Improvements For An Embedded Software Testing Method

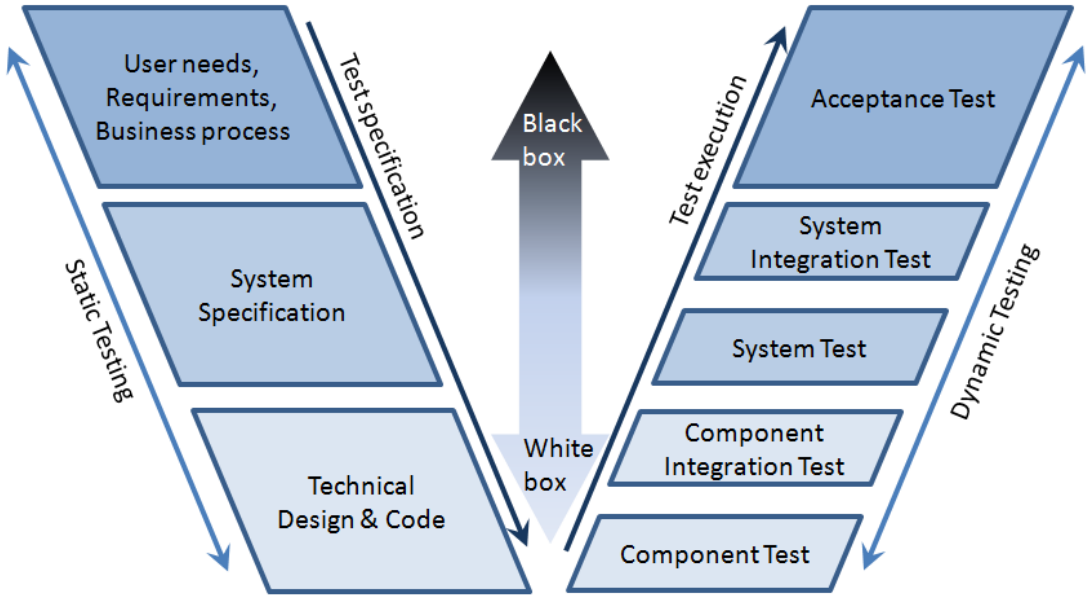
KRISTIAN GUSTAFSSON

KTH Industrial Engineering
and Management

Master of Science Thesis
Stockholm, Sweden June 2010

Improvements For An Embedded Software Testing Method

Kristian Gustafsson



Picture is inspired from Chris C. Schotanus book [1], page 6.

Master of Science Thesis MMK 2010:46 MDA 381
KTH Industrial Engineering and Management
Machine Design
SE-100 44 STOCKHOLM

KTH Industrial Engineering and Management

Master of Science Thesis MMK 2010:46 MDA 381

Improvements For An Embedded Software Testing Method

Kristian Gustafsson

Approved	Examiner	Supervisor
2010-06-03	Martin Törngren	Tahir Naseer
	Commissioner	Contact person
	Syntronic AB	Fredrik Lie

Abstract

Testing is a widespread subject that usually is divided into hardware testing and software testing. The software testing can be done in many different ways and there exist several standards and guidelines for successful testing.

This master thesis has investigated the possibilities for improvement of a software test method used by a large Swedish company. The work has been carried out at the Integration and Verification department at the company. An investigation of the undetected defects during the test execution is carried out to explore those characteristics, which are not covered by the current testing method.

The purpose with this master thesis is to evaluate the company's testing method and to give suggestions for possible improvements in the process during the system integration phase of the development process

One part of the work includes a survey by interviewing key personnel's for getting a better insight of the problem, a thorough literature study and attendance at a course providing an overview of the system.

The other part of the work was the examination of a database storing all the data related to the faults from different projects. The result from the examination shows that 4.4 percent of all the faults submitted are found after the testing phase, 79.1 percent of these faults are related to the software in the system. Further investigation showed that 51.2 percent of the total number of faults found after the test phase where related to the systems configuration database and the administrator tool for the system.

The conclusion to be drawn from these results is that the testing method being used by the company is good. However, there is room for improvement for some parts of the method. Most of the faults discovered after the testing process are faults related to the system configuration database and the administrator tool for the system. The method for testing these two parts of the system should be reviewed and improved.

KTH Industriell teknik och management

Examensarbete MMK 2010:46 MDA 381

Förbättringar av en testmetod för ett inbyggt system

Kristian Gustafsson

Godkänt	Examinator	Handledare
2010-06-03	Martin Törngren	Tahir Naseer
	Uppdragsgivare	Kontaktperson
	Syntronic AB	Fredrik Lie

Sammanfattning

Testning är ett stort ämne som vanligtvis delas in i hårdvaru- och mjukvarutestning. Mjukvaran i ett system kan testats på många olika sätt och det finns idag flera olika standarder och riktlinjer för hur en lyckad testning skall gå till.

Detta examensarbete har utrett möjligheterna att förbättra en testmetod för mjukvara som ett stort svenskt företag använder. Arbetet har utförts på Integrations- och verifieringsavdelningen på företaget. De fel som har förblivit oupptäckta efter den ordinarietestningen har blivit undersökta för att se om dessa har något karaktärsdrag som inte stöds utav den aktuella testmetoden.

Syftet med detta arbete är att utvärdera företagets testmetod och komma med förslag till förbättringar av systemintegrationsfasen av utvecklingsprocessen.

En del av arbetet har innefattar en undersökning där nyckelpersoner har blivit intervjuade för att få djupare insikt av problemet, men även en litteraturstudie har genomförts samt deltagande i en kurs som gav en överblick över hur systemet fungerade.

Den andra delen av arbetet var undersökningen utav en databas som innehåller all information som är relaterad till fel från olika projekt. Resultatet från undersökningen visar att 4,4 procent av alla inrapporterade fel är upptäckta efter avslutad testning och att 79,1 procent av dessa fel är relaterade till mjukvaran i systemet. Vidare undersökning visade att 51,2 procent av det totala antalet fel efter avslutad testning var relaterade till systemets konfigurations databas och administrationsverktyget för systemet.

Slutsatserna man kan dra utifrån dessa resultat är att företagets testmetod är bra, men det går att förbättra vissa delar av den. De flesta felen som upptäcktes efter testprocessen var relaterade till systemets konfigurations databas samt till systemets administrativa verktyg. Den testmetod som används för att testa dessa två borde ses över och förbättras.

Acknowledgements

I would like to thank my supervisor Fredrik Lie at Syntronic for all support and guidance with my master thesis. I also want to thank my supervisor Tahir Naseer at KTH for helping me to complete the report in a good way.

I would like to thank Torbjörn Hansson and Catarina Vesterlund at the Integration and Verification department for all their help and support with finding information and to make sure that I got it all right. I also want to thank Laura Rivera from the same department for showing me how to search the database and how I should do to get the right information.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Testing of a system consisting of both hardware and software can be performed in many different ways. The hardware in the product can be tested for their withstand ability during earthquakes or electromagnetic compatibility. The software can be tested in many different ways with different methods. There exist several standards and guidelines for software testing. Software testing can roughly be divided into two levels. The first level is testing of each written function, the second level is integration between these functions and the third level is system testing. The testing at different levels results in finding different faults. The low level testing might find syntax faults in the code and the higher level testing might find faults related to how the data is transferred between functions. This master thesis has been carried out at Syntronic AB in cooperation with a Swedish company. The intention for this master thesis is to investigate the company's testing method. The work has been carried out at the Integration and Verification department at the company. This master thesis will focus on the faults, which are discovered during the late stages of different projects to see if these faults have some characteristic that remains undetected due to the company's way of testing.

Due to confidentiality reasons, the report lacks some information in addition to the usage of fictitious data where applicable. This includes the change of names and abbreviations along with exclusion of the description of the system under consideration.

1.1.1 Goals

This master thesis has tried to answer the following questions:

1. Which test methods are used in the industry?
2. In which phase of a system development can the most number of faults be discovered?

3. Which part of the system under consideration has more faults as compared to others?

4. How can we improve the testing method for an embedded software system?

1.2 Purpose and limitations

This master thesis focuses on software testing at the system integration test level in the V-model, see Figure 1.1. The System Under Consideration (SUC) will be investigated as one complete system.

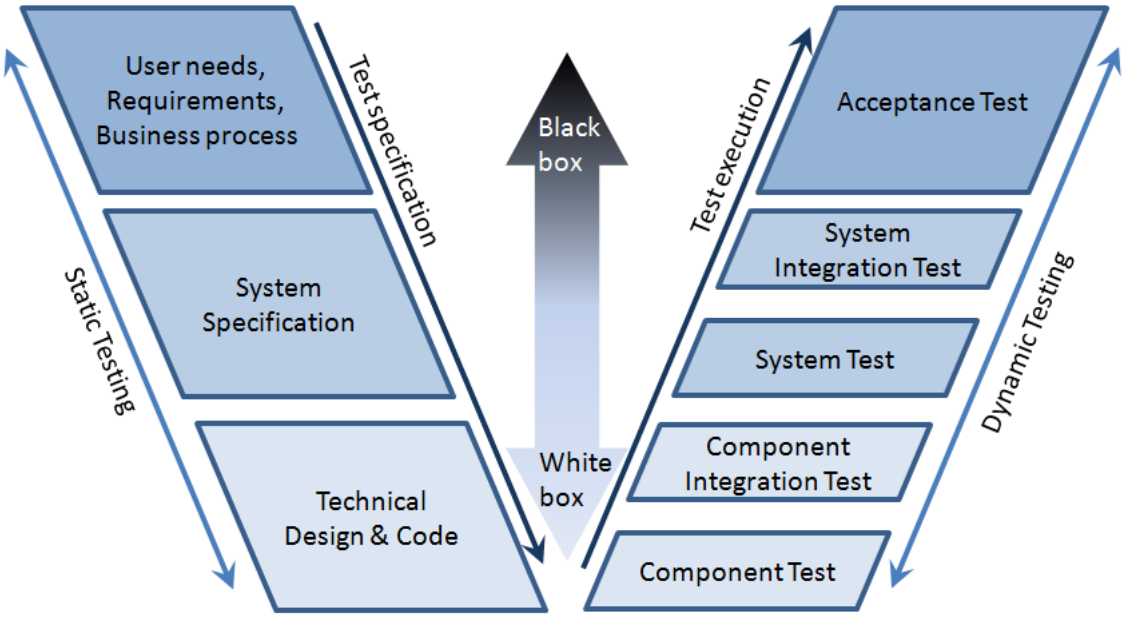


Figure 1.1: The V-model of system development and testing. Picture is inspired from Chris C. Schotanus book [1], page 6.

The second thing this master thesis will focus on is faults that slip through the company's testing. Only projects related to the complete SUC will be investigated.

1.3 Method

This master thesis work has been divided into three phases: pre-study, data gathering, and analysis. The pre-study consists of three parts: in-depth studies of the SUC, in-depth studies of software testing methods and in-depth study of the testing method used at the company. The pre-study phase also includes interviews, literature study and by a course on the system under consideration.

The company's database has been used to gather data about the projects and the faults related to them. This data is filtered and analyzed.

The results from the data collection are presented in the last phase of this study.

1.4 Structure

The outline for this master thesis is as follows: Chapter 2 describes the complexity of software testing and the test method under consideration. Chapter 3 describes how the data has been collected and how it has been filtered. Finally the results, analysis and conclusions are presented in Chapter 4.

1.5 Abbreviations

- DR = Defect Report
- FCS = First Customer Site
- FST = Fault Slip Through
- I&V = Integration and Verification
- RFA = Ready For Acceptance
- SUC = System Under Consideration

Chapter 2

Pre-studies

2.1 Software testing

Testing is a very broad area and the result depends very much on which method has been used and where this method is effective and the experience level of people carrying out the testing. Different methods are used to find different types of faults. How fault, error and failure are defined is of importance to understand testing, see Figure 2.1. A software fault originates from the code and it can cause an error. If the error becomes visible in the output it has turned into a failure. Failures and errors may cause more faults. Usually a Defect Report (DR) is written where the failure is described. There is no one-to-one relationship between a fault and a failure. One fault can cause several failures and different faults may lead to the same failure. These errors and failures, if they do not become visible, can result in another fault later on during the tested software's execution or development. These faults are often harder to find since they can affect another part of the system. The developer might make a workaround for the new problem leaving the actual source of the fault unfixed [2].

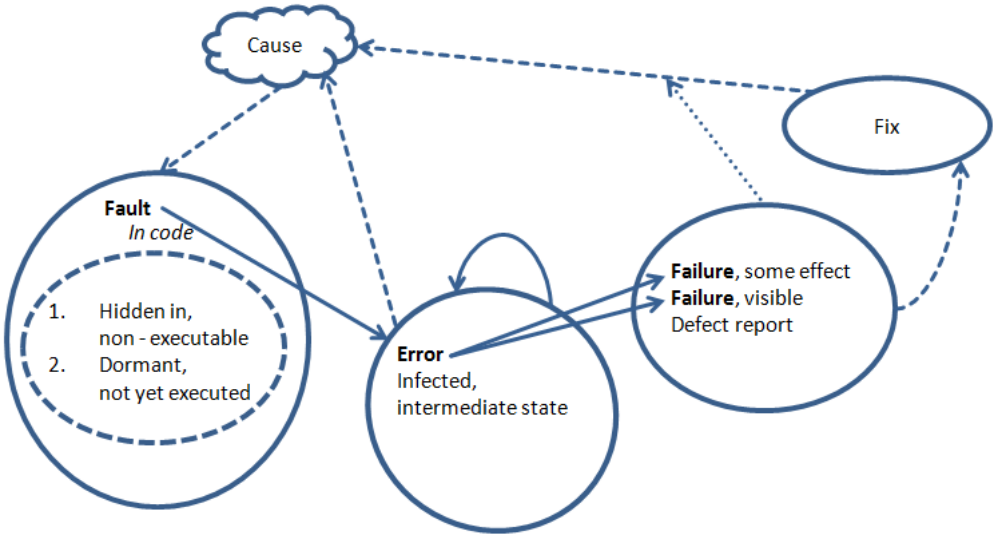


Figure 2.1: Definition of fault, error and failure. Picture is inspired from Sigrid Eldh's licentiate [2], page 101.

Test methods can be divided into two categories: static testing and dynamic testing [1], see Figure 2.2.

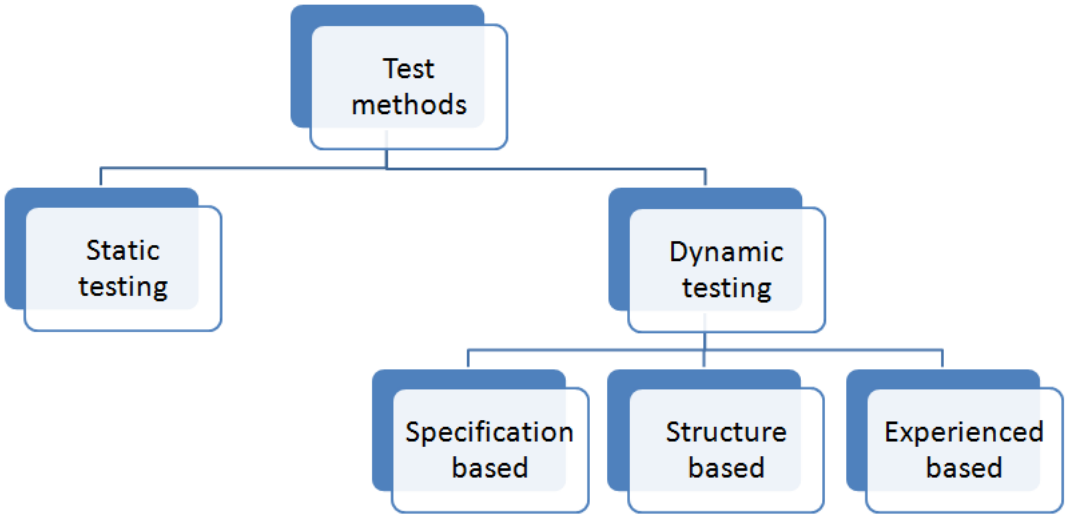


Figure 2.2: The two categories in testing.

- Static testing corresponds to e.g. reviewing of the code and the requirement specification. The goal with static testing is to find syntax error and requirements that are contradictory to each other. Static testing does not require the code to be finished or complete. Static testing is usually carried out in the left part of the V-model, see Figure 1.1.
- Dynamic testing is done when the code exists and can be executed, it is usually carried out on the right side of the V-model, see Figure 1.1.

Dynamic testing can be broken down into three subcategories: Specification based, structure based and experience based testing. Most of the specification based and structure based methods are described very thoroughly and are easy to apply on different systems.

The first two categories use different "methods" to measure the coverage, which is different for each method. One of the key personnel interviewed states *"One weakness associated by using a test method is that all faults have the same importance"*. However, the faults in a system do not have the same importance e.g. a fault in the breaking system of a car is more important then a fault in the electric windows. This is the reason why experience testing should be carried out as well. The experienced testers have the knowledge of how the system behaves and which functions are vital for the systems functionality. The focus of the experience testing is to find the important faults. Usually these tests are better in finding faults that matters compared to the methods. One reason for this can be that the tester knows where to look for faults.

The three dynamic methods usually find different faults since they have a different scope. Specification based testing will find faults that are connected to the requirement specification, the blue circle numbered 1 in Figure 2.3. It is not possible to find the faults in the implemented functions that are not specified in the system requirements. On the contrary structure based testing will only find faults in the implementation, the brown circle numbered 2 in Figure 2.3, and therefore miss the required parts that are not implemented. There is a risk with experienced testing as well. If the scope of their testing is too small the experts might only test some specific functions of the implementation and miss a lot, see circle 3 in Figure 2.3. However, the experienced testing can end up anywhere in the other two circles.

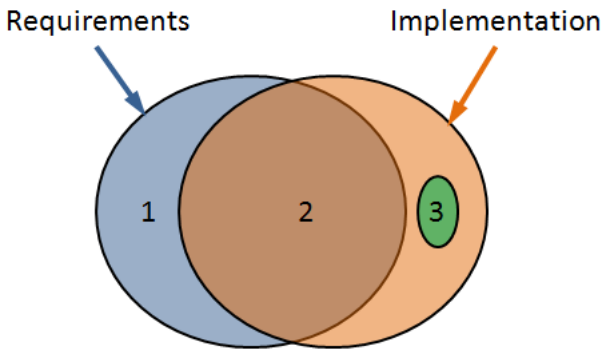


Figure 2.3: The scope for specification based, structure based and experienced testing methods. Picture is inspired from Paul C. Jorgensen [3].

These three categories use usually White-box or Black-box oriented approaches. The term Black-box testing and White-box testing are widely used today to describe the approach being used. However, just stating that it is a Black-box test is not enough since there are several different methods that are Black-box oriented. The same applies fo the White-box testing.

Black-box testing

When using a Black-box method the system is supposed to be completely unknown [1, 3, 4]. The only two things that can be monitored and controlled are the inputs and outputs, see Figure 2.4.

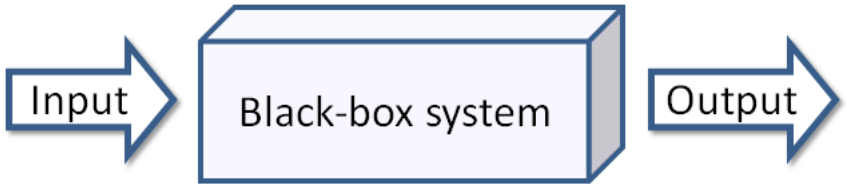


Figure 2.4: Black-box system under test.

The specifications are important when creating test cases for a Black-box, hence the specification holds the information about how the system should behave under specified circumstances. This makes it easier for the tester to verify if the functionality of the system under test complies with the specification or not. Examples of Black-box methods include boundary value analysis, pair-wise testing and equivalence partitioning [5].

White-box testing

The White-box testing technique is also called glass-box testing, clear-box testing, transparent-box testing etc. All these names refer to a well known and well documented system [1, 3, 4]. When using a White-box method all the parameters in the system are known and the system can be monitored and manipulated from the inside in a way which is not possible in Black-box testing, see Figure 2.5.

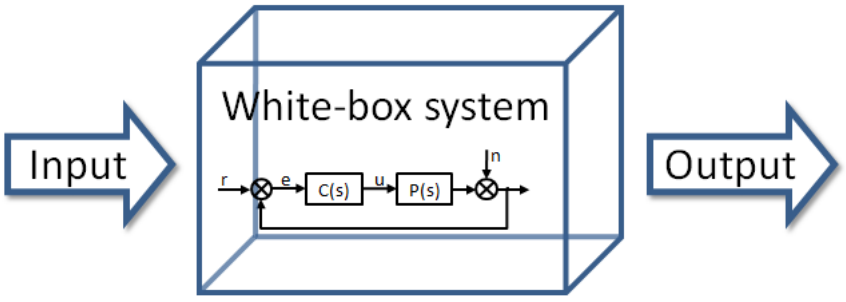


Figure 2.5: White-box system under test.

By using a White-box method it becomes easier to measure how much of the system that actually has been tested, since it is easier to see which parts of the system that has been tested and what parts that has been untested. A White-box method also allows the tester to create and select test cases better than the Black-box method. Examples of White-box methods include branch testing [5], code coverage testing [6] and data flow testing [5].

Black-box testing vs. White-box testing

The advantage with Black-box testing is just that the tester has no clue about how the system works and it is possible to test functions in a way that the developer would not have done. On the contrary, the tester does not know anything about the system and may create several test cases to test a function when it had been sufficient with one test case. One advantage with White-box testing is that the tester can make clever choices when selecting test cases and create a test suite that is more or less optimal for the system. A possible drawback is that the tester might miss some parts that are not so obvious to test and therefore are left out during test execution.

2.1.1 Industrial software testing standards

2.1.1.1 BS7925-2

There is a standard from the British Computer Society [5](BCS)(standard BS7925-2) which describes and gives examples about how to test software components. This standard covers five different areas: specified components, dynamic execution, techniques and measures, test process attributes and generic test process. With specified components it is meant that a software component needs to have a specification to be able to test accordingly to this standard. This standard supports dynamic execution[1] and analysis of the results. The standard also defines test case design techniques and test measurements techniques. These techniques are defined in such a way so that the user easily can design test cases and quantify the testing. Test process attributes are used to assess, compare and to improve test quality. The generic test process is defined to ensure that this standard conforms to the different requirements in the software industry.

2.1.1.2 Institute of Electrical and Electronics Engineers

Institute of Electrical and Electronics Engineers (IEEE) have a number of standards about software testing. One of them is standard 829 [8]. This standard creates a common base for test documentation for the software's complete life cycle e.g. supply, development and maintenance. It defines the content of the master test plan, the level test plan and the other related test documentation. Standard 829 also defines the test tasks required inputs and outputs.

Another document from IEEE is standard 1008 [9]. This standard's aim is to define how to approach software unit testing in a systematic and documenting way. It uses design and implementation information together with the requirements to determine the test coverage.

[1]Dynamic execution is a combination of several techniques introduced by Intel in their P6 microarchitecture [7].

2.1.1.3 The successor standard

In May 2007 a working group (WG26) were assigned a new task by the ISO/IEC JTC1/SC7 Software and System Engineering committee [10]. The task was to develop a new international standard for software testing. The new standard is named ISO/IEC 29119 [11]. This standard will replace some of the existing IEEE and BCS standards such as:

- IEEE 829 Test Documentation [8]
- IEEE 1008 Unit Testing [9]
- BS 7925-1 Vocabulary of Terms in Software Testing [12]
- BS 7925-2 Software Component Testing Standard [5]

The ISO/IEC 29119 standards scope is divided into four parts.

1. **Concepts and vocabulary**
   This part aims to describe and give an overview of software testing and to provide a vocabulary for the subject.

2. **Test process**
   This part will support a testing process model which can be used anywhere in a software development and its testing life cycle.

3. **Test documentation**
   This part will cover the test documentation for software testing. The standard will supply templates to be used for all the parts in the testing process model.

4. **Test technique**
   This part will describe all kind of software testing e.g. static, dynamic and experiences-based.

It is important to note that this standard is a working draft and there might be some changes before it is a final international standard. It is expected to be completed in October 2012.

2.1.2 Test method evaluation aspects

How to compare test methods with each other is non-trivial since every method has its own way to measure its quality. Another factor is that different test methods are more likely to find different faults. To be able to compare the different methods and to have some sort of knowledge about the software faults, all faults must be known beforehand. If all faults in the system are known or injected then there will be a maximum number of faults available. It will also be easier to compare the methods against each other. A maximum number of faults will make it easier to point out the faults missed or found by the used method. It would also be possible, if there are enough faults injected, to see a pattern in the faults found by the method. The information acquired from the pattern could be of great usage when selecting test

methods. However, it is also possible that the pattern is only applicable for the specific types of product/system where the pattern was originated.

Even though there are many things that affect different methods in different ways, one of the test expert interviewed states *"There are four aspects that will make the comparison between the methods more convenient: requirements coverage, structure/code coverage, test suite size and fault injection."*. These four aspects are and their pros and cons are as follows:

I **Requirement coverage**
Requirement coverage assumes that the requirements are well defined and that there is traceability towards the test programs. It is possible that the implementation differs from the specification, see Figure 2.6. The aim when developing any product is to meet all the requirements. Number 1 in the figure represents requirements that were never meet by the implementation. This results in a software that does not work as expected and required. Number 3 represents functions and features in the implementation which were not in the requirements. This could be desired functionality, which was forgotten in the requirement phase. It could also be unwanted functionality or even actual mistakes from the programmer. Number 2 represents the requirements which are actually implemented.

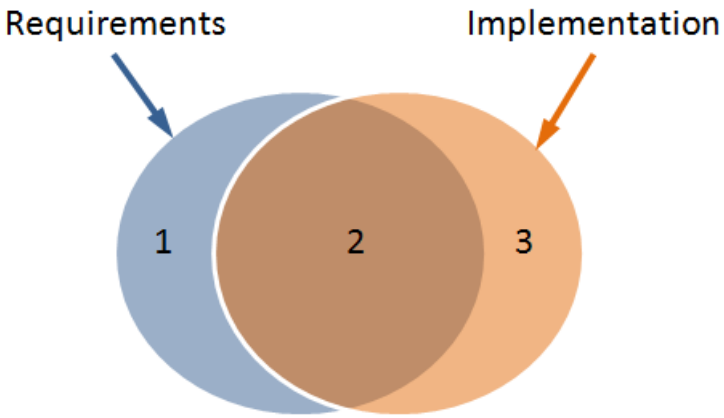


Figure 2.6: Illustration of how the implementation can differ from the original requirements.

II **Interface structure coverage and code coverage**
The code coverage is to measure how much of the code gets executed by a test case or a test suite. Code coverage can only be done in a White-box environment. The reason is that the execution of the code needs to be monitored. By monitoring the execution it is possible to see which parts of the code have actually been executed. This information is then used to calculate the percentage of the code that has been run.

The meaning with interface structure coverage is to test all graphical objects in an application e.g. buttons and text fields. Interface structure coverage is

usually used when a graphical user interface needs to be tested. This is done to make sure that every button and menu item is clicked and tested.

III **Test suite size**
The test suite size is the number of test cases in a test suite. The test suite size is easy to calculate and therefore it is often used to compare different methods. The number of test cases needed and the number of faults found is used to evaluate how efficient the method is.

IV **Fault injection**
The idea of inducing faults into a system has one advantage: the faults will be known and it is easy to have an absolute number of them. However, there are some disadvantages with this method. It is impossible to decide if the faults induced are representative for the system. It is also difficult to compare the different test methods against each other since the faults induced might favor one of the methods. A better approach to measure fault discovery is to use defects reports and see what faults that have been found, defects reports are described further in section 3.2. These faults can then be used to see if the test cases selected by the test method will find them

Fault injection can be used in the scheduled testing to verify the error handling functions.

2.2 Test method under consideration

This section describes the subject test method of this thesis. The information in this section has been obtained through interviews and by attending a course handling the SUC. The interviews have been carried out at three companies: the company which this method is applied and two consultant companies which are involved with the first company. The interviews have been spread out over the following areas of expertise:

- Verification Strategy Manager
- Release Coordinator
- Project Manager
- Test Expert
- Test Developer
- Software Developer
- Test Coordinator

The company uses an own developed project management model. At System, Design and I&V they use a product development process to develop the products. This process is described in section 2.2.1.

Before I&V department start their verification process they need to have a hardware that is approved, this means that the hardware in the product needs to be tested and verified before the beginning of the hardware and software integration process.

I&V does not verify the hardware, they integrate the software with the hardware and verify the software functionality. The system consists of several software components. To ease the integration between these components each component has a well defined interface for communication, see Figure 2.7. When the components have been integrated to a first build they perform a smoke test, which is described in section 2.2.1.2.

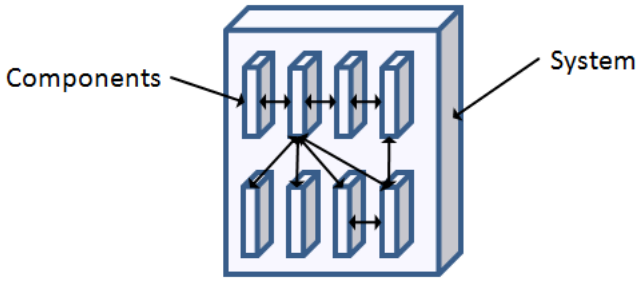


Figure 2.7: The arrows between the components represent the interfaces with each other.

2.2.1 Product development process

Most of the projects are related to the introduction of a new function/feature into already existing hardware and software. The process model consists of four main phases. The I&V department are not involved in all these phases. They are mainly involved in the phases of product specification and verification of the product in system environment, see Figure 2.8.

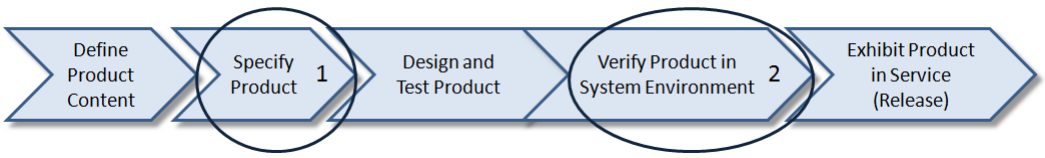


Figure 2.8: The "top level" of the development process with the important phases for the department in circles.

The process consist of several layers and this is the top layer of it, see Figure 2.8. These phases are just headings and inside these phases are more detailed models. Figure 2.8 shows the overall process. However, the I&V department has its own process. This process is divided into three parts i.e. Planning, Execution and Conclusion. Their description and the correspondence with the overall process are described as follows:

2.2.1.1 Planning

The planning at the I&V department is done in the Specify Product phase, numbered one in Figure 2.8. In this phase there exist three sub phases: Verification Survey, Risk Based Verification Analysis and Verification Planning as shown in Figure 2.9.

In these three phases the method and strategy for the verification of the new product is planned.

Figure 2.9: The Specify Product phase.

Verification Survey

In this phase the proposed requirements and the solutions are studied. An investigation of how the different solutions will affect the work in the latter phases is carried out. This is done to give valuable input and feedback to the system design team. If it turns out that the proposed solution will be expensive and an alternative solution would be cheaper the system team will be notified.

Risk Based Verification Analysis

The Risk Based Verification Analysis is performed in two ways and the results are presented as product and project risk exposure. The method used by the company comes from a Master thesis written by Yanping Chen [13]. However, the method has been manipulated to fit their way of working. The aim for this analysis is to create support for planning of the project, what to verify first and last. The parameters to be analyzed are complexity to verify and time to verify. These two factors are then merged into a risk exposure. This work is done early in the project. The result of the risk analysis is proof which back up the verification plan. If the conditions changes for any of the parts being analyzed the risk evaluation should be recalculated. See Figure 2.10.

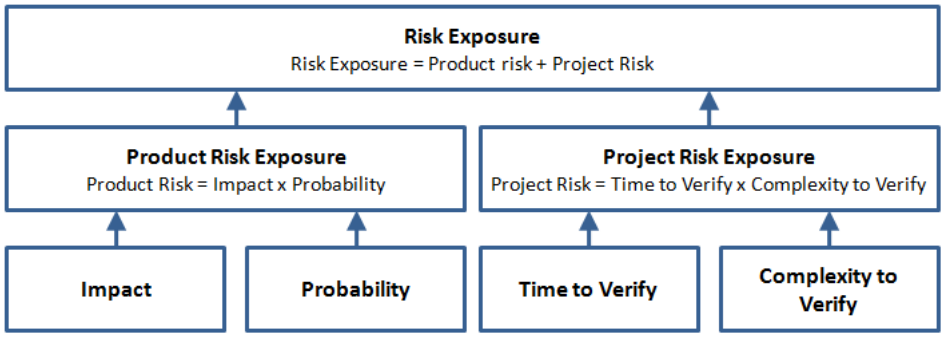


Figure 2.10: The Risk Based Verification Estimation illustrated [13].

Verification Planning

In this phase the investigation is carried out on the different kind of configurations applicable for the new function/feature. The configurations where the function will have the most influence are identified and selected for testing and verification. This work is important since the system exists in different configurations. It is meaningless

to test a configuration which is not affected by the new function. Therefore, it is crucial to find the right configurations so the testing can be as efficient as possible.

2.2.1.2 Execution

The most of the work by the I&V is carried out in the Verify Product in System Environment sub phase, number 2 in Figure 2.8. The four associated sub-phases are: System Impact Analysis for Verification, SW Build & Smoke Test, System Verification Preparation and System Verification.

System Impact Analysis for Verification
The System Impact Analysis for Verification phase is carried out together with the Risk Based Verification Analysis, see number 2 in Figure 2.9. This is the most important phase. Features are broken down into functions to be able to do an analysis of the impact on the system by the new feature. They do this together with Design and System department. The result is a verification plan for the system.

SW Build & Smoke Test
In this phase are the different components integrated into the first software build. A smoke test is performed on this build. The purpose of the smoke test is to discover any major faults in the software which will stop the basic functionality of the system e.g. system start, system stop and load system software. The basic functionality of the system has to work before the system verification begins.

System Verification Preparation
This sub phase is further divided into several sub phases, see Figure 2.11. During the first sub-phase an already existing test specification is either updated or a new one is created, see number 1 in Figure 2.11. From this test specification it is possible to do research about the testing tools needed for the new product, see number 2 in Figure 2.11. If there are any special tools that are required for the verification of the product it is important to order them in advance.

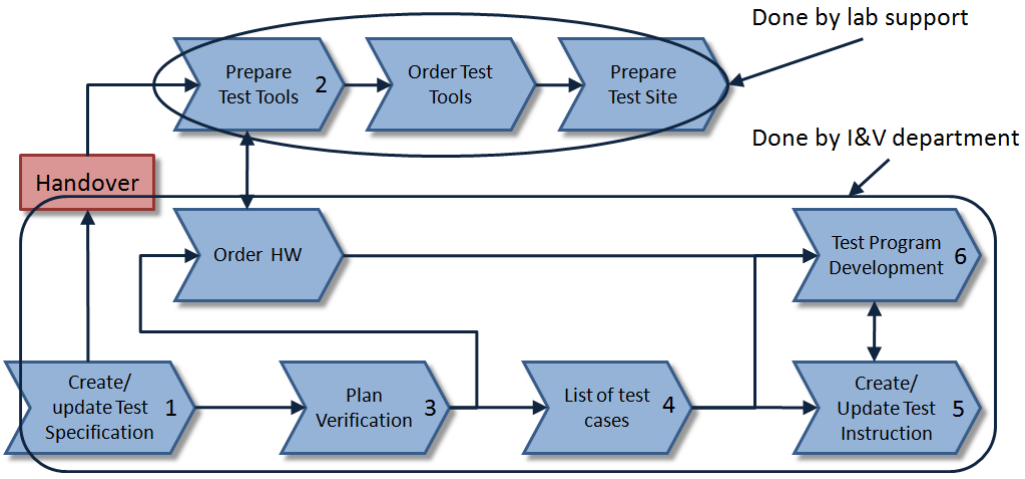


Figure 2.11: The System Verification Preparation phase.

When the test specification is approved the I&V start to plan how they should carry on the verification, see number 3 in Figure 2.11. The integration plan developed by the development team is also taken into consideration for developing a verification plan. The verification plan answers the questions: ”what to test?” and ”who will do it?” This work in addition to the coordination with other related teams is carried out by a test coordinator.

When the verification plan is approved the different tasks are given to the team leaders. The Team Leader has a number of testers, each having his/her own test suite, in which they list which test cases to run from the test program, see number 4 in Figure 2.11. In case the test case does not exist or need to be updated, the tester has to create/update it, see number 5 and 6 in Figure 2.11.

After this sub phase comes the System Verification phase, see Figure 2.12.

System Verification

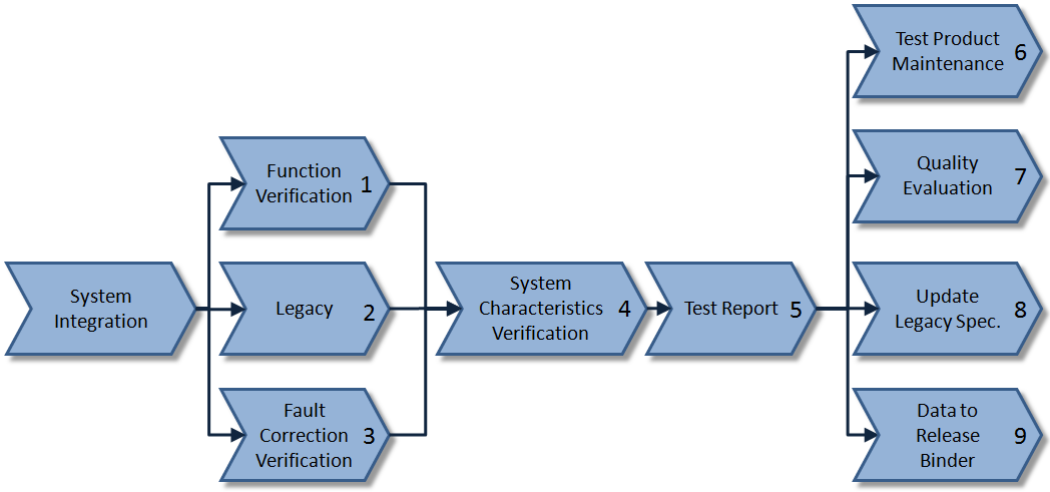


Figure 2.12: The System Verification phase.

The execution of test cases is done in the System Verification phase. When introducing a new function/feature there are some things that always need to be tested. These are tested in the Function Verification, see number 1 in Figure 2.12. All new and changed functions have to be tested (see red lines in Figure 2.13), to ensure that they work as expected.

The legacy functions are tested, see number 2 in Figure 2.12. Legacy is the old functions and functionality in the system that shall not have been affected by the new functions/functionality (the green lines in Figure 2.13)

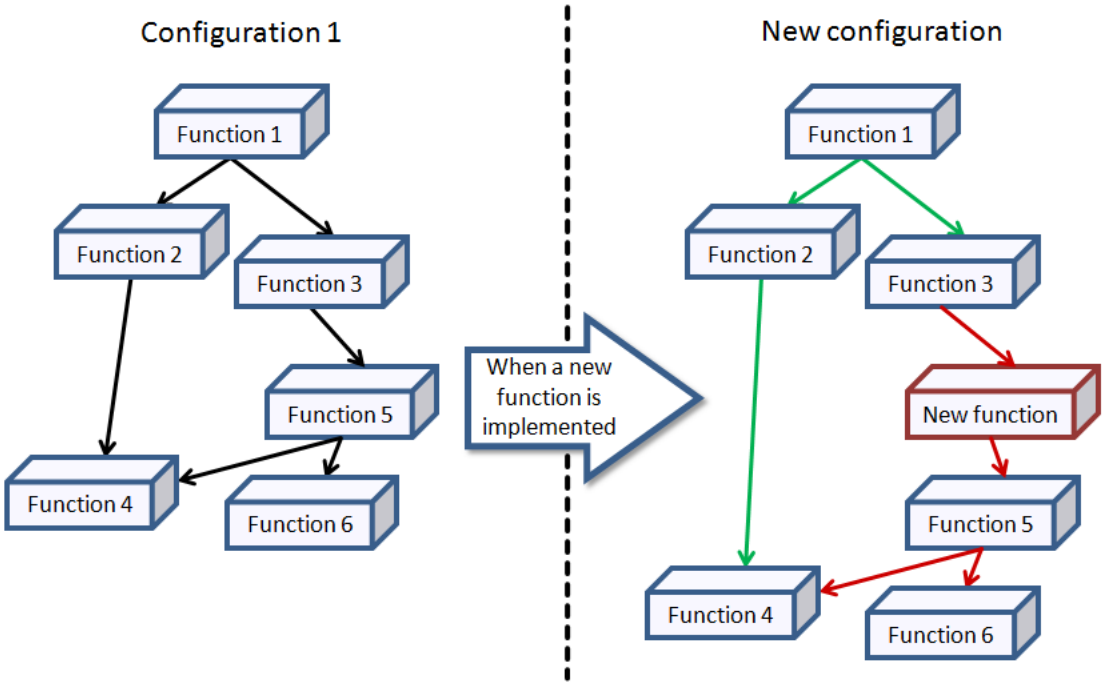


Figure 2.13: How a new function can affect the other functions.

The Fault Correction Verification are external defect reports, which are usally written by a customer, see number 3 in Figure 2.12. It is essential to test these faults and verify that they have been taken care of.

The systems characteristics needs to be tested, this is performed by a load test, see number 4 in Figure 2.12. A load test is a test where the system is exposed to a higher workload than normal, it could be near breaking point. This is done to see the system behaviours under pressure.

In a test report are all results from the testing, the conclusion and analysis contained, see number 5 in Figure 2.12. The testing tools and script developed during the project are saved in the Test Product Maintenance phase, see number 6 in Figure 2.12. These scripts and tools are saved for reuse later by maintenance teams. The Quality Evaluation is trying to answer how good the product is, see number 7 in Figure 2.12. The legacy specification needs to be updated since the new function/functionality will become old function/functionality in the next project, see number 8 in Figure 2.12. Release Binder contains the information about the product

being delivered, see number 9 in Figure 2.12.

2.2.1.3 Conclusion

After the execution is over the project is evaluated to see if there is something they could have done better.

In this phase of the project the verification of re-deliveries are handeled. These re-deliveries are software releases with corrected faults.

At the end, the new software is introduced and tested at the First Customer Site (FCS). The introduction is divided in three parts: lab, customer lab and live. At the first stage the testing is done in the company's lab environment. The second stage is testing in the customer's lab. The last stage is testing in the real network *"live"*.

Chapter 3

Data collection

The faults which are interesting for this study are the faults which appear after the testing is finished. These faults are termed as Fault Slip Through (FST). These faults may have some specific characteristics which makes them either impossible to be tested or deliberately untested by the testers.

3.1 Fault slip through

The definition of FST as used in the thesis is shown in Figure 3.1.

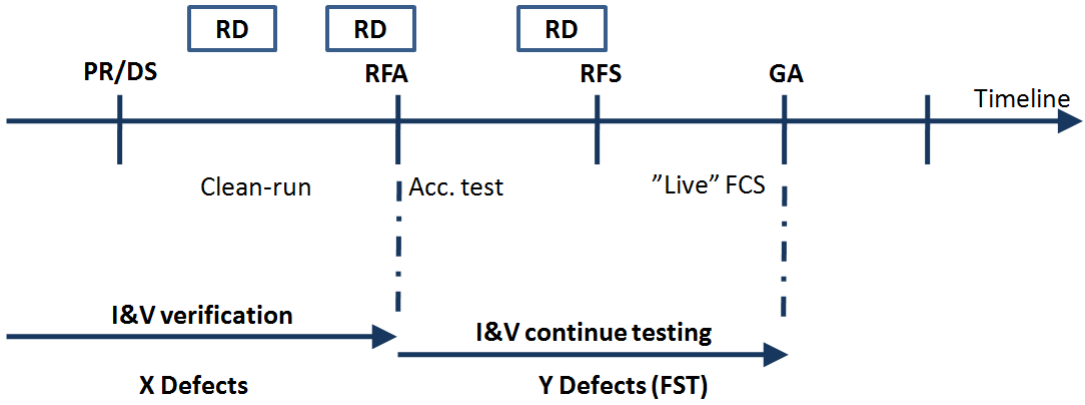


Figure 3.1: Definition of a fault slip through.

The abbreviations in the figure are as follows:

- PR – Product Release for pre-series production.
- DS – Design Status.
- RFA – Ready For Acceptance.
- RD – Re-design.
- RFS – Ready For Service.
- GA – General Availability. The product is ready for production, if the result from the FCS is successful.

As shown in Figure 3.1, the faults found before RFA (X Defects) are not counted as FST even if they are found very late in the process. Every fault after RFA is considered as FST (Y Defects). Like the Verification manager stated *"A fault found by I&V after RFA is just as serious as if the customer had found it, since there should not be any faults present"*.

3.2 Defect reports

A Defect Report (DR) is written when the tester have encountered an error while testing. The DR contains information about the system configuration and external tools used during the test execution. In an early phase of the project the integration leader handles the submitted DR's. But later into the project the DR's become more critical and are investigated more thoroughly. There is a possibility that a correction of a DR can lead to further DR's. Therefore, it is of importance to investigate the DR and analyze what parts of the system the solution will affect. This is the responsibility of a test coordinator. Each DR, which is generated in the later stages of the project, is discussed between the design, development and the I&V teams. A decision such as how to handle the fault or even holding of the new feature till next project is made.

3.3 Project selection and data search criteria

The data collected in this master thesis comes from five different projects. These projects are all related to the SUC. The reason why these projects have been selected for analysis are that all these projects are completed and a final report have been written for each project. The second reason is that these projects have followed the development process discussed earlier in this report. Another reason for selecting these projects is that they handle both hardware and software as compared to others, which are either focused on only hardware or only software.

A database for DR's has been used. The database is searched in exactly the same way for all the projects. The DR's analyzed have been filtered by submitted date and submitter ID. The date has been set to the RFA[1] date for each project and the submitter ID has been compared with a list of the IDs of the current staff at the department. This has been made to acquire and limit the number of DR's submitted after RFA by the I&V department.

For every finished project, a final report is written. These reports have a section containing information about the submitted DR's. This information has been compiled and used to compare the result from the database searches. An additional available information about the submitted DR's is also used for comparison. However, the primary data to be analyzed is the data acquired from the database search. The result from the database search is presented in Section 4.1

[1]The testing is finished and the software is ready for acceptance.

Chapter 4

Results, analysis and conclusion

4.1 Result

The data collected from the five selected projects is presented in Table 4.1. The three different columns represent the following three different sources from where the information is acquired:

- DB - Database: This is the company's database for DR's.
- FR - Final Report: The FR column represents the data acquired from the final reports for each project.
- TS - Test Specification: A TS is a paper with compiled information about the submitted DR's.

Table 4.1: DR's from the five projects.

Project A

	DB	FR	TS
All DR's (X)	288	236	-
I&V submitted	63	62	-
After RFA (Y')	84	6	-
I&V related (Y)	28	-	-
$FST'(\frac{Y'}{X}) * 100$	29.2	2.5	-
$FST(\frac{Y}{X}) * 100$	9.7	-	-

Project B

	DB	FR	TS
All DR's (X)	203	186	-
I&V submitted	35	41	-
After RFA (Y')	39	11	-
I&V related (Y)	8	-	-
$FST'(\frac{Y'}{X}) * 100$	19.2	5.9	-
$FST(\frac{Y}{X}) * 100$	3.9	-	-

Project C

	DB	FR	TS
All DR's (X)	176	158	-
I&V submitted	34	74	122
After RFA (Y')	6	0	-
I&V related (Y)	1	-	3
$FST'(\frac{Y'}{X}) * 100$	3.4	0	-
$FST(\frac{Y}{X}) * 100$	0.6	-	1.7

Project D

	DB	FR	TS
All DR's (X)	206	206	-
I&V submitted	36	70	113
After RFA (Y')	6	4	-
I&V related (Y)	1	-	1
$FST'(\frac{Y'}{X}) * 100$	2.9	1.9	-
$FST(\frac{Y}{X}) * 100$	0.5	-	0.8

Project E

	DB	FR	TS
All DR's (X)	111	129	-
I&V submitted	23	42	111
After RFA (Y')	15	12	-
I&V related (Y)	5	-	2
$FST'(\frac{Y'}{X}) * 100$	13.5	9.3	-
$FST(\frac{Y}{X}) * 100$	4.5	-	1.8

A summary of the acquired information is shown in Table 4.2.

Table 4.2: Summary for all five projects.

All projects

	DB
All DR's (X)	984
I&V submitted	191
After RFA (Y')	150
I&V related (Y)	43
$FST'(\frac{Y'}{X}) * 100$	15.2
$FST(\frac{Y}{X}) * 100$	4.4

The percentage of FST for each project is used as a measurement to evaluate the quality of the existing test method. The graph presents the projects total FST percentage and the percentage of FST for the I&V, see Figure 4.1.

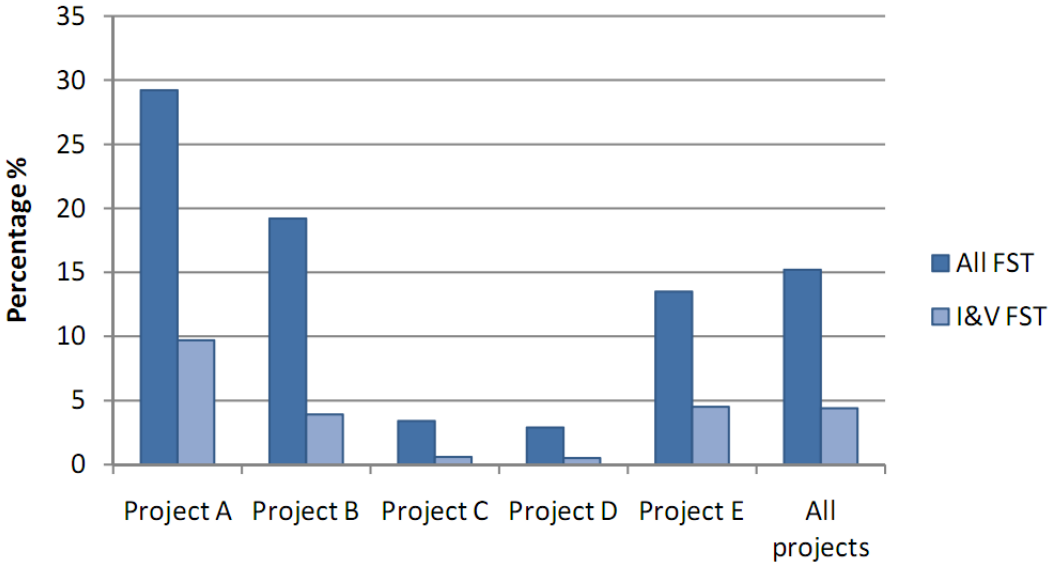
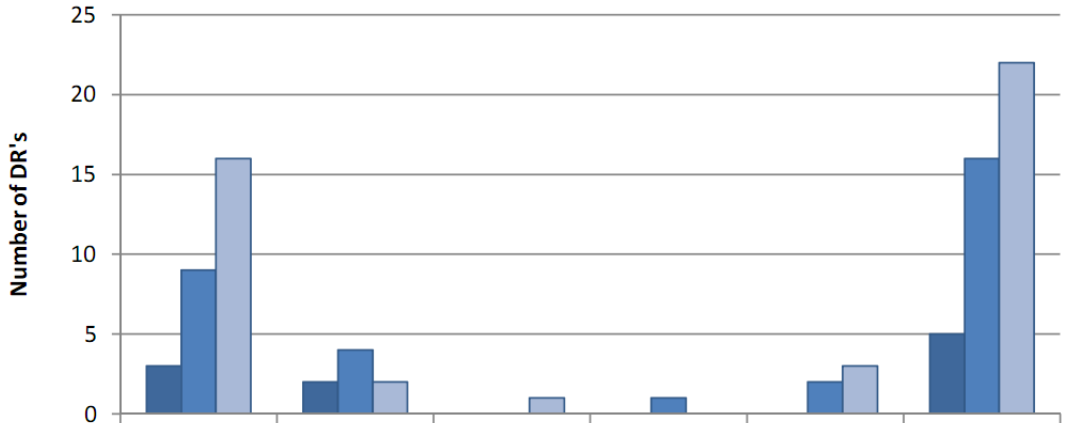


Figure 4.1: The graph shows the percentage of fault slip through for each project and the fault slip through percentage for I&V.

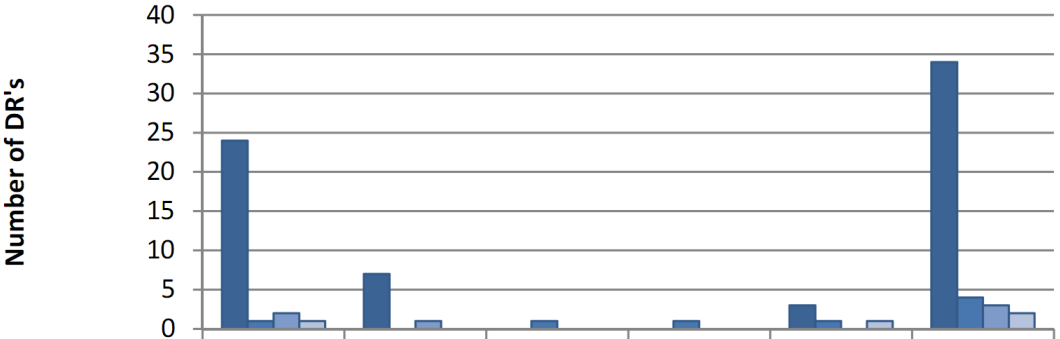
The DR's are divided into three different categories: Critical, Major and Minor. A critical fault will stop the software from functioning correctly. A major fault will stop some of the functions, but the basic functionality of the software will work. Minor faults are not so critical. These faults can be considered in the next project. The graph shows the different projects and the number of FST in each category, see Figure 4.2.



	Project A	Project B	Project C	Project D	Project E	Total
Critical	3	2	0	0	0	5
Major	9	4	0	1	2	16
Minor	16	2	1	0	3	22

Figure 4.2: The graph are showing the number of fault slip through sorted by fault class for each project.

The DR's that have been submitted after RFA are four different types: software fault, fault in tools, other and unknown. The graph shows what type of FST that have been found for each project, see Figure 4.3. A software fault refers to the software in the system and faults in tools refer to software faults in the tools. Other faults can be things in the documentation and unknown faults are faults that were not categorised.



	Project A	Project B	Project C	Project D	Project E	Total
SW-fault	24	7	0	0	3	34
Fault in tools	1	0	1	1	1	4
Other	2	1	0	0	0	3
Unknown	1	0	0	0	1	2

Figure 4.3: The graph are showing the number of fault slip through divided into fault type for each project.

The work in this thesis has shown different faults from different parts of the system. There are two areas of the system where there have been more DR's submitted from than any other part of the system. One of the areas is related to the product that the customer gets with the system, which is an administrative tool. The other area is related to the System Configuration Database (SCD). A brief description of the administrator tool and the SCD:

- **Administrator tool**
  The administrator tool is used to see the status of the system and to configure the systems configurations database.

- **System configuration database**
  The system configuration database contains information about the current configuration of the system.

The following graph shows the number of FST that are related to these two areas, see Figure 4.4.

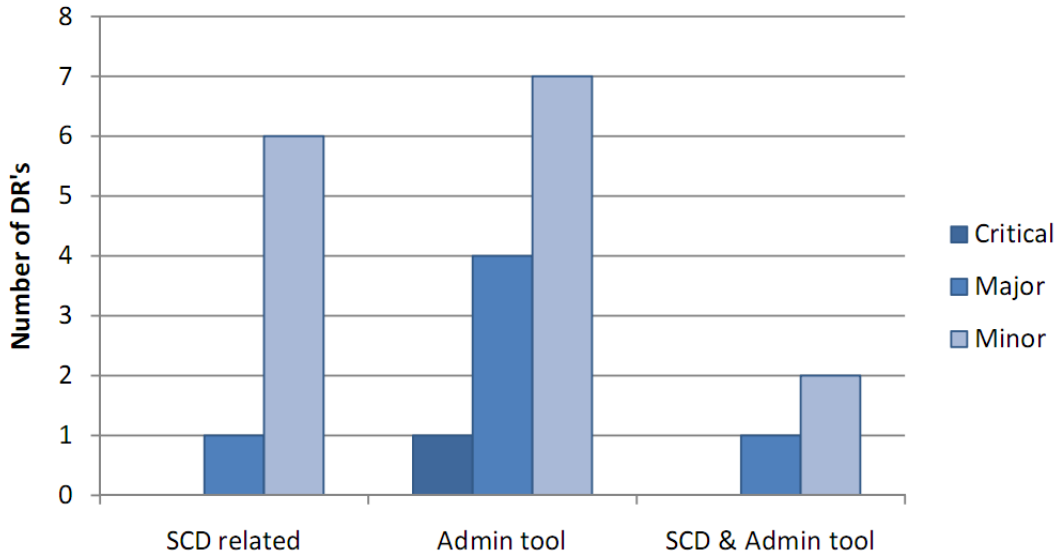


Figure 4.4: The graph are showing the number of defect's related to system configuration database and the administrator tool.

4.2 Analysis and conclusion

The total number of DR's submitted for all projects are 984 and the total number of DR's submitted after RFA is 150. In case of I&V, 191 DR's are submitted during verification and 43 of those are submitted after RFA. That results in an overall FST percentage of 4.4 for the I&V department, see Table 4.2. The company has a goal to keep it below 5 percent. This goal is met for all projects except project A, see Table 4.1. This result shows that the most faults in the system are found during the scheduled testing and verification phases. Accordingly to the findings in this report approximately 84.8 percent of all faults are found with the current test method, see Table 4.2.

As mentioned earlier, the total number of DR's submitted after RFA are 43. The faults found are of four different types, see Figure 4.3. The figure shows that 34 faults are related to the software in the system, it corresponds to 79.1 percent of all FST's submitted.

As seen in Figure 4.4 are the total number of DR's related to the administrator tool and the SCD are 22. This represents 51.2 percent of the total number of FST DR's. These faults are divided into 15 minor faults, 6 major faults and 1 critical fault. The administrator tool interacts with the SCD.

The administrator tool is a separate product and it is supposed to support many different configurations of the system. This can be the reason why there are so many faults found related to it. The complexity of the software in the administrator tool is very high due to the fact that it has to support many different systems.

The company's testing method is good and they find the most faults. However, the company should review the part of the testing method that handles the SCD and the administrator tool as the majority of the faults to this part of the system.

To answer the questions from the introduction, see section 1.1.1

1. Due to the competition in the industry it was difficult to acquire information about the test methods begin used by other companies. Therefore, have only the information from the subject company been investigated. However, a good assumption is that the testing is done in similar way.

   The company's testers are ISTQB[1] certified [14] and the test method the company uses complies to the ISTQB standard. However, the test method is modified to suite the company's way of working. The test method at the company is a mixture of structure based testing, specification based testing and experienced testing. At the investigated company the emphasis is on experienced testing.

2. Most of the faults are discovered during the scheduled testing, which corresponds to 84.8 percent of the faults discovered during this phase.

3. Most of the faults are related to the SCD and the administrator tool as compared to any other part of the system.

4. Due to time limitation the last question has been suggested as further work, see section 4.3.

4.2.1 Uncertainty with the collected data

The information about these projects has been acquired through searches in the company's database. The searches have been carried out in exactly the same for each project so that is becomes possible to compare the outcome from each of them. The total number of DR's acquired from these searches is definitive. However, the number of DR's reported by the I&V department is harder to determine. This is because of the fact that it is not possible to filter the searches by the submitter's department / organization. The information that can be extracted from the search result are the full name and the user ID of the submitter. With this information it is only possible to compare the submitter's user ID with the user ID's from the list of staff currently employed at the I&V department. However, due to the fact that some of the people who worked in these projects are not present today, the total number of submitted DR's from I&V is lower than the actual number. To reduce the impact of this uncertainty have the list of submitters is analyzed by the company.

The reliability of the data collected from the final reports for the selected projects should be good. However, the total number of DR's for each project's differs from the number of DR's from the searches. An investigation to find the cause to this difference where made. It turned out that the reports are inconsistent when gathering the DR data. Therefore, the total number of DR's reported in the final reports is not so reliable. Another problem with the final reports is that the DR number is not presented anywhere. This makes it impossible to analyze these DR's. The final

[1] International Software Testing Qualifications Board

reports contain information about DR's submitted from different departments and DR's submitted during FCS verification. These numbers are most likely affected by the inconsistency of the total number of DR's. However, the numbers presented from the final reports gives a hint for what the real number could be. The numbers in the final reports are management corrected which means that some of the DR's that appear late are not accounted for, as a FST. The reason can be either that the management is usually aware of the problem and it decides to fix it in a re-delivery or the DR is not considered as a fault at all.

Statistics about the DR's for these projects have also been acquired from a third source. This source does not contain information about all the projects. Specifically, project A and project B are not present in this source. This information comes from a document from the I&V department and it presents DR's written by them before and after RFA. These numbers has probably been management corrected as well and how these numbers have been produced is unclear.

4.3 Further work

It is recommended that the company do further investigations and that these investigations are focused on the following two areas:

- The system configuration database of the system. How are the requirements written? How is it created? How is it tested and verified?
- The administrator tool. How complex is this system? How is it verified? How critical is a fault related to it?

This study could also be applied in another industry segment. It could also be carried out at another company within this industry. The study could also be made at the design department at the studied company.

Bibliography

[1] Chris C. Schotanus. *"TestFrame an Approach to Structured Testing"*. Logica, 2009.

[2] Sigrid Eldh. *"On Evaluating Test Techniques in an Industrial Setting"*. Lic thesis, Mälardalens Universitet, Arkitektkopia Västerås, Sweden, 2007.

[3] Paul C. Jorgensen. *"Software Testing - A Craftsman's Approach"*. CRC Press, 2002.

[4] Rick D. Craig, Stefan P. Jaskiel. *"Systematic Software Testing"*. Artech House Publishers, 2002.

[5] British Computer Society. *"Standard for Software Component Testing"*. tech. rep., British Computer Society Specialist Interest Group in Software Testing, 2001

[6] Steve Cornett. *"Code Coverage Analysis"*. Webpage, March 2010. `http://www.bullseye.com/coverage.html`.

[7] Ofri Wechsler. *"Inside Intel Core Microarchitecture, Setting New Standards for Energy-Efficient Performance"*. Intel Corporation, 2006.

[8] IEEE. *"IEEE Standard for Software and System Test Documentation"*. tech. rep., IEEE, 2008.

[9] IEEE. *"IEEE Standard for Software Unit Testing"*. tech. rep., IEEE, 1986.

[10] Leonard Tripp, Peter Voldner. *"JTC1/SC7 Software & System Engineering"*. Webpage, March 2010. `http://www.jtc1-sc7.org/`.

[11] ISO/IEC JTC1/SC7 Working Group 26. *"ISO/IEC 29119 Software Testing, The New International Software Testing Standard"*. Webpage, March 2010. `http://softwaretestingstandard.org/`.

[12] British Computer Society. *"Glossary of Terms Used in Software Testing"*. tech. rep., British Computer Society Specialist Interest Group in Software Testing.

[13] Yanping Chen. *"Specification-Based Regression Testing Measurement With Risk Analysis"*. Master thesis, University of Ottawa, Ottawa Ontario, Canada, 2002.

[14] International Software Testing Qualifications Board. *"Home - ISTQB"*. Webpage, April 2010. http://istqb.dedicated.adaptavist.com/display/ISTQB/Home.