evaluated clearly, directly and quantitatively
- Because goals are specific, the root causes of issues are revealed and hence resolved more easily

Since the GQM method is metrics based, it encompasses all the aforementioned advantages that metrics offer. Despite this approach initially being developed for software, it is a measurement model and therefore can be used to measure any product or process. Many companies have successfully used GQM to improve their products and processes [4].

Despite the GQM approach being used extensively since its inception in the 80s, it has been criticised for ineffectively linking technical goals to business goals in an organisation [38][39]. This issue was identified by Basili et al and the GQM$^+$Strategies extension was developed in order to relate higher-level business goals with the measurement goals defined in GQM [39].

GQM has also faced criticism for not integrating validation into its process, as well as not accommodating additional measurement based on data already collected [40][41]. Both these issues are addressed by the V-GQM method, which adds three additional steps to the original GQM process. These steps follow data collection, and are metric validation, question analysis and goal refinement [41]. The V-GQM method is shown in Figure 11. Metric validation involves examining the collected metrics and categorising them based on the information they provide. Each metric may be unavailable, extended (providing more information than required),

generalizable (being relevant to more than one question) and sufficient. Analysis is then performed based on the validation of the metrics, and the related questions are categorised in the same way as the metrics.

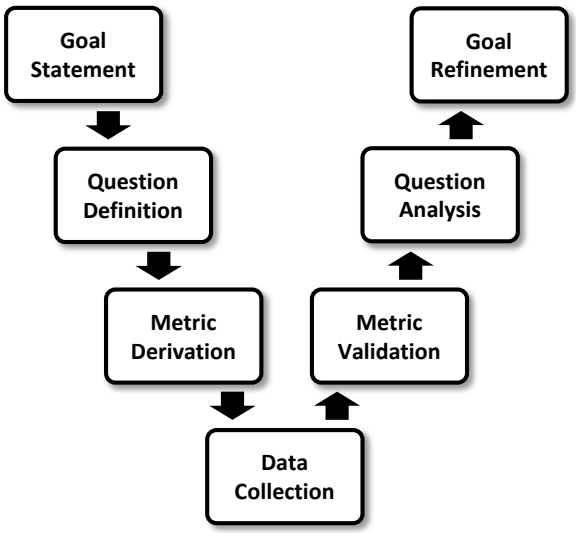

Figure 11: The V-GQM Process (adapted from [41])

The metrics' validation leads to questions being analysed and refined, with some questions being removed and new questions possibly included. Possible actions on how to resolve any issues that have arisen are also proposed at this point. The final step in V-GQM is goal refinement, which entails examining the results of the validation, and the possible actions to resolve issues. Decisions are made at this step on what actions to take, and goals are refined. This final step becomes the first step for the next GQM iteration. Hence, V-GQM is a cyclic GQM process, incorporating validation and refinement. The V-GQM extension therefore allows GQM to be implemented continuously and adaptably. The following section examines the use of GQM in industry more closely.

2.6 The Use of the GQM Method for Process Improvement

The Goal Question Metric approach has been used for process improvement extensively since being implemented at the NASA Software Engineering Laboratory in the 1980s [4][42]. Numerous studies have been carried out since then which have been based on using the GQM method for process measurement and improvement.

A GQM-based measurement program was instituted at an industrial company that manufactures and services systems for fuel stations [43]. The study focused on the reasons that developers were interrupted from work, and on how to decrease these interruptions. A subsequent study was performed on the Return on Investment (ROI) of a process improvement program [44]. The factors considered in order to calculate the ROI were the number of productive engineering hours and cost per engineer, and the number of hours saved due to improvement of the process. The resulting ROI was 2 and the program broke even. Despite this, several secondary benefits resulted from the program. These benefits included [44]:

- The project finished at least a week early due to the measurements
- The measurement analysis resulted in an update of documentation, which further prevented interruptions
- The awareness of quality and awareness of interruptions of the software team was raised
- Increased interruption awareness outside the department caused a decrease in interruptions in other projects in the department

By estimating the value of these indirect benefits, and considering them for the calculation, the ROI for the whole organisation was calculated to be 13. These studies show that GQM can be successfully used to provide a basis for process improvement, even for processes that are not specifically software oriented.

The GQM approach has also been used as a methodology to investigate the impact of other software engineering practices. A study published in 2009 examined the impact of the use of agile practices on projects by using the GQM approach [45]. The goal entailed examining whether the use of agile methods benefitted projects, and was defined by five questions. Eighteen projects were studied for this investigation. The projects were compared based on the development model employed: iterative, waterfall-like or agile. The study found that agile practices benefitted projects the most, and the use of GQM allowed useful recommendations to be made easily in terms of answers to the questions asked. The results of the study provided motivation for the use of agile methods, hence improving future development processes.

Another study made use of the GQM method to examine the defect management process in an organisation developing a telecommunications software product [46]. The study involved implementing a measurements program for four consecutive development projects. The results of the study were then compared to a previous project which did not make use a of a goal-driven measurement framework. Three main goals were defined for the study, namely [46]:

- Reduce the number of open defects
- Detect defects in earlier phases of development
- Increase the speed of verification activities

An additional fourth goal involved the defining of quality metrics to enable the personnel involved to be motivated to correct defects more effectively.

Three quality metrics were used to evaluate the defect management process [46]:

- Total number of unresolved defects at each week
- Percentage of defects found per phase – either during component testing phase (occurs earlier) or system testing phase (occurs later)
- Lifetime of major defects – either resolved in time or not resolved in time

All three metrics showed that the defect management process was improved for all the projects that instituted the measurement program. This study provides a strong indication that a goal-oriented measurements program can be used to improve defect management in general, and specifically in the case of telecommunications software.

It has been shown that process improvement begins with process measurement. If improvement is to occur successfully, measurement must be done accurately and in a structured manner. The abovementioned studies show that the goal question metric approach meets these requirements as a measurement model not only for telecommunications software development but also for any development process. This allows improvements to a process to be made easily and effectively.

The literature has shown that a research methodology that involves isolating specific goals for improvement is more likely to lead to those

goals being achieved. However, success in this regard can only be achieved if the goals defined point to specific measurements to be taken, and these measurements effectively feed back into the goals to be evaluated. The literature has also provided criteria for evaluating the sufficiency of the testing process, as well as possibilities of improving the process in accordance with the key research questions defined in section 1.3. The following chapter discusses the methodology and how the GQM approach described above is implemented in order to answer the key research questions.

3 Research Methodology

3.1 Introduction

This section discusses the methodology that was used to carry out the research and answer the research questions posed. The research questions asked of the investigation are:

1. Is the current testing process described in sections 1.2.2 and 1.2.3 adequate in terms of its:
    a. Efficiency
    b. Accuracy
2. If not, what improvements can be made?

Since the investigation relates to the testing process of telecommunications billing software, the research methodology involves measurement of the process in terms of metrics. The methodology entails determining which measurements should be taken by using the Goal Question Metric (GQM) approach and then collecting the required data. Since the metrics are taken in the context of high-level goals, the process as a whole can be evaluated by examining the results of each goal, allowing the identification of flaws and improvement of the process.

3.2 Research Methodology Overview

The research methodology is summarised in Figure 12. After proposing several metrics as a basis, the initial phase of the research involved studying the process documentation and the project documentation. The process documentation was examined in order to understand the process being measured. The documentation for each project was also studied in order to provide a context for the data collected.

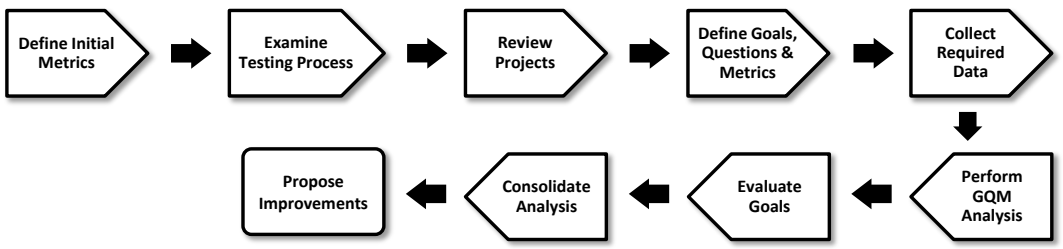

Figure 12: An overview of the research methodology used

Once the process was familiarised, the appropriate measurements to take were determined using the GQM approach. The approach began with specifying goals, questions to assess them and then defining metrics to answer the questions. Based on the metrics identified, the relevant data was collected. The data was then analysed in a bottom-up approach, applying the GQM method, i.e. using the answers of the questions to evaluate the goals. An overall critique of the process in each area was then carried out using the results obtained for each of the goals, in order to answer the research questions posed. Several suggestions for improvement in each area, as well as for the overall process have been recommended based on the results obtained.

3.3 Initial Testing Metrics

The initial set of metrics proposed is shown in Table 2, they were mainly adapted from [32] and [34]. Base metrics were selected to provide perspective on the system under investigation and on the results of the other metrics. The major metrics focus on the detection of defects at various stages in the testing process, since the main aim is to examine how well the process detects and corrects defects.

Table 2: Proposed initial test metrics for the investigation (adapted from [32] and [34])

Metric Type	Metric
Calculated Base Metrics	Ratio of tests executed to tests blocked
	Ratio of tests passed to tests failed
Defect Metrics	Defects by action taken
	Defects by detection phase
	Defects by Origin
	Defects found after launch per LOC
	Post-launch defects by severity
	Defect discovery time and cost
	Defect correction time and cost
	Defect removal Effectiveness by phase

The definition of initial metrics provided a starting point for the investigation, and provided a context when examining the process documentation and project documentation.

3.4 The Goal Question Metric Process

This section discusses how the GQM method discussed in the literature survey was undertaken, while the following chapter presents the actual results of applying GQM. Goals were defined, which lead to the relevant questions to ask of the process. These questions were then mapped to the final set of metrics used, in order to assess and achieve the outlined goals. A GQM model is used to define a goal, and its questions and metrics. A template of the model is shown in Table 3 and is based on [37].

Table 3: The GQM Model template (based on [37])

<table>
<tr><td>Goal</td><td>Purpose</td><td rowspan="4">The goal is described in these fields, and is phrased like a sentence.</td></tr>
<tr><td></td><td>Issue</td></tr>
<tr><td></td><td>Object</td></tr>
<tr><td></td><td>Viewpoint</td></tr>
<tr><td colspan="2">Question</td><td>Q1.1</td></tr>
<tr><td colspan="2">Metrics</td><td>M1.1.1
M1.1.2</td></tr>
<tr><td colspan="2">Question</td><td>Q1.2</td></tr>
<tr><td colspan="2">Metrics</td><td>M1.2.1
M1.2.2
M1.2.3</td></tr>
</table>

Since the research questions have been explicitly defined, it is useful to formulate goals based on the questions being asked of the research. The goals were therefore determined based on the research questions. This ensured that the results of the goals lead to the research questions being answered. The GQM process was then followed based on the goals derived from the research questions. After goals were defined, questions were drawn up which allowed each goal to be achieved. The questions were structured in a manner that allowed the goals to be evaluated directly from their answers. The questions mainly focused on determining three things:

1. Current performance in a particular area,
2. Whether this performance was sufficient,
3. And, with the cases of inadequate performance isolated, the causes of this poor performance were probed.

The first question aimed to assess performance in a particular area and establish a baseline. The second question then used the baseline

established in order to determine the performance in a particular area of the process, relative to this baseline. The second question also revealed where performance was poor. The third question then further examined the cases of poor performance to determine the causes. Each question produced a set of metrics that allowed the question to be answered directly. The data to be collected from NSN was then determined from the metrics. Data collection is discussed further in the next section. The above steps form part of the definition phase of the GQM method. The analysis phase of the research methodology is shown in Figure 13.

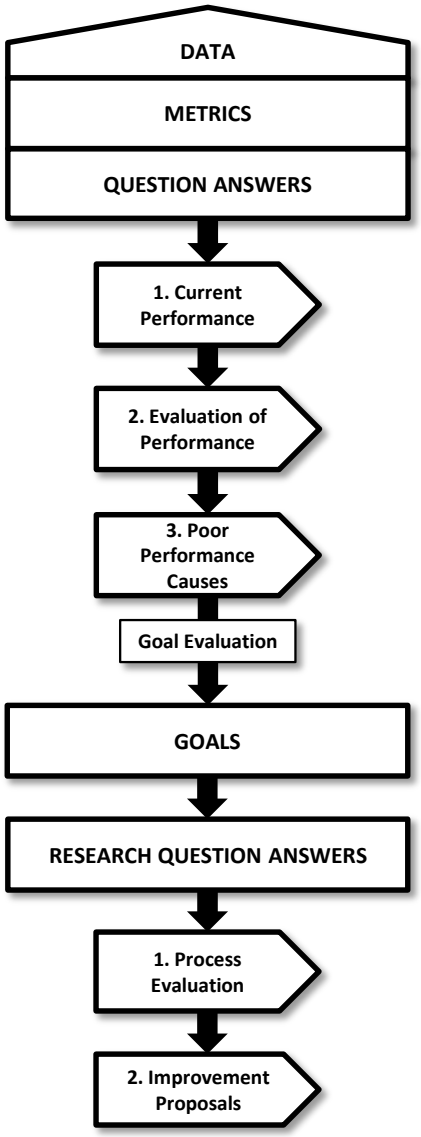

Figure 13: The expanded research model showing the analysis process of the research methodology

The analysis portion of the GQM process begins with calculating each metric from the data collected. Analysis was done focusing on a single goal and single question at a time. The metrics for a particular goal and particular question were calculated, and the metric results were then analysed in order to answer the question. Analysis then continued in this manner until all questions for a goal were answered. The answers of each question were then reviewed. As discussed, there are three main questions, and the answer of each question is dependent on the previous question's answer. The questions lead to the causes of poor performance in each area. The result of each goal was then determined based on the causes determined in the third question. Once each goal was evaluated, the results were analysed in order to answer the research questions.

3.5 Data Collection

Once the GQM metrics were defined, the data was collected. NSN makes use of a Test Management System (TMS) in order to track major errors during the testing process. The system tracks the errors in detail, from detection through all the steps taken to resolve the defect, until resolution. All parties involved in each step are logged, as well as the specific actions taken at each resolution step. All events are also dated so that resolution times can be calculated. In this way, the TMS logs every aspect of the testing process. Data was therefore collected from this system in the form of exported logs for each project and each error. Additional information such as project costs and project size was obtained from project managers and developers involved. In some cases, errors initially thought to be minor were not logged on the TMS. In these cases, testers and developers communicated via email to resolve the issues. These emails were obtained, analysed and formatted in a similar manner to TMS logs so that all the data was in a standard format, allowing metrics to be taken easily.

3.6 Improvement Proposals

An analysis of the results of each goal allowed the research questions to be answered. This formed the consolidated analysis of the results. The research questions aimed to determine whether performance is sufficient, and what improvements can be made. Since the goals were derived from the research questions, each goal led to the identification of areas of

improvement. As each goal was evaluated, improvement proposals for each area were then drawn up from each goal. In addition to specific proposals aimed at improving each area, proposals for improving the process as a whole were also determined.

The following chapter presents the results of the entire research process. This includes the derivation of the goals from the research questions, the definition of the measures used to assess performance, as well as a description of the questions asked for each goal, and the types of metrics used. The results of each metric, question and goal are then presented, with both quantitative and qualitative analyses. Possible improvement proposals are discussed based on the results of each goal. Improvement proposals are then discussed in further detail in a subsequent chapter.

4 Research Results

4.1 Introduction

The investigation mainly involved carrying out the Goal Question Metric process, and the results of each step of the process are presented in this chapter. The derivation of the goals from the research questions is discussed, followed by a discussion on the general set of questions that are asked. This is followed by a discussion on the metrics used to answer the questions. Since sets of questions are similar, the discussion focuses on the general characteristics of the questions, as is the case for the metrics.

4.2 GQM Goals

Before identifying goals explicitly, the research questions must be examined. Firstly, it must be noted that a testing process is being evaluated. The purpose of testing is to detect and resolve errors. Therefore, the evaluation of the process is actually the evaluation of defect detection and correction. Secondly, the process is being evaluated in terms of accuracy and efficiency. This leads to four specific aspects of the process that need to be examined:

- Defect detection accuracy
- Defect correction accuracy
- Defect detection efficiency
- Defect correction efficiency

This reasoning is depicted in Figure 14. The goals can therefore be based on these four aspects.

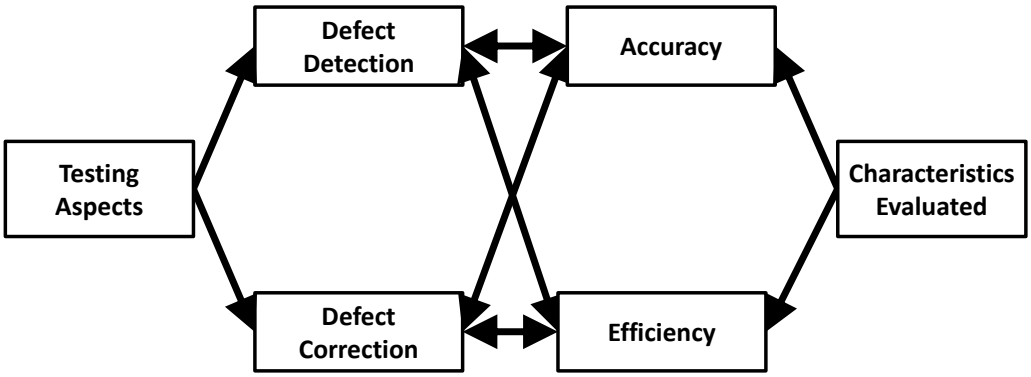

Figure 14: Linking the research questions to the testing process in order to derive goals

In order to ensure that the goals identified will answer the research questions, it is necessary to define accuracy and efficiency in the context of the investigation. Accuracy, in the context of defect detection and resolution, is defined by the following criteria:

- The ability to correctly classify defects
- The ability to implement an appropriate solution to resolve defects

In terms of accuracy with respect to classifying defects, a defect may be incorrectly identified or correctly identified, as well as falsely identified as a defect, when it is actually not a defect. In terms of accuracy with respect to defect resolution, an implemented solution may either be correct or incorrect.

The second aspect of the process being examined is its efficiency. Efficiency is usually defined by a ratio of an output to an input. In this case, efficiency is in terms of both effort (time) and cost, and the ratios to be noted are:

- The amount of effort expended on defect detection and resolution to the number of defects detected and resolved
- The amount of resources expended on defect detection and resolution to the number of defects detected and resolved
- The ratio of defects detected during the testing phase to all the defects detected, including after the testing phase (this ratio is known as defect detection efficiency)

The second research question, which deals with identifying improvements, follows from the first and is answered inherently as part of the GQM process. By bearing the abovementioned points in mind, the following goals were identified:

1. Improve the accuracy of defect detection
2. Improve the accuracy of defect correction
3. Improve the efficiency of defect detection
4. Improve the efficiency of defect correction

Since assessment is part of improvement, each improvement goal involves determining the current performance of the object with regard to the issue. By making the purpose of each goal improvement, both research questions can be answered.

4.3 GQM Questions

The GQM model for the investigation is presented from Table 4 to Table 7. Each goal, question and metric is numbered for easier reference. There are four goals, over 10 questions and almost 100 metrics. The goals are examined from a project management viewpoint, which is the most effective viewpoint from which to improve the process, since it is the lowest level that provides a balance between a high level view of the process, while still allowing the process to be examined in sufficient detail. This section focuses on the questions derived from each goal. The questions asked to assess and improve an aspect of the process are usually in groups of three: current performance is determined, its sufficiency is determined, and finally the causes of poor performance are determined. The tables of metrics are presented in the following pages.

Table 4: Goal 01: Improving defect detection accuracy

Goal 01:	Purpose	Improve
	Issue	The accuracy
	Object	Of defect detection
	Viewpoint	From a project management viewpoint
Question		Q1.1 What is the current defect detection accuracy?
Metrics		M1.1.1 Percentage of incorrectly classified defects per project M1.1.2 Average incorrect classifications M1.1.3 Standard deviation and Coefficient of Variation M1.1.4 Percentage of cases above average M1.1.5 Percentage of cases above 1 SD M1.1.6 Percentage false defects per project M1.1.7 Average false defects per project M1.1.8 Standard deviation and Coefficient of Variation M1.1.9 Percentage of cases above average M1.1.10 Percentage of cases above 1 SD
Question		Q1.2 Is the current level of detection accuracy

	sufficient?
Metrics	M1.2.1 Average delay due to inaccuracy M1.2.2 Standard deviation and Coefficient of Variation M1.2.3 Percentage of cases above average M1.2.4 Percentage of cases above 1 SD M1.2.5 Average cost due to delay M1.2.6 Standard deviation and Coefficient of Variation M1.2.7 Average % of project time spent on inaccuracy M1.2.8 Standard deviation and Coefficient of Variation M1.2.9 Percentage of cases above average M1.2.10 Percentage of cases above 1 SD
Question	Q1.3 What are the causes of detection inaccuracy?
Metrics	M1.3.1 Incorrect classification by cause frequency M1.3.2 Incorrect classification by cause time M1.3.3 Falsely classified defects by cause frequency M1.3.4 Falsely classified defects by cause time

Table 5: Goal 02: Improving defect correction accuracy

Goal:	Purpose	Improve
	Issue	The accuracy
	Object	Of defect correction
	Viewpoint	From a project management viewpoint
Question		Q2.1 What is the current defect correction accuracy?
Metrics		M2.1.1 Percentage of incorrect corrections per project M2.1.2 Average % of incorrect corrections per project

	M2.1.3 Standard deviation and Coefficient of Variation M2.1.4 Percentage of cases above average M2.1.5 Percentage of cases above 1 SD
Question	Q2.2 Is the current level of correction accuracy sufficient?
Metrics	M2.2.1 Average delay due to incorrect corrections M2.2.2 Standard Deviation and Coefficient of Variation M2.2.3 Percentage of cases above average M2.2.4 Percentage of cases above 1 SD M2.2.5 Average cost due to incorrect corrections M2.2.6 Standard Deviation and Coefficient of Variation M2.2.7 Average % of project time spent on incorrect corrections M2.2.8 Standard Deviation and Coefficient of Variation M2.2.9 Percentage of cases above average M2.2.10 Percentage of cases above 1 SD
Question	Q2.3 What are the causes of correction inaccuracy?
Metrics	M2.3.1 Incorrect corrections by cause frequency M2.3.2 Incorrect corrections by cause time

Table 6: Goal 03: Improving defect detection efficiency

Goal 03:	Purpose	Improve
	Issue	The efficiency
	Object	Of defect detection
	Viewpoint	From a project management viewpoint
Question		Q3.1 What is the current defect detection efficiency?

Metrics	M3.1.1 Defect detection efficiency per project M3.1.2 Average defect detection efficiency M3.1.3 Standard Deviation and Coefficient of Variation M3.1.4 Percentage of cases below average M3.1.5 Percentage of cases below 1 SD M3.1.6 Weighted defect detection efficiency per project M3.1.7 Average weighted defect detection efficiency M3.1.8 Standard Deviation and Coefficient of Variation M3.1.9 Percentage of cases below average M3.1.10 Percentage of cases below 1 SD
Question	Q3.2 Is the current defect detection efficiency sufficient?
Metrics	M3.2.1 Average time spent (per project) on undetected defects M3.2.2 Standard Deviation and Coefficient of Variation M3.2.3 Percentage of cases above average M3.2.4 Percentage of cases above 1 SD M3.2.5 Average cost (per project) due to undetected defects M3.2.6 Standard Deviation and Coefficient of Variation M3.2.7 Average percentage of project time spent on undetected defects M3.2.8 Standard Deviation and Coefficient of Variation M3.2.9 Percentage of cases above average M3.2.10 Percentage of cases above 1 SD
Question	Q3.3 What are the causes of defect detection inefficiency?
Metrics	M3.3.1 Undetected defects by cause frequency

	M3.3.2 Undetected defects by cause time M3.3.3 Undetected defects versus project size M3.3.4 Undetected defects versus number of test cases M3.3.5 Undetected defect correction time versus project size M3.3.6 Undetected defect correction time versus number of test cases

Table 7: Goal 04: Improving Defect Correction Efficiency

Goal 04:	Purpose	Improve
	Issue	The efficiency
	Object	Of defect correction
	Viewpoint	From a project management viewpoint
Question		Q4.1 What is the current speed of defect correction?
Metrics		M4.1.1 Average defect correction time M4.1.2 Standard Deviation and Coefficient of Variation M4.1.3 Percentage of cases above average correction time M4.1.4 Percentage of cases above 1 SD M4.1.5 Average percentage of test time taken by defect M4.1.6 Standard Deviation and Coefficient of Variation M4.1.7 Percentage of cases above average percentage of test time M4.1.8 Percentage of cases above 1 SD
Question		Q4.2 Is the current speed of defect correction sufficient?
Metrics		M4.2.1 Average above-average time spent M4.2.2 Standard Deviation and Coefficient of Variation

	M4.2.3 Percentage of cases above average M4.2.4 Percentage of cases above 1 SD M4.2.5 Average above average time spent as a percentage of test time M4.2.6 Standard Deviation and Coefficient of Variation M4.2.7 Percentage of cases above average M4.2.8 Percentage of cases above 1 SD
Question	Q4.3 What is the current cost of defect correction?
Metrics	M4.3.1 Average defect correction cost M4.3.2 Standard Deviation and Coefficient of Variation M4.3.3 Percentage of cases above average M4.3.4 Percentage of cases above 1 SD
Question	Q4.4 Is the current cost of defect correction sufficient?
Metrics	M4.4.1 Average above-average cost incurred M4.4.2 Standard Deviation and Coefficient of Variation M4.4.3 Percentage of cases above average M4.4.4 Percentage of cases above 1 SD
Question	Q4.5 What are the causes of defect correction inefficiency?
Metrics	M4.5.1 Delays in correction by cause frequency M4.5.2 Delays in correction by cause time M4.5.3 Correction time versus project size for delayed corrections M4.5.4 Correction time versus project size for all corrections

4.3.1 Determining Current Performance

These questions access the current performance of the process with respect to the aspect that is being improved. This is the first question asked in each goal. The metrics taken in this case are direct measures of the

performance concerning the aspect being improved. Although an assessment can be made on whether the performance is sufficient, an additional question is asked to ascertain this and provide a better context for assessment.

4.3.2 Determining if Current Performance is Sufficient

The second question usually asked directly assesses a given aspect of the process. This question makes use of more in-depth metrics to allow the performance to be evaluated thoroughly. The metrics of the previous question usually form the basis for the metrics that are used to answer these questions, and the data is usually the same for both sets of metrics.

4.3.3 Assessing the Causes of Poor Performance

The third question asked entails determining what the causes of poor performance are. The previous questions isolate the cases where performance is poor, and this question analyses those cases in order to determine the extent to which they affect overall performance. This analysis allows improvements to be proposed later in the GQM process.

4.4 GQM Metrics

The metrics that are taken measure the performance of the process with respect to specific aspects. A set of values of a given aspect is usually determined for projects or defects, with the average being taken in order to determine performance. In addition to the average, the standard deviation and coefficient of variation are also determined. Both the average and standard deviation are used as a baseline to highlight cases with poor performance. These measures are discussed further in this section.

4.4.1 The Standard Deviation and Coefficient of Variation

Averages are used as a base measure in most of the questions; however, an average is only meaningful if the standard deviation of the data set is also determined. The standard deviation measures the extent to which the values deviate from the average [47]. A high standard deviation indicates that most of the values are much smaller or larger than the average, and vice versa for a low standard deviation. Although the standard deviation on its own can be used to assess the extent of variation from the mean, the

coefficient of variation is a more direct measure of this. The coefficient of variation is given by [47]:

$$CV = \frac{SD}{Average} \times 100$$

It is useful to express the coefficient of variation as a percentage, and it is therefore the standard deviation as a percentage of the average. This contextualises the standard deviation with respect to the average. A coefficient of variation above 50% generally indicates a large amount of variation in the data set.

4.4.2 Examining Cases Above the Average and Above One Standard Deviation

In addition to determining the average, standard deviation and coefficient of variation, the number of cases that are above the average are also examined. This makes use of the average as a baseline. Furthermore, the number of values that are above one standard deviation above the average are also examined. These cases indicate poor performance, and if a large portion of the data is above the average as well as above one standard deviation, it indicates poor performance in general. Although it can be argued that a single standard deviation is a narrow margin of performance, it was found that there was a high amount of variability in the data, which significantly widened the standard deviation margin. In addition, while it may be appropriate to use wider margins when assessing other processes, this process and the needs of the industry required stricter standards. It is for these reasons that a single standard deviation was chosen.

4.4.3 Examining Causes by Frequency and Time

Questions that determine the causes of poor performance inspect each case of poor performance in a particular aspect of the process. The cause for that case of poor performance is then determined. Each cause is classified, and the number of cases that had that issue is determined. This is cause classification by frequency. Cases are also examined with respect to the time they took to resolve. This can be seen as a weighted version of the frequency examination, since more weight is given to cases that took up more time. Both these metrics provide different contexts for examining

issues. Although examining causes in terms of time is more accurate, using the frequency of the issues arising provides a useful comparison.

4.4.4 Cost Metrics

All cost metrics are proportional to time taken, and are based on the amount that the client is charged per day for that project. Even though examining the time spent on poor performance is sufficient to analyse its impact, cost metrics provide a real world perspective on the impact of poor process performance. These are the costs incurred by the client, and if the time was not spent, the cost of the remaining hours would still be incurred by the client, however the company would have saved on those costs in terms of resources. The costs therefore represent the total amount that the company could have saved on the resources expended on testing, such as staff.

4.5 GQM Results

The following sections present the results of each metric, question and goal. The goal is described in detail regarding the measures taken to assess it, and is summarised in terms of its questions. The results of each metric are discussed briefly before the answer to the associated question is determined based on the metric results. This represents a quantitative analysis. After each question is answered, the goal is discussed and assessed in terms of the answers. This is a qualitative analysis at the goal level. Preliminary improvement proposals are then discussed based on the goals' results. The improvement proposals for each goal are aggregated and discussed in detail in the subsequent chapter. Detailed results for each metric are presented in Appendix A.

4.6 Goal 01: Defect Detection Accuracy

The first goal was to improve defect detection accuracy. This goal comprised three questions:

1. What is the current defect detection accuracy?
2. Is the current defect detection accuracy sufficient?
3. What are the causes of defect detection inaccuracy?

In this context, accuracy is defined by correctly classifying a defect. This involves the correct analysis of the defect discovered and correctly

identifying it so that the correct steps can be taken to resolve it. Accuracy is also defined by correctly identifying a defect as a true defect, and not a false error. False errors may be identified for a number of reasons, such as a misunderstanding of the requirements by the tester, who then classifies correct functionality as defective. Falsely classified defects, like incorrectly classified defects, take up time and resources unnecessarily and are therefore indicators of poor process performance.

4.6.1 Question 1.1: Current Defect Detection Accuracy

This question assesses the current defect detection accuracy. The results of the metrics are presented in Table 8. Metrics 1.1.1 to 1.1.5 are related to incorrectly classified defects, while metrics 1.1.6 to 1.1.10 examine false defects. Metric 1.1.1 revealed that no projects classified defects incorrectly. This made the four metrics that follow redundant.

Table 8: Question 1.1 Metric Results

Metric Number	Metric Name	Result
M1.1.1	Percentage of incorrectly classified defects per project	0 for all projects
M1.1.2	Average incorrect classifications	0%
M1.1.3	Standard deviation and Coefficient of Variation	0, n/a
M1.1.4	Percentage of cases above average	0%
M1.1.5	Percentage of cases above 1 SD	0%
M1.1.6	Percentage false defects per project	Multiple Values
M1.1.7	Average false defects per project	29.28%
M1.1.8	Standard deviation and Coefficient of Variation	28.60%, 97.70%
M1.1.9	Percentage of cases above average	44.44%
M1.1.10	Percentage of cases above 1 SD	22.22%

Metrics 1.1.6 to 1.1.10 are related to false defects. It is seen that on average, almost 30% of the defects found are not actually defects. Since the standard deviation is almost equal to the mean however, there is much variation between projects. It is more useful to examine the cases that were above the average, and above one standard deviation, 44% and 22% respectively. The 44% implies that most of the cases were below the average, and an average of 30% is low. There are however a few cases that are of concern, since 22% of the cases were above one standard deviation, which is essentially twice the average, i.e. almost 60% of defects were false for those projects. Further examination of these cases was performed as part of question 1.3. Since the question regards assessing current defect

detection accuracy, the answer is numerical and is captured in the table above.

4.6.2 Question 1.2: Sufficiency of Detection Accuracy

This question involved determining if defect detection accuracy is sufficient. The results of the metrics taken are shown in Table 9.

Table 9: Question 1.2 Metric Results

Metric Number	Metric Name	Result
M1.2.1	Average delay due to inaccuracy	21.03 days
M1.2.2	Standard deviation and Coefficient of Variation	24.99 days, 118.86%
M1.2.3	Percentage of cases above average	33.33%
M1.2.4	Percentage of cases above 1 SD	19.05%
M1.2.5	Average cost due to delay	R57 128.66
M1.2.6	Standard deviation and Coefficient of Variation	R70 732.81, 123.81%
M1.2.7	Average % of project time spent on inaccuracy	21.79%
M1.2.8	Standard deviation and Coefficient of Variation	32.10%, 147.31%
M1.2.9	Percentage of cases above average	30%
M1.2.10	Percentage of cases above 1 SD	20%

The average amount of time that a false error takes to resolve is 21 days. Since the standard deviation and coefficient of variation are high, there is a large amount of variation and the average is not an accurate representation of the delays. However, further metrics provide an adequate representation. A third of the cases were above the average, and of those, almost two thirds were also above one standard deviation. Considering the fact that the average is high, this is definitely not sufficient. False errors should be identified much faster.

The cost of inaccuracy was also examined. The cost of each false error was calculated based on the amount of time (in days) taken to resolve the error. On average, a false error cost almost R60 000. Since the coefficient of variation is high, there is much variation about this average. The average does however put the effects of inaccuracy into perspective. The subsequent metrics provide a better assessment of the results.

In order to contextualise the time taken and costs incurred, M1.2.7 expresses the amount of time spent as a percentage of the total time allocated to the testing phase of the project. Like the case with the first metric, the amount of variation between projects does not make the average a useful quantity; however, an average of 21.79% of testing time is high. This insufficiency is further highlighted by the fact that 30% of the cases were above the average, and of those cases, two thirds were above one standard deviation.

The answer to the question of whether defect detection accuracy is sufficient is two parted. In terms of correctly classifying errors, accuracy is perfect. However, in terms of correctly determining if a defect is actually a defect, accuracy is poor. This insufficiency eclipses the accuracy of classification, and therefore overall defect detection accuracy is poor.

4.6.3 Question 1.3: Causes of Detection Inaccuracy

This question involves analysing the causes of inaccuracy. The metrics are listed in Table 10. Since there were no incorrect classifications, there is no data for metrics M1.3.1 and M1.3.2. Metrics M1.3.3 and M1.3.4 examine the causes of falsely classifying defects. The metrics focus on all the cases in which defects were falsely classified.

Table 10: Question 1.3 Metric List

Metric Number	Metric Name
M1.3.1	Incorrect classification by cause frequency
M1.3.2	Incorrect classification by cause time
M1.3.3	Falsely classified defects by cause frequency
M1.3.4	Falsely classified defects by cause time

Frequency refers to what percentage of cases the corresponding cause can be attributed to, while the time percentage is based on the percentage of time the errors took to resolve, for each specific cause. The cause categories include:

- Administration error: Administration errors include the following:
    - A fault being incorrectly logged for the wrong subsystem by a tester.

    - The client logging a false fault that they later resolve with the assistance of testers and developers.
    - Testers using the test management system to log an enquiry instead of an actual fault.
- Misunderstanding: Any form of misunderstanding, such as misunderstanding requirements and logging an error assuming that they were not met.
- Test Tool: Several software tools are used for testing, and are specific to the platform on which the software is developed. This testing software needs to be updated in line with the development software. Issues arise when testing is done with a tool (or tools) of an older version than what is required for the platform on which the solution was developed, resulting in a false error.
- Unrelated: The error is completely unrelated to the solution being developed, and originates within other systems that the developed solution makes use of.
- Other: These are any other causes that do not fall into the other categories.

The answer to this question is captured in Figure 15, which depicts the results of M1.3.3 and M1.3.4.

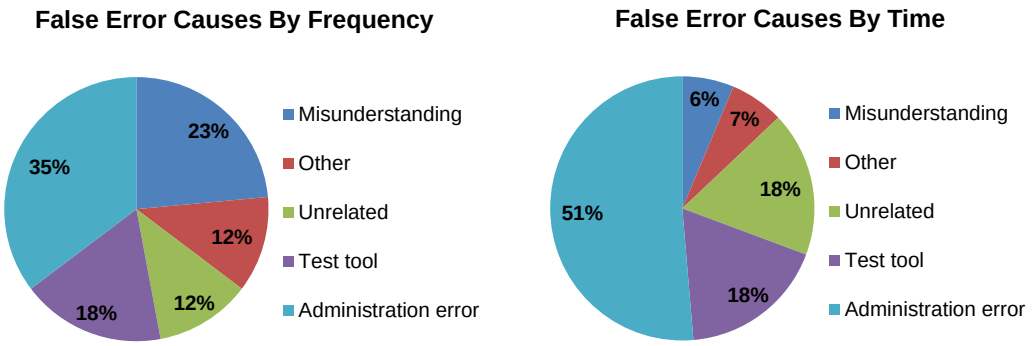

Figure 15: False error causes by time and frequency

Although it is more accurate to examine the causes in terms of time, examining the frequency of the causes provides an additional perspective on the analysis. In terms of both time and frequency, administration errors were the main cause of false errors. This is mainly because these errors are forwarded to several different departments and/or staff before being classified as incorrectly logged. Test tools being outdated and unrelated

errors being logged, took up roughly 18% of the total time taken to identify false errors. Misunderstandings and other causes did not take up much time, however misunderstandings do occur in 23% of the cases, making them important to mitigate.

4.6.4 Goal 01 Analysis

This goal entailed assessing and improving defect correction accuracy. Defect correction accuracy was found to be poor overall, with a significant impact on process performance. Key findings in this area were that:

- On average, 29% of the defects per project were false
- The average percentage of project time spent on inaccuracy was 21%
- The average cost per project due to inaccuracy was roughly R57 000

The causes of this were examined, and it was found that administration errors and test tools were the main causes, together making up almost 70% of the time spent on inaccuracy. Test tool issues in particular are trivial, and should not be permitted to have such an impact. The following is proposed to reduce defect detection inaccuracy:

- Better collaboration between developers and testers, as well as testers and clients will mitigate misunderstandings and administration errors:
    - Regular internal as well as external fault meetings are suggested.
    - Logging on the test management system should be done more regularly, so that all parties involved are aware of the issues, making the process faster.
- Test tools should be updated and provided more regularly

These proposals will be discussed in further detail in subsequent sections.

4.7 Goal 02: Defect Correction Accuracy

This goal entailed improving defect correction accuracy, and comprised three questions:

1. What is the current defect correction accuracy?
2. Is the current defect correction accuracy sufficient?
3. What are the causes of defect correction inaccuracy?

Correction accuracy is defined by implementing the correct rectification once a defect is classified. If, upon retesting, the defect still exists then the correction implemented is considered incorrect. Note that this process is followed many times during developer testing, mainly for minor errors. This goal focuses on major corrections that were insufficiently implemented during the testing phase.

4.7.1 Question 2.1: Current Correction Accuracy

Table 11 summarises the results of the metrics for this question. Metric 2.1.1 involved examining all the corrections made for all the errors found for each project. The percentage of those corrections that were incorrect was then taken for each project. All these percentages were then averaged for M2.1.2.

Table 11: Question 2.1 Metric Results

Metric Number	Metric Name	Result
M2.1.1	Percentage of incorrect corrections per project	Multiple Values
M2.1.2	Average % of incorrect corrections per project	3.70%
M2.1.3	Standard deviation and Coefficient of Variation	10.48%, 282.84%
M2.1.4	Percentage of cases above average	11.11%
M2.1.5	Percentage of cases above 1 SD	11.11%

As seen, on average only 3.70% of the corrections made were incorrect per project. Although the standard deviation and coefficient of variation are high, the low average suggests that in most of the cases, corrections were implemented sufficiently. Only a small percentage of cases were above this average and above one standard deviation, confirming that generally corrections are made properly. The following question assesses this result in further detail.

4.7.2 Question 2.2: Sufficiency of Correction Accuracy

The results of the metrics for this question are shown in Table 12. Metrics 2.2.1 and 2.2.2 show that the delays by incorrect corrections are extremely small. All the subsequent metrics show that incorrectly implemented corrections do not have a big impact, since corrections are implemented

appropriately almost all the time. It is clear that defect correction accuracy is sufficient.

Table 12: Question 2.2 Metric Results

Metric Number	Metric Name	Result
M2.2.1	Average delay due to incorrect corrections	0.094 days
M2.2.2	Standard Deviation and Coefficient of Variation	0 days, 0%
M2.2.3	Percentage of cases above average	25.50%
M2.2.4	Percentage of cases above 1 SD	0%
M2.2.5	Average cost due to incorrect corrections	R286.11
M2.2.6	Standard Deviation and Coefficient of Variation	R0, 0%
M2.2.7	Average % of project time spent on incorrect corrections	0.10%
M2.2.8	Standard Deviation and Coefficient of Variation	0%,0%
M2.2.9	Percentage of cases above average	0%
M2.2.10	Percentage of cases above 1 SD	0%

4.7.3 Question 2.3: Causes of Correction Inaccuracy

This question examines the causes of incorrectly implemented error corrections. The metrics for this question were not useful since there was only a single cause of incorrect corrections, and this was incorrect error analysis by the developer. Analysis of this cause shows that developer understanding of the error is essential to correct it effectively. Previous questions' answers indicate that these cases are isolated, and developers implement required corrections sufficiently.

4.7.4 Goal 02 Analysis

The first two questions show that defect correction accuracy is not an issue. An analysis of the small number of cases where the issue existed shows that these cases are isolated and generally, developers accurately implement the required corrections. A possible improvement may be to have testers clearly propose a solution for the error if required. Any proposal made for this issue will have an unnecessary overheard since it will only be useful in a small percentage of cases, and therefore improvement proposals are not essential for this goal.

4.8 Goal 03: Defect Detection Efficiency

Goal 03 involved the assessment and improvement of defect detection efficiency. Since defect detection efficiency has the biggest impact on process performance, this is a key goal in the study. Defect detection efficiency (or effectiveness) refers to how well a particular phase of development detects defects, and it is characterised by the ratio of defects found in a particular phase to the total number of defects found, including in subsequent phases. This is a well-documented metric and has been defined by several authors [48]. The questions asked for this goal are:

1. What is the current defect detection efficiency?
2. Is the current defect detection efficiency sufficient?
3. What are the causes of defect detection inefficiency?

As with other goals, the questions for this goal aim to assess performance, and investigate the causes of poor performance.

4.8.1 Question 3.1: Current Defect Detection Efficiency

The metric results for this question are summarised in Table 13. On average, defect detection efficiency is under 40%. This means that on average, 6 in 10 errors are found after the testing phase.

Table 13: Question 3.1 Metric Results

Metric Number	Metric Name	Result
M3.1.1	Defect detection efficiency per project	Multiple Values
M3.1.2	Average defect detection efficiency	36.67%
M3.1.3	Standard Deviation and Coefficient of Variation	36.21%, 98.75%
M3.1.4	Percentage of cases below average	44.44%
M3.1.5	Percentage of cases below 1 SD	44.444%
M3.1.6	Weighted defect detection efficiency per project	Multiple Values
M3.1.7	Average weighted defect detection efficiency	36.33%
M3.1.8	Standard Deviation and Coefficient of Variation	37.08%, 102.07%
M3.1.9	Percentage of cases below average	55.56%
M3.1.10	Percentage of cases below 1 SD	44.44%

Although the standard deviation and coefficient of variation are high, even with significant variation about the average, performance is not

satisfactory since the average itself is low. Metrics M3.1.4 and M3.1.5 show that all the cases that are below the average are also below one standard deviation, indicating 0% efficiency in 44% of the cases. To provide further perspective on these metrics, a weighted version of efficiency was used. This version assigns a weighting to each defect detected based on the amount of time that a defect took to correct. The weighting uses resolution time as a measure of the severity of the defect. In this way, the efficiency calculation takes the severity of the defects into account, and if defects detected after the test phase are more severe, this will negatively affect the calculated efficiency – the opposite applies for defects detected during the testing phase. The results for the metrics related to time-weighted efficiency are very similar for the non-weighted efficiency, but indicate that performance is slightly worse, with over half the cases with below-average efficiency.

4.8.2 Question 3.2: Sufficiency of Detection Efficiency

This question assess whether the results of the previous question were sufficient. Although the results of the previous question can be used to answer these questions, further metrics allow for a more thorough assessment. These results are shown in Table 14. The average amount of time spent on undetected or post-test phase defects is 19 days. The high standard deviation and coefficient of variation suggest that efficiency is either extremely high or extremely low. Although only a third of the cases took above the average time, two thirds of these cases were also above a standard deviation, which is extremely high. Cost metrics were also taken to put the results into perspective. On average, just over R51 000 per project must be paid by the client due to undetected defects. The cost was calculated based on the time taken to resolve the defects. Once again, the high values of the standard deviation and coefficient of variation show that costs are either very high or very low. The high average however, indicates that efficiency is not sufficient. In order to obtain further perspective, the amount of time spent on defects detected after the testing phase was calculated as a percentage of the total time allocated to the testing phase. It was found that on average, almost 70% of test phase time was spent per project resolving defects after the testing phase. This result and the results preceding it indicate that defect detection efficiency is far from sufficient.

Table 14: Question 3.2 Metric Results

Metric Number	Metric Name	Result
M3.2.1	Average time spent (per project) on undetected defects	19.40 days
M3.2.2	Standard Deviation and Coefficient of Variation	18.91 days, 97.48%
M3.2.3	Percentage of cases above average	33.33%
M3.2.4	Percentage of cases above 1 SD	22.22%
M3.2.5	Average cost (per project) due to undetected defects	R51 419.51
M3.2.6	Standard Deviation and Coefficient of Variation	R48 661.62, 94.64%
M3.2.7	Average percentage of project time spent on undetected defects	68.80%
M3.2.8	Standard Deviation and Coefficient of Variation	80.31%, 116.74%
M3.2.9	Percentage of cases above average	33.33
M3.2.10	Percentage of cases above 1 SD	22.22

4.8.3 Question 3.3: Causes of Detection Inefficiency

The reasons why defects were only detected after the testing phase ended were investigated for each post-test phase defect. As with previous questions, the metrics examined causes by how often they occurred (frequency) and by the amount of time they took to resolve. In addition to this, the effect of project size and the effect of the number of test cases on the number of post-test phase defects were investigated. The time spent correcting post-test phase defects was also investigated in this respect. The metrics for this question are listed in Table 15.

Table 15: Question 3.3 Metric List

Metric Number	Metric Name
M3.3.1	Undetected defects by cause frequency
M3.3.2	Undetected defects by cause time
M3.3.3	Undetected defects versus project size
M3.3.4	Undetected defects versus number of test cases
M3.3.5	Undetected defect correction time versus project size
M3.3.6	Undetected defect correction time versus number of test cases

The following causes of defects going undetected were identified:

- No test case coverage or insufficient test case coverage: In these cases, the defect went undetected because the solution was not covered by a test case or the test case did not test for the specific defect.
- Misunderstanding Requirements: These cases occur when certain functionality is not implemented because the requirements of the solution are misinterpreted. The customer reports these defects as requirements not being met or not being met sufficiently.
- Other: These are cases that do not fall into the other categories.

The results of metrics 3.3.1 and 3.3.2 are shown in Figure 16. The figure shows that in terms of frequency and time taken, insufficient testing is the main cause of inefficiency, with misunderstandings being the second biggest cause.

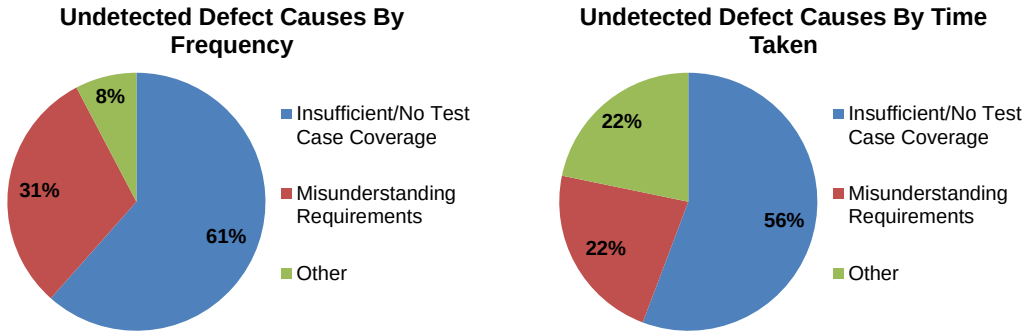

Figure 16: Causes of undetected defects by frequency and time

The other causes were mainly errors that were closed because of insufficient information provided by the client regarding the error. The reason a large portion of time was taken was that time was spent awaiting a response from the client. An examination of the number of defects found after testing against the size of the project and the number of test cases is shown in Figure 17. Project size is measured by the number of modules that were modified or added to the project. Although this is not a very accurate measure of the size, it provides a sufficient basis for comparison. The scatter plot on the left shows the effects of project size on post-test phase defects. Since both small and large projects are associated with a small number of defects, there is no noticeable relationship between the two. The scatter plot on the right compares post-test defects to the

number of test cases. Fewer test cases are linked to more undetected defects as well as fewer undetected defects, while more test cases are associated with fewer defects. There are not enough points to confirm that a trend exists, however it should be noted that fewer post-test defects are associated with more (above 30) test cases.

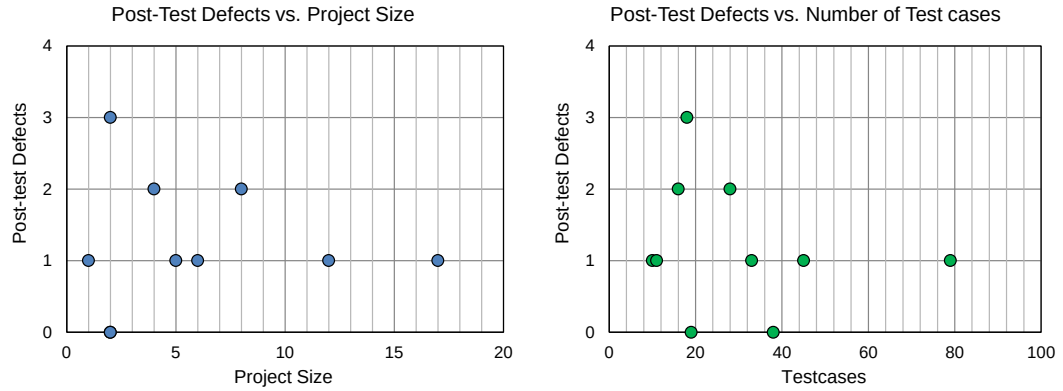

Figure 17: The effect of project size and the number of test cases on post-test phase defects

Figure 18 shows the results of metrics 3.3.5 and 3.3.6, which examine the relationships between the time spent on correcting undetected defects and the project size and number of test cases. The scatter plot on the left shows that undetected defects of smaller projects are linked to long and short correction times and that larger project' defects are linked to shorter correction times. There does not seem to be any link between the project size and the time spent correcting post-test defects.

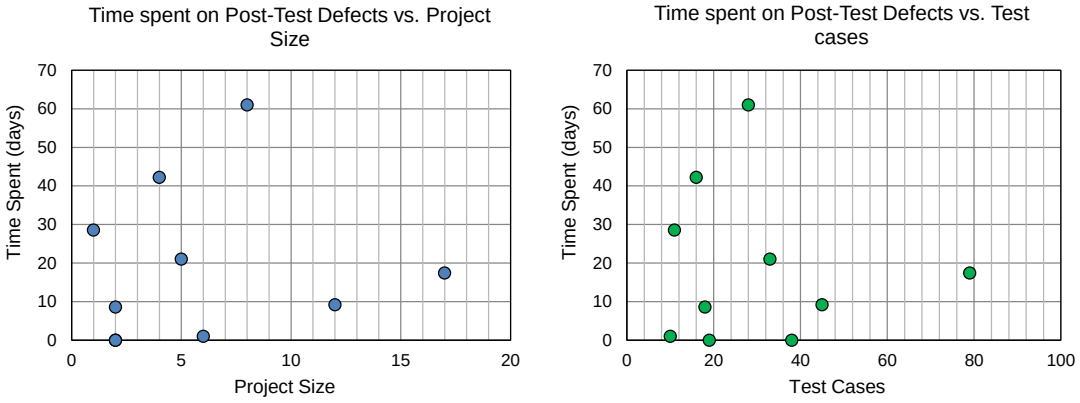

Figure 18: The effect of project size and the number of test cases on the time spent on post-test phase defects

The scatter plot on the right examines the relationship between the number of test cases and the time spent correcting undetected defects.

Although a small number of test cases is associated with less time spent, it is also linked to more time spent. With more test cases (over 30), it is seen that less time is spent, indicating less severe errors.

The scatter plots are a useful method of examining if relationships exist between any quantities. Project size seems to be independent of undetected defects. However, in both the scatter plots that investigate the effects of the number of test cases, it is seen that more test cases are associated with fewer and less severe errors. There are not enough points to identify a trend; however, there are two separate cases indicating that increased test cases lead to fewer major errors.

4.8.4 Goal 03 Analysis

The questions asked have revealed that defect detection efficiency is not sufficient, and that inefficiency is costly. The main findings for this goal were:

- The average defect detection efficiency was found to be 37%
- 19 days were spent on average per project correcting undetected errors
- 68% of project time on average was spent on correcting undetected errors

The causes of inefficiency have been isolated, and are mainly that testing is not thorough enough, and that requirements are not understood sufficiently. The lack of thorough testing is confirmed by an examination of the number of test cases that has shown that more test cases are related to fewer and less severe errors. Based on these results, preliminary proposals include:

- Improving the testing process by:
    - Including developers in the test design process
    - Increasing collaboration between testers
    - Increase the amount of time spent planning test cases
- Ensure that requirements are well documented and understood by both developers and testers.
- Implement better methods of requirements elicitation from the client.

These proposals will be discussed in further detail in a subsequent section.

4.9 Goal 04: Defect Correction Efficiency

Goal 04 focused on assessing and improving defect correction efficiency. In this case, efficiency refers to the rate at which defects are corrected after being detected. This goal comprised the following questions:

1. What is the current speed of defect correction?
2. Is the current speed of defect correction sufficient?
3. What is the current cost of defect correction?
4. Is the current cost of defect correction sufficient?
5. What are the causes of defect correction inefficiency?

For this goal, costs associated with correcting defects were not examined with regard to the total test cost. This is because the test cost is fixed and based on the time allocated to testing. The number of defects however is not fixed, and depends not on the testing process (which is the focus of the study) but on the development process (which is not the focus of the study). It follows from this that as the number of defects increase, the cost per defect decreases, implying better cost efficiency, but not drawing attention to the fact that the development process injects many defects. In the opposite case, with fewer defects and a fixed cost, the cost per defect is large, indicating poor cost efficiency, while overlooking better quality code.

4.9.1 Question 4.1: Current Defect Correction Speed

The metric results of this question are shown in Table 16. On average, it takes 12 days to correct a defect. Since the standard deviation and coefficient of variation are high, there is much variation about the average, indicating that the times are either very short or very long. Although fewer than 50% of cases are above the average time, 8% of cases are above a standard deviation, which is high (24 days). These results are put into context by expressing the time taken as a percentage of the time allocated to the testing phase. On average, a defect takes 40% of testing time to correct. As with the other metrics, there is much variation with each defect. Only 38% of the cases are above the average, however almost half these cases are also above a standard deviation, which is over 90% of test time. Those cases need to be investigated further.

Table 16: Question 4.1 Metric Results

Metric Number	Metric Name	Result
M4.1.1	Average defect correction time	12.64 days
M4.1.2	Standard Deviation and Coefficient of Variation	11.40 days, 90.19%
M4.1.3	Percentage of cases above average correction time	45.83%
M4.1.4	Percentage of cases above 1 SD	8.33%
M4.1.5	Average percentage of test time taken by defect	39.95%
M4.1.6	Standard Deviation and Coefficient of Variation	52.80%, 132.16%
M4.1.7	Percentage of cases above average percentage of test time	37.50%
M4.1.8	Percentage of cases above 1 SD	16.00%

4.9.2 Question 4.2: Sufficiency of Correction Speed

This question aims to analyse the sufficiency of defect correction. The results are presented in Table 17. The metrics use the average correction time as a basis to assess the speed of defect correction. Every defect that took above the average correction time was examined. The average correction time was subtracted from the correction time for each case, to examine how much above the average these cases were. Metric 4.2.1 is the average of the differences for each defect. Therefore, the defects that took above-average time to correct took on average 9.93 days more than the average correction time, which was 12.64 days.

Table 17: Question 4.2 Metric Results

Metric Number	Metric Name	Result
M4.2.1	Average above-average time spent	9.93 days
M4.2.2	Standard Deviation and Coefficient of Variation	9.18 days, 92.36%
M4.2.3	Percentage of cases above average	18.18%
M4.2.4	Percentage of cases above 1 SD	0%
M4.2.5	Average above average time spent as a percentage of test time	34.95%
M4.2.6	Standard Deviation and Coefficient of Variation	44.57%, 127.52%
M4.2.7	Percentage of cases above average	27.27%
M4.2.8	Percentage of cases above 1 SD	18.18%

There is significant variation with each correction time as with the other metrics; however, M4.2.3 shows that very few cases were above the average correction time. In addition to this, no cases were above a standard deviation. This means that there are very few cases with extremely slow correction times. The results are further analysed by comparing them to the time allocated to testing for each project. The time that each case was above the average was expressed as a percentage of the total testing time for that project. The average of each of these values was taken, and this is metric M4.2.5. It is seen that the above-average time taken is on average 35% of project time. Even though the values vary significantly from this average, this is relatively high. Further metrics show that almost 30% of these cases are above the average, and more than half of those cases are above one standard deviation.

These results indicate that although correction speed is marginally acceptable, there are too many cases with unsatisfactory speeds that decrease the overall performance of error correction. If for any reason defects increase during development, the error correction process will be under significant strain.

4.9.3 Question 4.3: Costs of Defect Correction

This question focuses on the costs associated with correcting defects. This is an auxiliary question that puts the results of the previous questions into a real world perspective. The results are shown in Table 18. Since the costs are proportional to the time spent, the results are similar to those of question 4.1. As with many other questions, there is much variation between each case.

Table 18: Question 4.3 Metric Results

Metric Number	Metric Name	Result
M4.3.1	Average defect correction cost	R31 948.60
M4.3.2	Standard Deviation and Coefficient of Variation	R29 619.91, 92.71%
M4.3.3	Percentage of cases above average	45.83%
M4.3.4	Percentage of cases above 1 SD	12.5%

4.9.4 Question 4.4: Sufficiency of Correction Costs

This question is also an auxiliary question that converts the results of other questions into more tangible units. The results of this question are shown in Table 19.

Table 19: Question 4.4 Metric Results

Metric Number	Metric Name	Result
M4.4.1	Average above-average cost incurred	R25 810.24
M4.4.2	Standard Deviation and Coefficient of Variation	R23 647.29, 91.62%
M4.4.3	Percentage of cases above average	27.27%
M4.4.4	Percentage of cases above 1 SD	18.18%

The average above-average cost per defect, R25 810.24, can be seen as the amount that can be saved per defect if correction times are reduced to the average correction time. Although the cost varies per defect, a large number of cases are above the average and a standard deviation, indicating room for improvement. As with the answer of question 4.2, there are many cases of unsatisfactory performance which need to be investigated so that correction times and hence correction costs are reduced.

4.9.5 Question 4.5: Causes of Correction Inefficiency

This question investigates the causes of delays in correcting defects. The metrics for this question are listed in Table 20.

Table 20: Question 4.5 Metric List

Metric Number	Metric Name
M4.5.1	Delays in correction by cause frequency
M4.5.2	Delays in correction by cause time
M4.5.3	Correction time versus project size for delayed corrections
M4.5.4	Correction time versus project size for all corrections

The cases with above-average correction times are focused on. As with similar questions, causes are examined by the number of occurrences and the total time spent on resolution. In addition, any relationship between

the project size and correction time is also examined, for both delayed corrections and all corrections. Four causes of delays were identified:

- Complex dependencies: The correction was minor but dependencies were complex, since core modules had to be modified, which took time
- Major correction: In these cases modules needed to be corrected or updated, and this took time due to the extent of the correction
- Other concurrent errors: There were other concurrent errors being corrected, which slowed down the process
- Lack of information: There was a lack of information provided by client, or the information was delayed

Figure 19 shows the results of metrics 4.5.1 and 4.5.2. In both cases, major corrections and complex dependencies are the main causes of delays. A lack of information can be expected to cause delays since most of the time is spent awaiting information. The error correction process is capable of handling multiple concurrent errors, since it is the smallest cause of delays.

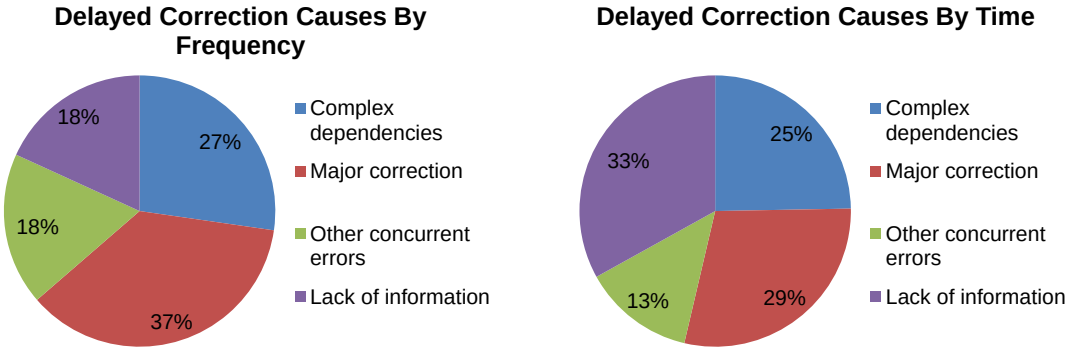

Figure 19: Delayed correction causes by frequency and time spent

The results of metrics 4.5.3 and 4.5.4 are shown in Figure 20. The scatter plots show that short (under 10 days) and long (over 15 days) correction times are associated with small, medium and large projects. Correction time is therefore independent of the size of the project.

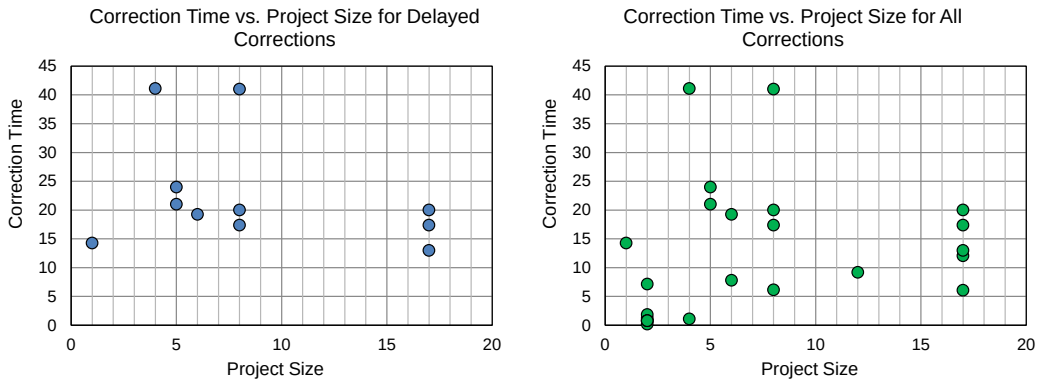

Figure 20: Correction time versus project size for delayed corrections and all corrections

4.9.6 Goal 04 Analysis

It has been determined that defect correction efficiency is not sufficient overall, due to many cases of correction delays. The causes of this were identified so that improvement proposals could be drawn up. The effects of project size on correction time were examined, and they were found to be independent of each other. Key findings for this goal include:

- The average defect correction time is 12 days
- The average percentage of test time taken by a defect is 40%
- Of the above-average cases, 10 days were spent, on average, above the average error correction time
- On average, the additional cost incurred in delayed cases was R25 810.24

Based on the results found, the following improvements can be considered:

- Analysis needs to be performed at the beginning of the project on which modules will be required, and possible pretesting should be done so that they are updated for the current project.
- At the onset of the project, an impact analysis needs to be performed to assess which changes need to be made so that core modules can be modified as part of development instead of as a corrective process.
- A system should be in place to close faults faster if information is not provided within a specific time or after a certain number of requests.
- Concurrent errors can only be mitigated at prior stages, since they involve error injection. Most suggestions will not be feasible in this

case, since a small percentage of the corrections are delayed due to concurrent errors and the overhead will be significant. A more feasible solution would be to increase emphasis on development phase testing.

These improvement proposals, as well as those for other goals are discussed in more detail in the next chapter.

4.10 Summary of Results

Testing involves defect detection and correction, and the investigation aimed to assess the accuracy and efficiency of these testing aspects. Four goals were defined from these aspects and the aims of the research. The first goal involved assessing defect detection accuracy. Accuracy was defined to be correctly classifying an error. No errors were found to be incorrectly classified, however on average almost 30% of errors per project were found to be false errors – errors that were shown to not actually be errors after investigation. Defect detection accuracy was therefore deemed insufficient. The second goal entailed evaluating defect correction accuracy. Correction accuracy was defined by the ability to implement a correct solution to an error once it had been classified. Correction accuracy was found to be adequate, since only a single case existed where an error was incorrectly corrected, and this was due to incorrect analysis of the error.

The last two goals focused on efficiency. Goal 03 involved assessing defect detection efficiency. Detection efficiency is characterised by how well the process detects defects, and examines the ratio of defects detected in-phase, to defects detected after a particular phase. The main result in this area was an efficiency of just under 40%, which means that 6 in 10 defects are detected after the testing phase is complete. Defect detection efficiency was therefore found to be insufficient. The final goal examined defect correction efficiency. Correction efficiency is related to the speed at which defects are corrected once discovered. A baseline was determined based on the average correction speed, and cases with long correction times were isolated. It was found that 45% of the cases were above the average correction time, and 8% were above a single standard deviation. Correction efficiency was determined to be satisfactory for most cases; however, there were too many cases with large delays which impacts performance in this

area. The following chapter examines these results and further discusses the proposals for improvement that were presented in each section of this chapter. Additionally, possibilities for improving the process as a whole are discussed.

5 Process Improvement Proposals

5.1 Introduction

This section presents several proposals for improving the testing process in terms of accuracy and efficiency. The proposals are based on the findings of the GQM process. The study involves four goals, and there is a set of improvement suggestions presented for each goal, with the exception of the second goal, involving the improvement of defect correction accuracy. This is because the defect correction accuracy of the testing process was found to be adequate.

5.2 Improving Defect Detection Accuracy

This section presents proposals for improving defect detection accuracy, based on the findings of the study. Defect detection accuracy has been assessed based on the following aspects:

1. Correctly classifying a defect that has been detected
2. Correctly classifying a defect as an actual defect

It was found that all defects detected were classified correctly, and therefore the first aspect requires no improvement. The second aspect of defect detection accuracy however, has significant room for improvement. The main causes of falsely identifying defects were found to be:

1. Administration Errors
2. Test Tool issues
3. Unrelated Errors
4. Misunderstandings

Although the research literature is scarce regarding falsely classified defects, some simple steps can be taken to reduce their effect on the performance of the testing process. Based on the findings of the study, the following is proposed to reduce defect detection inaccuracy:

- Better collaboration between developers and testers, as well as testers and clients
- Test tools should be updated and provided more regularly

It can be argued that although false defects are a waste of resources, they are a sign of rigorous testing, and that it is preferable to be more cautious by identifying these possible defects, than to risk them being actual defects going undetected by testing. However, the causes of false defects identified

in this study show that falsely classifying defects is not usually indicative of rigorous testing, and hence this aspect of testing should be improved.

5.2.1 Improving Collaboration between Developers, Testers, and Clients

It was found that administration-based errors were a major cause of incorrectly logging defects. Misunderstandings and unrelated errors also contributed to false defects. Both these issues can be dealt with by better testing collaboration between developers, testers and clients. Regular logging on the test management system should be encouraged, so that everyone involved is aware of issues as they occur, making the resolution process faster. Another way that better collaboration can be achieved is through regular fault meetings. These meetings should take place as long as there are unresolved faults for the project, and should include developers and testers. If the fault relates to an issue discovered by the client, they should also be present. These meetings will allow:

- Actual errors to be classified due to developer and project management input
- Administrative errors to be resolved due to both input from developers and project management
- Any misunderstandings to be cleared through discussion between all parties

This will in effect reduce the amount of time and effort spent on these issues, since they are dealt with immediately and directly between all the relevant participants. This proposal is well aligned to the Scrum development approach, which entails several types of meetings [49]. Daily Scrum meetings involve the following [49]:

- They last only 15 minutes
- Each participant reports on what they have achieved in the past day, what they plan on achieving by the end of the current day, and any issues they have had with executing their tasks
- After the meeting, the issues raised are attended to by the relevant participants, and facilitated by the Scrum Master – the facilitator and manager of the Scrum process

There are several documented cases highlighting the overall benefits of using the Scrum development process [50][51][52], and it is suitable for the

projects of the company since they already use short development cycles as part of a lightweight process which focuses on minor additions of functionality to existing core customer products. It is therefore proposed that the Scrum methodology be implemented not only to improve defect detection accuracy, but the process as a whole. This proposal is discussed in detail in a subsequent section.

5.2.2 Regularly Updating Test Tools

Section 4.6.3 discusses the finding that out of date test tools are one of the main causes of false errors being detected. This is because they falsely report errors on newer functionality that is not recognised. This issue is trivial and should not be the source of inefficiencies. The following is recommended in this regard:

- Test tools should be checked for compatibility with current solutions before each project begins, so that they can be prepared for the project's testing phase
- Regular updates should be scheduled so that tools are not out of date

Since this issue is straightforward, the steps to be taken to resolve it are basic.

5.3 Improving Defect Detection Efficiency

Defect detection efficiency is one of the most important aspects of testing, since the consequences of poor performance in this area are greater than other areas. It was found that defect detection efficiency was not satisfactory, with efficiency below 40%. The following causes of poor efficiency were identified:

1. Inadequate test or test case coverage
2. Misunderstanding requirements

The following recommendations are made to improve this aspect:

- Improve the quality of test cases and tests by:
    - Including developers in the test design process
    - Increasing collaboration between testers
    - Increase the amount of time spent planning test cases

- Ensure that requirements are well documented and understood by both developers and testers.
- Implement better methods of requirements elicitation from the client.
- Reduce the amount of development errors before the test phase by increasing the amount of testing done by developers themselves, possibly through test-driven development (TDD)

These recommendations are discussed in further detail below.

5.3.1 Improving Test Cases and Tests

It was found that testing was not rigorous enough for a sufficient level of defect removal effectiveness. One of the ways to improve the quality of testing is to include developers in the test design process. This will allow a broader perspective when designing test cases and improve test coverage. Developers will also be able to highlight any areas of complexity that should be focused on during testing to ensure that testing is done thoroughly. The Scrum approach has been proposed to increase to amount of collaboration between all parties involved in development and testing. Promoting increased collaboration between testers through Scrum and its daily meetings, will result in higher quality test cases and tests. The financial implications of increasing the role of developers in testing should also be considered, since a trade-off exists where increased developer involvement will cost more staff hours yet possibly save on hours in the long term due to significantly fewer errors requiring resolution.

In addition to the above recommendations, the amount of time spent planning test cases should be increased. Better and more effective planning will result in a more accurate analysis of the system being tested. This will ensure that the coverage of test cases and tests is sufficient.

5.3.2 Improving Requirements Elicitation

Several proposals for improving the process of eliciting requirements from the client are presented. It should be noted that the investigation is focused on the testing phase of the process, however since inadequately captured requirements affected testing, suggestions are offered to improve upon this process. Based on the project documentation examined, and the

results of the investigation, it was found that generally, requirements are captured adequately. The issues that arose from a lack of requirements elicitation were technical in nature, such as specific features not being documented properly. Recommendations that do not affect the entire requirements elicitation process have therefore been made, instead of techniques that demand an overhaul of the entire process. These recommendations will specifically address the minor issues found, and leave the requirements gathering process as it is, since it is satisfactory overall.

Although there are many methods of improving requirements elicitation, there is a method that also improves the testing process. This involves creating a test specification instead of requirements. It is proposed that in this case, both be done such that the test specification complements the existing requirements specification. A test specification entails an exhaustive list of test cases based on the features required of the product. At this level of detail, any minor misunderstandings can be rectified by the client. This method also allows developers to more easily implement the solution, and works well with test-driven development. This method also allows the test planning process to be done more easily, since a thorough list of test cases already exists, which may require minor refinements at most.

5.3.3 Reducing Errors before the Testing Phase

Although the study is focused on the testing phase, the possibility of implementing Test-driven development (TDD) at the coding phase is discussed. TDD involves creating tests before writing code, which reduces the number of errors made while coding, since code is written specifically to pass the tests created [53]. This proposal is well suited to the suggestion to increase the amount of testing done by developers. The current process entails a small amount of developer testing before the solution is released to the testers. Although reducing the amount of errors entering the testing phase will not increase defect detection efficiency of the testing phase, it will increase the defect detection efficiency of the process as a whole. Development testing is discussed in detail the section on overall process improvement proposals.

5.4 Improving Defect Correction Efficiency

Defect correction efficiency involves the rate at which defects are corrected. It was found that the main causes of delays in correcting defects were:

1. Major Corrections
2. Complex Dependencies
3. Lack of Information
4. Other Concurrent Errors

In order to resolve the first two issues, preliminary analysis on which modules will be required should be done. This should involve possible development testing to ensure that issues are resolved before the testing phase. An impact analysis also needs to be performed to assess which changes need to be made so that core modules can be modified as part of development instead of as a corrective process, since core modules do not receive adequate attention from developers, resulting in major corrections during testing. A lack of error information provided by the client was found to be a minor issue and it is suggested that faults should be closed faster if information is not provided within a specific time or after a certain number of requests. The issue of concurrent errors can only be resolved by improving prior phases, since they involve error injection. Overall process improvement suggestions are discussed in a further section, and development testing in particular will mitigate this issue.

5.4.1 Performing Preliminary Development Analyses

Two of the main causes of delays in resolving errors had to deal with major corrections to modules of software and small but complex corrections due to dependencies between modules. These issues can be avoided by analysing the requirements of the project before development begins. Based on the requirements, the core modules to be modified for the specified project should be determined and modified or corrected during the development process. A second analysis should be carried out on the dependencies between the modules required for the project. If possible, these dependencies should be reduced so that error resolution is more easily performed. All the modules that will be modified for the project should be tested during development so that any issues that arise can be resolved immediately.

5.5 Overall Process Improvement Proposals

This section discusses proposals to improve the process as a whole. They encompass most of the specific recommendations made in the previous sections, but since they affect the process as a whole, the impact and overhead must be considered.

5.5.1 The Scrum Approach

The current process followed by NSN can be viewed as iterative and incremental, but lacking in agility. The Scrum approach is an agile development methodology that was developed in 1995 by Ken Schwaber and Jeff Sutherland [49]. It focuses on small, incremental additions of functionality to a software product, encouraging faster development and flexibility. This approach is suited to the existing processes of the company since:

- The current process uses short project times
- Small additions of functionality are developed for existing customer products
- The teams that develop the solution are small, allowing:
    - Communication to take place effectively
    - Client involvement and interaction to occur more easily
- A close relationship already exists between the company and its clients

The Scrum process is shown in Figure 21. Scrum is defined by a collection of roles, events, artefacts and rules. There are three main Scrum roles, a Scrum Master, Product Owner, and a Development Team. The Scrum master facilitates the Scrum process, and can be thought of as a project manager, ensuring that the process is followed properly, and assisting the Team with any issues or obstructions associated with their tasks. The Product Owner serves as an interface between the Team and the client and the organisation. The Product Owner is responsible for managing the Product Backlog, which is a list of project requirements and tasks to be performed. The Product Owner may also assist with the tasks in the Product Backlog. The Development Team is responsible for creating a functional increment of the product. The team as a whole is responsible for the product, and they are cross-functional, including testers, designers,

developers, etc. Sub-teams of specific roles such as developers and testers are discouraged. Small team sizes are encouraged to ensure effective communication and easier team management. In order for the organisation to implement Scrum, the impacts (organisational, procedural, etc.) of combining the testing and development need to be assessed.

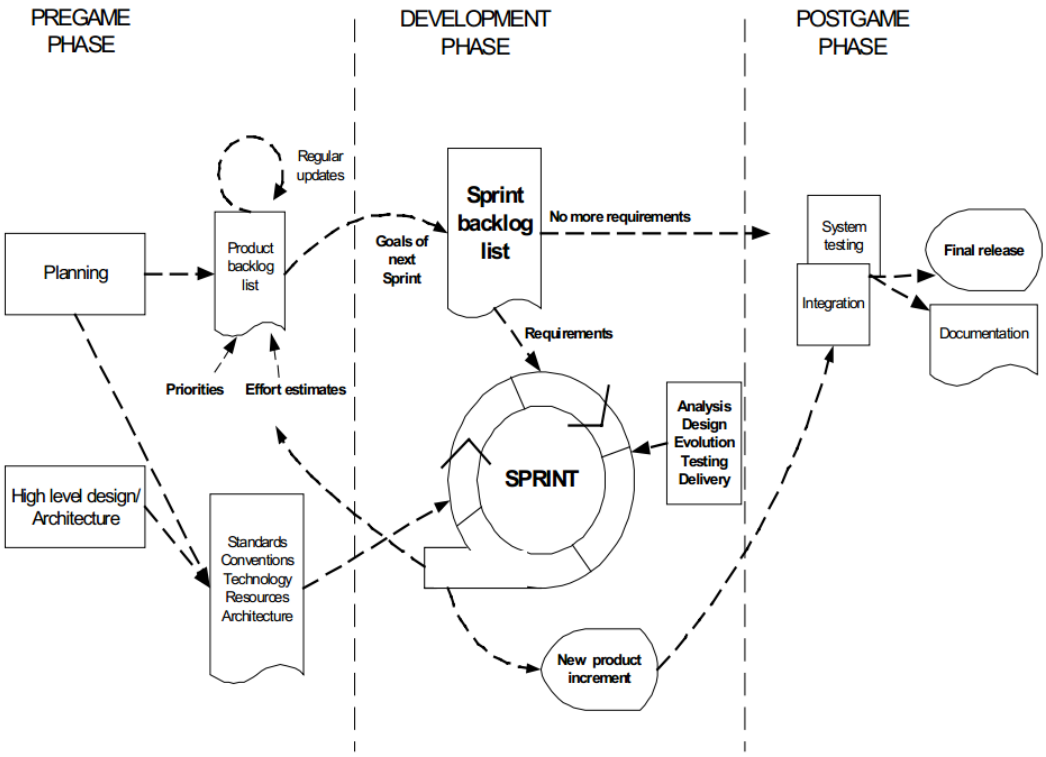

Figure 21: The Scrum development Process [54]

The development of an increment of software in Scrum is referred to as a Sprint, which is meant to be short, and not recommended to last longer than a month. The team implements the requirements as outlined in the Product Backlog, and is capable of adapting to requirements as they change. Since a Sprint is short, the cost of not meeting requirements is minimised.

There are several types of meetings in Scrum. One of the major meetings is a Sprint Planning Meeting, which is used to plan the next Sprint. The first part of the meeting involves the Product Owner and the Team, who determine what needs to be done in terms of the Product Backlog. The second part involves the Team only, and entails planning how the

development will be executed. Daily Scrum meetings also take place, in which each team member reports on what has been done in the past day, and what is planned for the current day, as well as any obstacles faced. This allows issues to be dealt with immediately, and allows progress to be monitored continuously, increasing the chances of meeting the goals of the project. After a Sprint, a Sprint Review meeting takes place. This meeting is used to evaluate the tasks completed and the tasks still to be done. The successes and the failures of the Sprint, as well as any issues faced, are reported by the Team. The Product Backlog is updated in the process. After the Sprint Review, a Sprint Retrospective meeting takes place, involving the Team members, who examine the previous Sprint and determine what improvements can be made.

The implementation of Scrum will lead to increased collaboration between all parties involved in development, reducing the impact of the issues raised in previous sections. As mentioned, Scrum is suitable for the organisation for several reasons, mainly due to the current process being compatible with the Scrum process. However, a significant change that must be made to the existing organisation is the amalgamation of staff into a single Development Team responsible for a solution. The feasibility of this will determine the feasibility of Scrum as a whole.

Numerous agile methodologies can be implemented for process improvement. The most prevalent methodologies include Scrum, Extreme Programming (XP), Crystal Clear, Lean Software Development, Dynamic Systems Development Method (DSDM) and Agile Unified Process. These methodologies differ in terms of characteristics such as:

- Team size
- Project size
- Organisational Level
- Project Duration

In addition to these characteristics, an approach that is well aligned to the specific proposals made will be most suitable, the proposals that need to be taken into account are:

- Increased collaboration between all parties involved in the project
- Regular meetings to discuss project progress and issues

By taking the characteristics mentioned, and the specific process improvement proposals into account, it was found that Scrum is the most appropriate agile methodology. All methodologies are compatible with the company's team and project sizes, and project durations. The methodologies are applicable to certain organisational levels – either the development or testing level, or the project management level. This study focused on process improvement from a project management perspective, as indicated in the GQM goals, hence some methodologies, although useful at lower levels, were not considered for the improvement proposals of this study. As discussed, Scrum is an agile approach that is implemented at the project management level, which aims to improve the development process as a whole. Most other methodologies, such as XP, are focused on a lower level, namely development and testing.

In addition to this, few methodologies prescribe regular meetings with all parties, which are focused on each individual's progress and hindrances – this is in line with one of the major improvement proposals. Overall, considering the nature of the organisation, and the suggestions for improvement, Scrum has been determined to be the most suitable methodology to be implemented.

5.5.2 Increasing Development Testing

A recent study was carried out by Forrester Consulting into the effects of including testing within the development phase of software [55]. The study surveyed IT leaders from over 200 companies in North America and Europe. Almost all those surveyed plan on increasing the amount of development testing and believe that development testing leads to better software quality, security and most importantly, cost reduction. This point has also been raised in the literature survey.

Development testing does not necessarily need to be performed by developers alone, and the test team may also assist in detecting defects in the development phase. In this way, development testing lends itself well to agile development and Scrum. This is because a Scrum Team is cross-functional and solutions are developed in short, small increments, which incorporate a testing process. Development testing also works well with test-driven development (TDD). Using Scrum and TDD removes the need

for a formal testing process that is heavy on resources and only suitable for large projects. As with the implementation of Scrum, the organisational impacts of development testing should be considered, especially in terms of staff, since the role of dedicated testers changes and possibly diminishes as testing moves earlier in the life cycle. If the task of testing is too great to keep the development cycle short, it may be feasible to focus on low level testing during development, such as unit testing and performance testing, while a separate test team focuses on higher level testing such as integration testing in a separate testing phase.

6 Conclusions

It is evident that software process improvement, including testing, is necessary in the telecommunications sector. Studies have shown that errors have the potential to be extremely costly. The investigation successfully assessed the testing process of a telecommunications billing solutions provider using the Goal Question Metric approach. Flaws and inadequacies were revealed leading to suitable proposals for improvement being made.

The use of a research model in which GQM goals were derived from the research questions ensured that the evaluation of the goals allowed the research questions to be answered. This yielded four GQM goals. After defining these goals, the GQM methodology was followed as documented in the literature, resulting in 14 questions and almost 100 metrics. This is adequate considering an average of 4 questions and 25 metrics per goal. The questions were structured to reveal three things in each area: the current performance, the sufficiency of the current performance, and what the causes of poor performance were. The current performance was used to determine a baseline for comparison. The sufficiency of the performance was then assessed with this baseline, and in the process, cases of poor performance were highlighted. The third question then closely examined these cases of poor performance in order to determine their causes. This systematic approach ensured that major causes of poor performance were isolated so that strategies for addressing them directly could be drawn up.

The goals were related to four aspects of the testing process: defect detection accuracy, defect correction accuracy, defect detection efficiency, and defect correction efficiency. Defect detection accuracy was found to be insufficient due to the amount of false errors classified. The main causes of falsely identifying defects were found to be administration errors and test tool issues. In order to resolve administration errors improved collaboration between developers, testers, and clients was proposed. This proposal entails regular meetings, which focus on any issues found during testing. The Scrum approach, which is a type of agile development method, is useful in this regard, since it encourages close collaboration within cross-functional teams with regular meetings that address any issues that arise. Out of date test tools falsely report errors on newer functionality that is not recognised. It was proposed that these tools be

updated more regularly through scheduled updates so that they are prepared for use when the testing phase begins. The second goal involved defect correction accuracy, which is related to correctly implementing an appropriate solution once a defect is correctly classified. This aspect was found to be sufficient, since almost all the corrections implemented resolved the errors found in each case.

Defect detection efficiency is one of the most important aspects of testing, since any defects that go undetected lead to increased correction costs and customer dissatisfaction. Efficiency was found to be inadequate, since on average 60% of errors are only detected after the testing phase, indicating an efficiency of 40%. The main causes of inefficiency were isolated to be a lack of test or test case coverage and a misunderstanding of requirements in some cases. The main recommendations to resolve this were to include developers in the test design process, increasing collaboration between testers as well as increasing the amount of time spent planning test cases. These recommendations are also well aligned to the Scrum approach. Increasing the amount of development testing would also decrease the burden of error detection on the testing phase. Test-driven development was recommended in this regard, as it has proven successful in many cases. An improved requirements elicitation technique was suggested which focuses on writing test cases as opposed to requirements, as this would ensure that requirements are clear to all parties, and later assist both developers and testers in the test design process. The last goal examined defect correction efficiency, which examines the speed at which defects are corrected. This area was found to be slightly inadequate, since there were many cases where the implementation of corrections was slow. The main causes of this were that major corrections needed to be made in these cases, as well as the complexity of the dependencies within the software. It was suggested that prior to development, analysis should be performed on the software requirements so that all core modules can be updated and tested as part of the development phase, which will prevent issues arising during the testing phase.

In order to improve the process as a whole, it was suggested that several proposals be amalgamated so that the multiple issues found are mitigated with a single framework. This framework should be based on the Scrum

methodology, which will ensure collaboration between development and testing teams by merging them into a single cross-functional team. The development process should be test driven, so that errors are minimized significantly and there is less of a strain on the testing process.

Further investigation can be achieved by implementing V-GQM. V-GQM is an extension on GQM, which integrates feedback into the GQM process, and makes it iterative as opposed to linear. It would be useful to examine the process in accordance with V-GQM should the improvement proposals be implemented. Alternatively, the next iteration of GQM can be implemented at a similar company that makes use of a more agile process, allowing for comparison in terms of the same criteria that evaluated the testing process in the study.

In summary, the investigation has demonstrated three things:

- Process improvement is important in telecommunications
- Measurements should be connected to high-level goals
- The process examined was inadequate in most areas

Process improvement is important in telecommunications, especially in the testing of billing software. The studies discussed show that the lack of a quality testing process can be very costly in terms of revenue, resources, and most importantly, customer satisfaction. Secondly, measurements need to be focused and linked to high-level goals. In this regard, the GQM approach has been shown to be adept at measuring, analysing and improving processes. The proposals made directly address the issues arising out of inaccuracy and inefficiency in the testing process.

Finally, the study found that the process was inadequate in most areas. The cases of poor performance were isolated and investigated. Specific proposals were made to resolve the issues found. Generally, the process as a whole can be improved by implementing Scrum and development testing with TDD. The organisational impacts of the implementation should however be considered.

If the results of this study are seen as an indication of the current state of software testing in the telecommunications industry, then it has revealed several trends. It has shown that the industry requires large-scale, long

term additions of products and services less often, and it is actually small-scale, innovative services that are released competitively that determine the success of companies. Because of this, traditional software life cycles and development processes are inadequate, since they are tailored to large-scale solutions. The industry requires a more agile approach. Although certain aspects of the process studied are agile in nature, the process still lacks the ability to be adequately accurate and efficient. A life cycle that segregates development and testing in terms of both activities and staff is no longer satisfactory, since standalone testing processes have become deprecated and outdated when used for small projects. Therefore, for smaller projects that focus on incremental increases of functionality, the amalgamation of the development and testing processes within an agile framework results in a better quality process. The investigation undertaken has contributed useful knowledge to a field in which similar research is scarce. The study has provided evidence of the unsuitability of traditional lifecycle-based processes for small telecommunications software projects. In addition, the research serves as motivation for the use of agile methodologies in the development and testing of telecommunications software.

References

[1] Hunter J M., Thiebaud M E. *Telecommunications Billing Systems: Implementing and Upgrading for Profitability*. McGraw-Hill TELECOM Professional, New York, first edition, 2003, pp. 3-14.

[2] Alekseev S, Tollkühn P, Dai ZR, Hoffmann A, Rennoch A, Schieferdecker I, *Testing Customizable Software for Telecommunication Services*, Proceedings of the 11th International Conference on Intelligence in Next Generation Networks, October 2007.

[3] Nokia Siemens Networks, Proprietary internal company documentation.

[4] Shull F, Seaman C, Zelkowltz M. *Victor R. Basili's Contributions to Software Quality*, Software, IEEE , Vol 23, January-February 2006, pp. 16-18.

[5] Pressman R S., *Software Engineering: A Practitioner's Approach*, McGraw-Hill, New York, fifth edition, 2001.

[6] Royce, W.W., *Managing the Development of Large Software Systems: Concepts and Techniques*, Proceedings of the 9th International Conference on Software Engineering, August 1970.

[7] Boehm B., Spiral model, 1988. http://en.wikipedia.org/wiki/File:Spiral_model_(Boehm,_1988).svg, Last accessed 20 January 2012.

[8] Kroll P., Kruchten P. *The Rational Unified Process Made Easy: A Practitioner's Guide to Rational Unified Process*, Addison-Wesley Professional, 2003.

[9] Iterative Development Illustration, http://en.wikipedia.org/wiki/File:Development-iterative.gif, Last accessed 20 January 2012.

[10] Beck, K, et al. *Principles Behind the Agile Manifesto*, http://agilemanifesto.org/principles.html, Last accessed 21 January 2012.

[11] Boehm B., Turner, R. *Balancing Agility and Discipline: A Guide for the Perplexed*, Boston, Addison-Wesley. 2004.

[12] Tassey G. *The Economic Impacts of Inadequate Infrastructure for Software Testing*, Planning Report 02-3, prepared by RTI for the National Institute of Standards and Technology (NIST), May 2002.

[13] US–Canada Power System Outage Task Force, *Final Report on the August 14, 2003 Blackout in the United States and Canada: Causes and Recommendations*, US Dept. of Energy, April 2004.

[14] ICF Consulting, *The Economic Cost of the Blackout: An issue paper on the Northeastern Blackout, August14, 2003*, 2003.

[15] Euler E E, Jolly S D, Curtis H H. *The Failures of the Mars Climate Orbiter and Mars Polar Lander: A Perspective from the People Involved*, Proceedings of Guidance and Control, American Astronautical Society, paper AAS 01-074, Colorado, 2001.

[16] Mars Program Independent Assessment Team Summary Report, March 14 2000.
http://sunnyday.mit.edu/accidents/mpiat˙summary.pdf, Last accessed 15 October 2011.

[17] NASA, *Mars Climate Orbiter Fact Sheet*.
http://mars.jpl.nasa.gov/msp98/orbiter/fact.html, Last accessed 15 October 2011.

[18] NASA, *Mars Polar Lander Fact Sheet*.
http://mars.jpl.nasa.gov/msp98/lander/fact.html, Last accessed 15 October 2011.

[19] Zhivich, M., Cunningham, R.K, *The Real Cost of Software Errors*, Security & Privacy, IEEE , Vol 7, March-April 2009, pp. 87-90.

[20] Grottke M, Graf C, *Modelling and Predicting Software Failure Costs*, 33rd Annual IEEE International Computer Software and Applications Conference, IEEE Computer Society, Los Alamitos, 2009, pp. 180–189.

[21] Boehm B W. *Software Engineering Economics*. Prentice-Hall, New Jersey, first edition, 1981.

[22] Royce, W., 1993. *Why software costs so much: how to get people and technology to work together*. IEEE Software Vol 10, May/June 1993, pp. 90-91.

[23] Westland, J C., *The Cost of Errors in Software Development: Evidence From Industry,* Journal of Systems and Software, Volume 62, May 2002, pp. 1-9.

[24] Berry, D. M, *Appliances and Software: The Importance of the Buyer's Warranty and the Developer's Liability in Promoting the use of Systematic Quality Assurance and Formal methods*, Workshop on Modelling Software System Structures in a Fastly Moving Scenario, Santa Margherita Ligure, Italy, 2000.

[25] CMMI Product Team, *CMMI for Development version 1.3 – Improving processes for developing better products and services*, SEI Report CMU/SEI-2010-TR-033, November 2010, http://www.sei.cmu.edu/reports/10tr033.pdf, Last accessed 10 January 2012.

[26] Humphrey, W S., *Using a Defined and Measured Personal Software Process*, IEEE Software, Vol 13, No. 3, May 1996, pp. 77-88.

[27] Humphrey, W, *The Personal Software Process*, Technical Report, Software Engineering Institute, Carnegie Mellon University, 2000.

[28] Humphrey, W, *The Team Software Process*, Technical Report, Software Engineering Institute, Carnegie Mellon University, 2000.

[29] Davis N, Mullaney J., *The Team Software Process in Practice: A Summary of Recent Results*, Technical Report, Software Engineering Institute, Carnegie Mellon University, September 2003.

[30] Veenendaal, E. *Test Maturity Model Integration (TMMi) version 3.1*, TMMi Foundation, 2010, http://www.tmmifoundation.org/downloads/tmmi/TMMi%20Framework.pdf, Last accessed 11 January 2012.

[31] Esselaar S, Gillwald A, Moyo M, Naidoo K, *South African ICT Sector Performance Review 2009/2010: Towards Evidence-based ICT Policy and Regulation*, Volume Two, Policy Paper 6, 2010.

[32] Heimann D I. *Implementing Software Metrics at a Telecommunications Company: A Case Study* in *Cases on Telecommunications and Networking*, Idea Group Inc., Pennsylvania, first edition, 2006.

[33] Fenton N, Neil M. *Software Metrics: Roadmap*. Proceedings of the Conference on the Future of Software Engineering, Ireland, June 2000, pp. 357-370.

[34] Pusala R, *Operational Excellence through Efficient Software Testing Metrics*. Infosys Technologies Limited, 2006.

[35] Lazic L, Mastorakis N. *Cost Effective Software Test Metrics*, WSEAS Transactions on Computers, Issue 6, Vol 7, June 2008, pp. 599-619.

[36] Basili V R, Weiss D. *A Methodology for Collecting Valid Software Engineering Data*. IEEE Transactions on Software Engineering, Vol 10, November 1984, pp. 728-738.

[37] Basili V R, Caldiera G, Rombach H D. *The Goal Question Metric Approach*, Encyclopaedia of Software Engineering, Vol 1, John Wiley & Sons, 1994, pp. 528-532.

[38] Buglione L, Abran A. *Balanced Scorecards and GQM: what are the differences?* Proceedings of the FESMA/AEMES Software Measurement Conference, October 2000, Madrid, Spain.

[39] Basili V, Heidrich J, Lindvall M, Münch J, Seaman C, Regardie M, Trendowicz A. *Determining the impact of business strategies using principles from goal-oriented measurement*. Proceedings of Wirtschaftsinformatik, Vienna, 2009, pp. 545-554.

[40] Niessink, F, Van Vliet H. *Measurements should generate value, rather than data*. Sixth International Software Metrics Symposium, 1999, pp. 31-38.

[41] Olsson T, Runeson P. *V-GQM: A feed-back approach to validation of a GQM study.* Seventh International Software Metrics Symposium, 2001, pp. 236-245.

[42] Basili V R, McGarry F E, Pajerski R, Zelkowitz M V. *Lessons learned from 25 years of process improvement: the rise and fall of the NASA software engineering laboratory*. Proceedings of the 24th International Conference on Software Engineering, Florida, 2002, pp. 69-79.

[43] van Solingen R, Berghout E, van Latum F. *Interrupts: Just a Minute Never Is*. IEEE Software, Vol 15, 1998, pp. 97–103.

[44] van Solingen R. *Measuring the ROI of software process improvement*. Software, IEEE Software, Vol 21, May-June 2004, pp. 32- 38.

[45] Rudzki J, Hammouda I, Mikkola T. *Agile Experiences in a Software Service Company*. Euromicro Conference on Software Engineering and Advanced Applications, August 2009, pp. 224-228, 27-29.

[46] Korhonen, K, Salo O. *Exploring Quality Metrics to Support Defect Management Process in a Multi-site Organization - A Case Study*, 19th International Symposium on Software Reliability Engineering, November 2008, pp. 213-218.

[47] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M C., Regnell, B., Wesslén, A. *Experimentation in Software Engineering: An Introduction*, Kluwer Academic Publications, Massachusetts, second edition, 2000.

[48] Kan, S H. *Metrics and Models in Software Quality Engineering*. Addison-Wesley, Boston, second edition, 2003.

[49] Schwaber, K., Beedle, M. *Agile Software Development with Scrum*, Prentice-Hall, New Jersey, first edition, 2002.

[50] Paasivaara M, Durasiewicz S, Lassenius C. *Using Scrum in Distributed Agile Development: A Multiple Case Study*. Proceedings of the 4th IEEE International Conference on Global Software Engineering, July 2009, pp. 195-204.

[51] Ionel N. *Critical Analysis of the Scrum Project Management Methodology*. Proceedings of the 4th International Economic Conference on European Integration - New Challenges for the Romanian Economy, Oradea, May 2008, pp. 435-441.

[52] Dingsøyr T, Hanssen G K, Dybå T, Anker G, Nygaard J O. *Developing Software with Scrum in a Small Cross-Organizational Project*, Proceedings of the 13th European Conference on Software Process Improvement, Finland, 2006, pp. 5-15.

[53] Beck, K., *Test Driven Development: By Example*, Addison Wesley, Boston, 2003.

[54] Abrahamsson, P., Salo O., Ronkainen J., and Warsta J. *Agile Software Development Methods: Review and Analysis*, VTT Publications, Finland, first edition, 2002.
http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf Last accessed 20 January 2012.

[55] Forrester Consulting, *Development Testing: A New Era In Software Quality*, November 2011.

Bibliography

Abrahamsson, P., Salo O., Ronkainen J., and Warsta J. Agile Software Development Methods: Review and Analysis, VTT Publications, Finland, first edition, 2002.

Basili V R, Caldiera G, Rombach H D. The Goal Question Metric Approach, Encyclopaedia of Software Engineering, Vol 1, John Wiley & Sons, 1994, pp. 528-532.

Beck, K., Test Driven Development: By Example, Addison Wesley, Boston, 2003.

Boehm B., Turner, R. Balancing Agility and Discipline: A Guide for the Perplexed, Boston, Addison-Wesley. 2004.

Boehm B W. Software Engineering Economics. Prentice-Hall, New Jersey, first edition, 1981.

Hunter J M., Thiebaud M E. Telecommunications Billing Systems: Implementing and Upgrading for Profitability. McGraw-Hill TELECOM Professional, New York, first edition, 2003.

Kan, S H. Metrics and Models in Software Quality Engineering. Addison-Wesley, Boston, second edition, 2003.

Pressman R S., Software Engineering: A Practitioner's Approach, McGraw-Hill, New York, fifth edition, 2001.

Schwaber, K., Beedle, M. Agile Software Development with Scrum, Prentice-Hall, New Jersey, first edition, 2002.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M C., Regnell, B., Wesslén, A. Experimentation in Software Engineering: An Introduction, Kluwer Academic Publications, Massachusetts, second edition, 2000.

Appendix

A Detailed Results

This section presents the complete results of the investigation, which includes results for each project and each error in the context of the metrics taken. Projects are numbered from PRJ001 to PRJ010. Errors are numbered based on the project and order in which they were discovered, for example, the first error of PRJ001 is P01E01.

A.1 Goal 01: Defect Detection Accuracy

A.1.1 Question 1.1: Current Defect Detection Accuracy

The results of M1.1.1 to M1.1.5 are shown in Table A.1, and the results of M1.1.6 to M1.1.10 are shown in Table A.2. Since project PRJ007 did not have any errors, its results were not included in the calculation of the statistics.

Table A.1: Question 1.1 results for metrics M1.1.1 to M1.1.5

Project	Total Incorrect Classifications	Total Classifications	Percentage	Above Average ?	Above 1 SD?
PRJ001	0	2	0	No	No
PRJ002	0	5	0	No	No
PRJ003	0	5	0	No	No
PRJ004	0	2	0	No	No
PRJ005	0	1	0	No	No
PRJ006	0	3	0	No	No
PRJ007	0	0	n/a	n/a	n/a
PRJ008	0	1	0	No	No
PRJ009	0	2	0	No	No
PRJ010	0	5	0	No	No
Average / % of Cases			0.00%	0%	0%
SD			0.00%		
CV			n/a		

Table A.2: Question 1.1 results for metrics M1.1.6 to M1.1.10

Project	Total False Classifications	Total Errors	Percentage	Above Average?	Above 1 SD?
PRJ001	3	5	60.00	Yes	Yes
PRJ002	1	6	16.67	No	No
PRJ003	4	9	44.44	Yes	No
PRJ004	0	2	0.00	No	No
PRJ005	6	7	85.71	Yes	Yes
PRJ006	2	5	40.00	Yes	No
PRJ007	0	0	n/a	n/a	n/a
PRJ008	0	1	0.00	No	No
PRJ009	0	2	0.00	No	No
PRJ010	1	6	16.67	No	No
Average / % of Cases			29.28%	44.44%	22.22%
SD			28.60%		
CV			97.70%		

A.1.2 Question 1.2: Sufficiency of Detection Accuracy

The results of M1.2.1 to M1.2.4 are shown in Table A.3. The cost metric M1.2.5 and metrics M1.2.6 to M1.2.10 are shown in Table A.4.

Table A.3: Question 1.2 results for metrics M1.2.1 to M1.2.4

Project	Incorrect/False Classification	Time spent (d)	Above Average?	Above 1 SD?
PRJ001	P01E01	1.00	No	No
	P01E02	17.34	No	No
	P01E03	7.00	No	No
PRJ002	P02E05	38.21	Yes	No
PRJ003	P03E02	21.16	Yes	No
	P03E06	20.00	No	No
	P03E07	20.00	No	No
	P03E08	66.23	Yes	Yes
PRJ004	none	0.00	No	No
PRJ005	P05E01	4.00	No	No
	P05E02	1.00	No	No
	P05E03	51.15	Yes	Yes
	P05E04	58.32	Yes	Yes
	P05E05	25.24	Yes	No
	P05E06	11.32	No	No
PRJ006	P06E03	90	Yes	Yes
	P06E01	7	No	No
PRJ007	none	0.00	n/a	n/a
PRJ008	none	0.00	No	No
PRJ009	none	0.00	No	No
PRJ010	P10E02	2.58	No	No
Average/ % of Cases		21.03 days	33.33%	19.05%
SD		24.99 days		
CV		118.86%		

Table A.4: Question 1.2 results for metrics M1.2.5 to M1.2.10

Project	Incorrect/False Classification	Cost	Total Testing Time (d)	Percent Test Time spent on Inaccuracy	Above Average?	Above 1 SD?
PRJ001	P01E01	R2570.40	46	2.17	No	No
	P01E02	R4474.25				
	P01E03	R17992.80				
PRJ002	P02E05	R100082.45	88	43.42	Yes	No
PRJ003	P03E02	R55143.93	34	62.25	Yes	Yes
	P03E06	R52110.00				
	P03E07	R52110.00				
	P03E08	R172569.05				
PRJ004	none	-	17	0.00	No	No
PRJ005	P05E01	R10648.80	40	10.00	No	No
	P05E02	R2662.20				
	P05E03	R136175.84				
	P05E04	R155247.24				
	P05E05	R67186.35				
	P05E06	R30129.63				
PRJ006	P06E03	R272646.00	94	95.74	Yes	Yes
	P06E01	R21205.80				
PRJ007	none	-	15	0.00	n/a	n/a
PRJ008	none	-	14	0.00	No	No
PRJ009	none	-	18	0.00	No	No
PRJ010	P10E02	R6647.17	60	4.31	No	No
Average/ % of Cases		R57128.66		21.79%	30.00%	20.00%
SD		R70732.81		32.10%		
CV		123.81%		147.31%		

A.1.3 Question 1.3: Causes of Detection Inaccuracy

The results of metrics M1.3.3 and M1.3.4 are shown in Table A.5 and Table A.6. Note that since there were no incorrect classifications, metrics M1.3.1 and M1.3.2 were not necessary.

Table A.5: Question 1.3 error results for metrics M1.3.3 and M1.3.4

Project	False Classification	Cause	Time Spent (d)
PRJ001	P01E01	Administration error	1.00
	P01E02	Misunderstanding requirements	17.34
	P01E03	Misunderstanding implementation	7.00
PRJ002	P02E05	Test tool error	38.21
PRJ003	P03E02	Test tool error	21.16
	P03E06	Test tool error	20.00
	P03E07	Unrelated error	20.00
	P03E08	Administration error	66.23
PRJ005	P05E01	Incorrect version tested	4.00
	P05E02	Misunderstanding product	1.00
	P05E03	Administration error	51.15
	P05E04	Unrelated error	58.32
	P05E05	Temporary error	25.24
	P05E06	Administration error	11.32
PRJ006	P06E03	Administration error	90
	P06E01	Administration error	7
PRJ010	P10E02	Misunderstanding implementation	2.58

Table A.6: Question 1.3 results for metrics M1.3.3 and M1.3.4

Cause	Count	Time (d)	Count %	Time %
Misunderstanding	4	27.92	23.529	6.324
Other	2	29.24	11.765	6.621
Unrelated	2	78.32	11.765	17.736
Test tool	3	79.38	17.647	17.977
Administration error	6	226.70	35.294	51.341

A.2 Goal 02: Defect Correction Accuracy

A.2.1 Question 2.1: Current Correction Accuracy

The results of metrics M2.1.1 to M2.1.5 are shown in Table A.7.

Table A.7: Question 2.1 results for metrics M2.1.1 and M2.1.5

Project	Total Incorrect Corrections	Total Corrections	Percentage	Above Average?	Above 1 SD?
PRJ001	0	2	0	No	No
PRJ002	0	5	0	No	No
PRJ003	0	4	0	No	No
PRJ004	0	2	0	No	No
PRJ005	0	1	0	No	No
PRJ006	1	3	33.33	Yes	Yes
PRJ007	0	0	n/a	n/a	n/a
PRJ008	0	1	0	No	No
PRJ009	0	2	0	No	No
PRJ010	0	4	0	No	No
Average/ % of Cases			3.70%	11.11%	11.11%
SD			10.48%		
CV			282.84%		

A.2.2 Question 2.2: Sufficiency of Correction Accuracy

The results of question 2.2 are shown in Table A.8. Note that since there was only a single case, metrics pertaining to the number of cases above the average and one standard deviation were not useful. These metrics are M2.2.3, M2.2.4, M2.2.9 and M2.2.10.

Table A.8: The results of metrics M2.2.1, M2.2.2, M2.2.5 to M2.2.8

Project	Incorrect Correction	Time spent (d)	Cost (R)	Total Testing Time	Percent Test Time spent on Inaccuracy
PRJ006	P06E02	0.094	286.11	94	0.100472813
Average		0.094	286.11		0.100472813
SD		0.00	0.00		0.00
CV		0.00%	0.00%		0

A.2.3 Question 2.3: Causes of Correction Inaccuracy

The results of question 2.3 are shown in Table A.9. Since there was only one case, the metrics were not useful.

Table A.9: Results of Question 2.3

Project	False Classification	Cause	Time Spent (days)
PRJ006	P06E02	Incorrect error analysis	0.094

A.3 Goal 03: Defect Detection Efficiency

A.3.1 Question 3.1: Current Defect Detection Efficiency

The results of metrics M3.1.1 to M3.1.5 are shown in Table A.10. The results of metrics M3.1.6 to M3.1.10 are shown in Table A.11. Note that for effectiveness calculations duplicate and false errors were not taken into account, since they are not true errors. Duplicate errors are two or more errors that are logged by different testers, which after investigation are discovered to be the same error, or arising from the same issue.

Table A.10: Results of metrics M3.1.1 to M3.1.5

Project	Actual Errors in phase	Actual Post phase errors	Effectiveness (%)	Below average?	Below 1 SD?
PRJ001	1	1	50	No	No
PRJ002	4	1	80	No	No
PRJ003	2	2	50	No	No
PRJ004	1	1	50	No	No
PRJ005	0	1	0	Yes	Yes
PRJ006	0	3	0	Yes	Yes
PRJ007	0	0	n/a	n/a	n/a
PRJ008	0	1	0	Yes	Yes
PRJ009	0	2	0	Yes	Yes
PRJ010	4	0	100	No	No
Average/ % of Cases			36.67%	44.44%	44.44%
SD			36.21%		
CV			98.75%		

Table A.11: Results of metrics M3.1.6 to M3.1.10

Project	In-phase correction time	Post-phase correction time	Weighted Efficiency (%)	Below average?	Below 1 SD?
PRJ001	24.00	21.00	53.33	No	No
PRJ002	51.13	17.38	74.63	No	No
PRJ003	23.53	61.00	27.83	Yes	No
PRJ004	19.24	7.81	71.13	No	No
PRJ005	0.00	9.19	0.00	Yes	Yes
PRJ006	0.00	8.57	0.00	Yes	Yes
PRJ007	0.00	0.00		n/a	n/a
PRJ008	0.00	14.28	0.00	Yes	Yes
PRJ009	0.00	42.19	0.00	Yes	Yes
PRJ010	4.15	0.00	100.00	No	No
Average/ % of Cases			36.33%	55.56%	44.44%
SD			37.08%		
CV			102.07%		

A.3.2 Question 3.2: Sufficiency of Detection Efficiency

The results of metrics M3.2.1 to M3.2.4 are shown in Table A.12, cost metrics' M3.2.5 to M3.2.6 results are shown in Table A.13, while Table A.14 depicts metric M3.2.7 to M3.2.10.

Table A.12: Results of metrics M3.2.1 to M3.2.4

Project	Post Test-phase Defect	Resolution Time (d)	Project Total (d)	Above Average?	Above 1 SD?
PRJ001	P01E05	21.00	21.00	Yes	No
PRJ002	P02E06	17.38	17.38	No	No
PRJ003	P03E05	20.00	61.00	Yes	Yes
	P03E09	41.00			
PRJ004	P04E02	1.00	1.00	No	No
PRJ005	P05E07	9.19	9.19	No	No
PRJ006	P06E02	0.21	8.57	No	No
	P06E04	1.24			
	P06E05	7.13			
PRJ007	none	0	0	n/a	n/a
PRJ008	P08E01	14.28	14.28	No	No
PRJ009	P09E02	41.10	42.19	Yes	Yes
	P09E01	1.09			
PRJ010	none	0.00	0.00	No	No
Average/ % of Cases			19.40	33.33%	22.22%
SD			18.91		
CV			97.48%		

Table A.13: Results of metrics M3.2.5 and M3.2.6

Project	Post Test-phase Defect	Cost (R)	Project Total	Project Total (R)
PRJ001	P01E05	R53 978.40	R5 670.00	R53 978.40
PRJ002	P02E06	R45 524.95	R4 693.29	R45 524.95
PRJ003	P03E05	R52 110.00	R16 470.00	R158 935.50
	P03E09	R106 825.50		
PRJ004	P04E02	R2 624.40	R270.00	R2 624.40
PRJ005	P05E07	R24 468.27	R2 481.57	R24 468.27
PRJ006	P06E02	R629.02	R2 314.19	R25 965.22
	P06E04	R3 741.38		
	P06E05	R21 594.82		
PRJ007	none	n/a	n/a	n/a
PRJ008	P08E01	R42 942.61	R3 854.81	R42 942.61
PRJ009	P09E02	R105 536.75	R11 391.82	R108 336.26
	P09E01	R2 799.51		
PRJ010	none	n/a	n/a	n/a
Average				R51 419.51
SD				R48 661.61
CV				94.64%

Table A.14: Results of metrics M3.2.7 to M3.2.10

Project	Post-test phase Defect	Test Time	% of Test Time on Post Test-phase Errors	Project Total (%)	Above Average?	Above 1 SD?
PRJ001	P01E05	46.00	45.65	45.65	0	0
PRJ002	P02E06	88.00	19.75	19.75	0	0
PRJ003	P03E05	34.00	58.82	179.41	1	1
	P03E09	34.00	120.59			
PRJ004	P04E02	17.00	5.88	5.88	0	0
PRJ005	P05E07	40.00	22.98	22.98	0	0
PRJ006	P06E02	94.00	0.22	9.12	0	0
	P06E04	94.00	1.31			
	P06E05	94.00	7.58			
PRJ007	none	15.00	n/a	n/a	n/a	n/a
PRJ008	P08E01	14.00	101.98	101.98	1	0
PRJ009	P09E02	18.00	228.34	234.40	1	1
	P09E01	18.00	6.06			
PRJ010	none	60.00	0.00	0.00	0	0
Average/ % of Cases				68.80%	33.33%	22.22%
SD				80.31%		
CV				116.74%		

A.3.3 Question 3.3: Causes of Detection Inefficiency

The results of metrics M3.3.1 and M3.3.2 are shown in Table A.15 and Table A.16. Data for metrics M3.3.3 to M3.3.6 is listed in Table A.17. Note that project size is measured by the number of modules that were added or modified for the project.

Table A.15: Question 3.3 error results

Project	Post Test-phase Defect	Cause	Time (d)	Project Size	Test cases
PRJ001	P01E05	Test case not tested properly	21.00	5	33
PRJ002	P02E06	No test case coverage	17.38	17	79
PRJ003	P03E05	Misunderstanding requirements	20.00	8	28
	P03E09	No test case coverage	41.00	8	28
PRJ004	P04E02	No test case coverage	1.00	6	10
PRJ005	P05E07	No test case coverage	9.19	12	45
PRJ006	P06E02	No test case coverage	0.21	2	18
	P06E04	No test case coverage	1.24	2	18
	P06E05	Misunderstanding requirements	7.13	2	18
PRJ007	none		0.00	2	19
PRJ008	P08E01	Misunderstanding requirements	14.28	1	11
		No test case coverage	14.28	1	11
PRJ009	P09E02	Insufficient information	41.10	4	16
	P09E01	Misunderstanding requirements	1.09	4	16
PRJ010	none		0.00	2	38

Table A.16: Question 3.3 results for metrics M3.3.1 and M3.3.2

Cause	Count	Time spent	Count %	Time %
Insufficient/No Test Case Coverage	8	105.29	61.54	55.74
Misunderstanding Requirements	4	42.50	30.77	22.50
Other	1	41.10	7.69	21.76

Table A.17: Data for metrics M3.3.3 to M3.3.6

Project	Post Test-phase defects	Time spent (days)	Size	Test cases
PRJ001	1	21.00	5	33
PRJ002	1	17.38	17	79
PRJ003	2	61.00	8	28
PRJ004	1	1.00	6	10
PRJ005	1	9.19	12	45
PRJ006	3	8.57	2	18
PRJ007	0	0.00	2	19
PRJ008	1	28.55	1	11
PRJ009	2	42.19	4	16
PRJ010	0	0.00	2	38

A.4 Goal 04: Defect Correction Efficiency

A.4.1 Question 4.1: Current Defect Correction Speed

The results for this question are shown in Table A.18 to Table A.19. Note that only actual errors were used for this goal, since false and duplicate errors cannot be considered as true errors.

Table A.18: Results of metrics M4.1.1 to M4.1.4

Project	Defect	Correction Time (d)	Above Average?	Above 1 SD?
PRJ001	P01E04	24.00	Yes	No
	P01E05	21.00	Yes	No
PRJ002	P02E01	12.07	No	No
	P02E02	20.00	Yes	No
	P02E03	13.00	Yes	No
	P02E04	6.06	No	No
	P02E06	17.38	Yes	No
PRJ003	P03E01	6.16	No	No
	P03E03	17.37	Yes	No
	P03E05	20.00	Yes	No
	P03E09	41.00	Yes	Yes
PRJ004	P04E01	19.24	Yes	No
	P04E02	7.81	No	No
PRJ005	P05E07	9.19	No	No
PRJ006	P06E02	0.21	No	No
	P06E04	1.24	No	No
	P06E05	7.13	No	No
PRJ007	none	n/a	n/a	n/a
PRJ008	P08E01	14.28	Yes	No
PRJ009	P09E02	41.10	Yes	Yes
	P09E01	1.09	No	No
PRJ010	P10E01	1.84	No	No
	P10E03	0.77	No	No
	P10E04	0.77	No	No
	P10E05	0.77	No	No
Average/ % of Cases		12.64 days	45.83%	8.33%
SD		11.40 days		
CV		90.19%		

Table A.19: Results of metrics M4.1.5 to M4.1.8

Project	Defect	Test Time (d)	Percent of Test Time taken to resolve	Above Average?	Above 1 SD?
PRJ001	P01E04	46	52.17	Yes	No
	P01E05	46	45.65	Yes	No
PRJ002	P02E01	88	13.72	No	No
	P02E02	88	22.73	No	No
	P02E03	88	14.77	No	No
	P02E04	88	6.89	No	No
	P02E06	88	19.75	No	No
PRJ003	P03E01	34	18.10	No	No
	P03E03	34	51.10	Yes	No
	P03E05	34	58.82	Yes	No
	P03E09	34	120.59	Yes	Yes
PRJ004	P04E01	17	113.16	Yes	Yes
	P04E02	17	45.93	Yes	No
PRJ005	P05E07	40	22.98	No	No
PRJ006	P06E02	94	0.22	No	No
	P06E04	94	1.31	No	No
	P06E05	94	7.58	No	No
PRJ007	none	15	n/a	n/a	n/a
PRJ008	P08E01	14	101.98	Yes	Yes
PRJ009	P09E02	18	228.34	Yes	Yes
	P09E01	18	6.06	No	No
PRJ010	P10E01	60	3.07	No	No
	P10E03	60	1.28	No	No
	P10E04	60	1.28	No	No
	P10E05	60	1.28	No	No
Average/ % of Cases			39.95%	37.50%	16.00%
SD			52.80%		
CV			132.16%		

A.4.2 Question 4.2: Sufficiency of Correction Speed

The results for this question are shown in Table A.20 and Table A.21. Note that only cases with above average times are relevant for these metrics.

Table A.20: Results of metrics M4.2.1 to M4.2.4

Project	Defect	Above Average Time (d)	Above Average?	Above 1 SD?
PRJ001	P01E04	11.36	No	No
	P01E05	8.36	No	No
	P02E02	7.36	No	No
	P02E03	0.36	No	No
	P02E06	4.74	No	No
	P03E03	4.73	No	No
	P03E05	7.36	No	No
	P03E09	28.36	Yes	No
PRJ004	P04E01	6.59	No	No
PRJ008	P08E01	1.63	No	No
PRJ009	P09E02	28.46	Yes	No
Average/ % of Cases		9.93 days	18.18%	0%
SD		9.18 days		
CV		92.36%		

Table A.21: Results of metrics M4.2.5 to M4.2.8

Project	Defect	Above Average Time as percentage of project time	Above Average?	Above 1 SD?
PRJ001	P01E04	24.69	No	No
	P01E05	18.16	No	No
	P02E02	8.36	No	No
	P02E03	0.40	No	No
	P02E06	5.38	No	No
	P03E03	13.91	No	No
	P03E05	21.63	No	No
	P03E09	83.40	Yes	Yes
PRJ004	P04E01	38.78	Yes	No
PRJ008	P08E01	11.66	No	No
PRJ009	P09E02	158.10	Yes	Yes
Average/ % of Cases		34.95%	27.27%	18.18%
SD		44.57%		
CV		127.52%		

A.4.3 Question 4.3: Costs of Defect Correction

The results of metrics M4.3.1 to M4.3.4 are shown in Table A.22.

Table A.22: Results of metrics M4.3.1 to M4.3.4

Project	Defect	Correction Cost (R)	Above Average?	Above 1 SD?
PRJ001	P01E04	R61 689.60	Yes	Yes
	P01E05	R53 978.40	Yes	No
PRJ002	P02E01	R31 612.42	No	No
	P02E02	R52 380.00	Yes	No
	P02E03	R34 047.00	Yes	No
	P02E04	R15 871.81	No	No
	P02E06	R45 524.95	Yes	No
PRJ003	P03E01	R16 037.46	No	No
	P03E03	R45 265.56	Yes	No
	P03E05	R52 110.00	Yes	No
	P03E09	R106 825.50	Yes	Yes
PRJ004	P04E01	R50 484.13	Yes	No
	P04E02	R20 490.37	No	No
PRJ005	P05E07	R24 468.27	No	No
PRJ006	P06E02	R629.02	No	No
	P06E04	R3 741.38	No	No
	P06E05	R21 594.82	No	No
PRJ007	none	-	n/a	n/a
PRJ008	P08E01	R42 942.61	Yes	No
PRJ009	P09E02	R105 536.75	Yes	Yes
	P09E01	R2 799.51	No	No
PRJ010	P10E01	R4 738.79	No	No
	P10E03	R1 982.24	No	No
	P10E04	R1 982.24	No	No
	P10E05	R1 982.24	No	No
Average/ % of Cases		R31 948.60	45.83%	12.50%
SD		R29 619.91		
CV		92.71%		

A.4.4 Question 4.4: Sufficiency of Correction Costs

The results for metrics M4.4.1 to M4.4.3 are shown in Table A.23 below. Note that only cases above the average were considered for these metrics.

Table A.23: Results of metrics M4.4.1 to M4.4.4

Project	Defect	Cost Above Average (R)	Above Average?	Above 1 SD?
PRJ001	P01E04	R29 188.02	Yes	No
	P01E05	R21 476.82	No	No
PRJ002	P02E02	R19 263.89	No	No
	P02E03	R930.89	No	No
	P02E06	R12 408.84	No	No
PRJ003	P03E03	R12 320.15	No	No
	P03E05	R19 164.59	No	No
	P03E09	R73 880.09	Yes	Yes
PRJ004	P04E01	R17 299.74	No	No
PRJ008	P08E01	R4 910.29	No	No
PRJ009	P09E02	R73 069.30	Yes	Yes
Average/ % of Cases		R25 810.24	27.27%	18.18%
SD		R23 647.29		
CV		91.62%		

A.4.5 Question 4.5: Causes of Correction Inefficiency

The data for metrics M4.5.1 and M4.5.2 is shown in Table A.24 and the results for these metrics are shown in Table A.25 below. Table A.26

Table A.24: Data for metrics M4.5.1 and M4.5.2

Delayed Correction	Cause	Correction Time (d)	Project Size
P01E04	Complex dependencies	24.00	5
P01E05	Major correction	21.00	5
P02E02	Other concurrent errors	20.00	17
P02E03	Other concurrent errors	13.00	17
P02E06	Major correction	17.38	17
P03E03	Complex dependencies	17.37	8
P03E05	Complex dependencies	20.00	8
P03E09	Lack of information	41.00	8
P04E01	Major correction	19.24	6
P08E01	Major correction	14.28	1
P09E02	Lack of information	41.10	4

Table A.25: Results of metrics M4.5.1 and M4.5.2

Cause	Count	Time (d)	Count %	Time %
Complex dependencies	3	61.37	27.27	24.71
Major correction	4	71.90	36.36	28.95
Other concurrent errors	2	33.00	18.18	13.29
Lack of information	2	82.10	18.18	33.06

Table A.26: Data for metrics M4.5.3 and M4.5.4

Project	Error	Project Size	Correction Time
PRJ001	P01E04	5	24.00
	P01E05	5	21.00
PRJ002	P02E01	17	12.07
	P02E02	17	20.00
	P02E03	17	13.00
	P02E04	17	6.06
	P02E06	17	17.38
PRJ003	P03E01	8	6.16
	P03E03	8	17.37
	P03E05	8	20.00
	P03E09	8	41.00
PRJ004	P04E01	6	19.24
	P04E02	6	7.81
PRJ005	P05E07	12	9.19
PRJ006	P06E02	2	0.21
	P06E04	2	1.24
	P06E05	2	7.13
PRJ008	P08E01	1	14.28
PRJ009	P09E02	4	41.10
	P09E01	4	1.09
PRJ010	P10E01	2	1.84
	P10E03	2	0.77
	P10E04	2	0.77
	P10E05	2	0.77

A.5 Concluding Remarks

The data collected and results of each metric used in the investigation have been presented. Clarification on certain calculations and assumptions has been made where required.