Master Thesis
Software Engineering
Thesis no: MSE-2007:08
January-2007

Evolving test-case selection at a large scale company

Samir Drincic, Asim Dedic

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 40 weeks of full time studies.

Contact Information:
Author(s):
Samir Drincic
E-mail: samir.drincic@gmail.com
Asim Dedic
E-mail: asim.dedic@gmail.com

External advisor(s):
Kennet Henningsson
UIQ Technology AB
Address: UIQ Technology, Soft Center VII. SE – 372 25 Ronneby, Sweden
Phone: +46 457 464700

University advisor(s):
Håkan Grahn
Department of System and Software Engineering / Blekinge Institute of Technology
Phone: +46 457 385804

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/tek
Phone : +46 457 38 50 00
Fax : + 46 457 271 25

ABSTRACT

This paper presents a possible solution for selection of test-cases that should be executed to provide good coverage and quality at UIQ Technologies. It describes how the company can sort and prioritize their test-cases, so that quality is maintained while having controlled amount of test-cases. The idea behind the proposed method is to prioritize the test-cases and execute those that received highest prioritization first and thereby ensuring that all high-risk defects are found first. We have created a model that performs those tasks and we have also executed it on an ongoing project at UIQ Technology to provide data for comparison with the currently used model. We also make a comparison proving that our proposed method is more effective then the current method. We have emphasized adaptability and changeability of the model so that UIQ Technology easily can modify and adapt the model later on.

Keywords: test, regression, improvement, large scale company

CONTENTS

1 INTRODUCTION

Testers involved in testing large software systems are daily faced with the, amongst others, problem of limited testing-time and resources. As those systems expand in both size and complexity so do the test suites for those systems [19, 25]. The result of those circumstances is that in certain situations some test-cases or test-items are omitted [1] e.g. if a system is containing a lot of test-cases while having short delivery time. There exists a lot of research in this area, but most of the methods are concentrating on covering as much as possible of the source code [8, 21, 22,24]. This approach does give a good overview of how much of the source code that has been tested but is unfortunately not applicable when the systems, that are the subject of the testing, are large.

Lately, research about function testing has been conducted and written in reports [1, 2]. The idea of function testing is that most functionality of the software is, by the user seen, as black-box functions [19]. Testing the behavior of those functions will satisfy the customers according to Bezier [19]. Even though this kind of testing is easier to conduct, the size of the test suite will sometimes get too big to cover. It is here Baser introduce the prioritization of test-cases. A lot of work has been done in this area as well [7, 12, 13, 17, 20, 21, 22, 24]. Main idea of this approach is to identify high risk/priority faults and execute test-cases that reveal those, to satisfy customer expectations on the software [1]. This approach can be used from several different viewpoints. Meeting customer expectations is e.g. user point of view ensuring that crucial functionalities for the user are fulfilled and product ready for release.

Another large part of software maintenance is regression testing. Regression testing is performed on new or modified software to ensure that *"the changed parts of software behave as intended and that the unchanged parts of the software have not been adversely affected by the modification [33]"*. There are two basic approaches to do regression testing, one being the so-called retest all approach and the other being selective regression testing approach. The retest all approach is the safest approach to choose but this approach will in many cases lead to exceeded resources. Looking at the selective regression testing techniques it is quickly noted that most of the techniques work on white-box level i.e. studying the code and comparing changes. Few of the methods mentioned in reports require an in-depth-analysis of the source code [3, 13, 31, 32]. However after further research we managed to find work conducted by Chen, Probert and Sims [4] where they apply prioritization on a black-box(specification based) level to select test-cases for regression testing.

The solution proposed in this thesis is based on a selective testing approach. To achieve selective testing the method we propose, uses prioritization of test-cases. This proposed method was matched against UIQ Technology[1] current method for testing and was proven more efficient. Some of the evaluation points were number of test-cases executed and also severity of failures detected by executed test-cases. It is our opinion that the results of the method were satisfying but we feel that the method can be improved by further work. Further information about methodology used in this thesis can be found in sub-chapter 1.3.

[1] www.uiq.com

1.1 Aims and objectives

The aim of this thesis is to try and find a suitable method to help UIQ Technology improve their testing selection process. We aim at fulfilling following objectives

- Identify different methods for decreasing amount of testing
- Find a method that is acceptable for UIQ Technology
- Customize the found method so that it can be applied to UIQ Technology data
- Virtually execute this method on well documented UIQ Technology project
- Identify strengths and weaknesses of the proposed method

1.2 Research questions

To be able to conduct thesis in a structured way following research question were created together with Kennet Henningsson[2] and Cecilia Wester[3] both employees at UIQ Technology:

1. Is there a process that could be suited to fit the company's regression testing?
    a. What amount of test-cases that should be rerun would this process result in compared to the amount of test-cases rerun today?
    b. How will the final quality of the product be, compared to the quality of the product as it is today?
2. Is there any way to do initial[4] test-case selection so it decreases number of test-cases run while maintaining same quality?
    a. How would this work practically?
    b. Is this method feasible to use for a company like UIQ Technology?
    c. What are the advantages and disadvantages of such process?

The main purpose of those questions is to keep the research on track and to provide us with some goals to strive towards. In other words, we do not seek to answer those questions with 100% accurate answers, but rather use them as guidelines during our thesis research.

1.3 Methodology

This section describes the different approaches carried out while doing the thesis at UIQ Technology. It also provides some information about why the certain methodology is used for the study. It contains qualitative research approaches, quantitative research approaches and a case-study that is carried out.

1.3.1 Qualitative study

To gain understanding of the area of software testing and software regression testing a literature study is conducted. Study journals, books and peer-reviewed reports are used to provide us with knowledge about the area. The searched topics include

[2] Kennet Henningsson is currently employed at Development System Management Department at UIQ Technology
[3] Cecilia Wester is currently employed as Test Leader at UIQ Technology
[4] The concept behind expression ”initial” is thoroughly explained in section 2.4.1

testing in general as well as regression testing. To be sure, to detect a possible method suiting UIQ Technology all found methods are assessed for possible validity to be the solution.

In order to, further improve the understanding of the problem definition posed on the thesis, interviewing of UIQ Technology employees is conducted. Two of the company employees are selected as interview targets. The reasoning behind selecting the two employees is their knowledge of testing process at UIQ Technology as well as their current positions at the company. Regarding more interview targets this option was not possible at the company at the time of thesis were in final phase of a major release, they had no labor resources to provide to us. However, professional profiles of the persons that were made available to us convinced us about their expertise in the area. In order to gain knowledge of UIQ Technology testing process, tools and different issues a case-study of company documentation and tools is conducted. In depth, research on functional specification trace database is conducted to ensure knowledge about company testing process, notation of test results and test specifications. Unstructured interviews are conducted to gain knowledge about database used for storage of test-cases and their results.

1.3.2 Quantitative study

To prove usefulness of proposed method a quantitative approach is used. An extensive experiment is conducted using the proposed method on a UIQ Technology ongoing project. Data about current testing method at the company is compared to data from our proposed method so that it is measurable what the benefits and drawbacks of the proposed method as well as the current method are.

The results gained from the quantitative study are used in a direct comparison with current testing method at the company to provide valuable indication of the potential of the method.

1.4 Overview

Chapter 2 of the thesis covers briefly UIQ Technology information and the issues with the company test-process Section 3 covers common testing methods and different aspects of testing e.g. initial and regression testing. In section 4 we present the two methods for initial and regression testing that our final method is based on. In section 5 we analyze both the current and also our proposed method and we also present advantages and disadvantages of both methods. Section 6 covers execution of our proposed method on UIQ test-cases together with analysis of performance of the method.

2 COMPANY INFORMATION

In this chapter, UIQ Technology is described as well as their product. After the company has been introduced, an overview of the company testing process is provided together with the issues that were presented to us. We also present some UIQ Technology specific expressions, tool and some statements about further introduction of tools, models and preferred solutions to the issues handles on the thesis.

2.1 UIQ Technology

UIQ Technology is a software development company, which develops an open software platform that is licensed to leading mobile phone manufacturers. The platform is customizable so that it can fit several types of mobile phones, with very little change in the source code. This is a great way to reduce the development time for mobile phone software, but also the time-to-market for mentioned software. UIQ Technology is developing their products on top of the Symbian OS. They also face rather complicated requirements on their software since they are developing products for mobile phones. Some of the requirements are e.g. close to 100% uptime, that most mobile phones are supposed to have and great scalability because the developed software needs to function on several different phone types. Since the software is developed for handheld devices, they also face requirement of adapting the software for limited resources e.g. limited memory and limited computing capability. This leads to the fact that UIQ Technology needs to thoroughly test their software so that they fulfill those requirements.

2.2 Testing at UIQ Technology

Together with Kennet Henningsson and Cecilia Wester, decision was taken to research and try to improve UIQ Technology testing process. They both work at the company, and they presented the problem issues during our first discussions. First, the initial selection of test-cases that should be tested was done with very loose guidelines. There is no structured way of selecting test-cases that should be tested which leads to in some cases unnecessary testing. This loose way of selecting test-cases to test also leads to the fact that less important test-cases might take testing time away from more important test-cases. The test-leaders are responsible for determining when the testing was finished. For this, we were told, expertise and previous knowledge by the test leaders is used.

This is the main problem area, UIQ Technology has no definite way of telling if the testing conducted is enough to guarantee high-enough quality, so it is their gut-feeling and intuition that determines when to stop testing. More about this problem and solutions we proposed can be found in sections 5 and 6. Furthermore, to overcome this problem, they perform a basic retest-all approach. This leads to the fact that they in the end test too much, exceeding both time and resources in some cases.

The second problem that was proposed for this thesis, to solve, was the regression testing-phase at the company. Similar to the initial testing, the regression testing was done without any structured approach and regression testing is seen as finished when no more defects are found with the selected test-cases. Also selection of the test-cases that should be a subject of regression testing is once again done with loose guidelines and UIQ Technology are looking for more structured way to select test-cases also here.

2.2.1 Abstraction level of the thesis in respect to testing

It was clearly stated during early discussion that the thesis should aim at researching the, per definition [34], functional/acceptance testing. Acceptance/functional tests aim at satisfying customer expectation [34] which is something taken into consideration in the thesis while working on a solution. Other testing e.g. unit testing is not taken into consideration in the thesis and is not used in any kind of evaluation and/or measurement.

As we previously wrote, the aim of the thesis is at the acceptance/functional level of testing. However, the UI test-cases are not taken into consideration. Main reason behind this is the ongoing effort to automate the testing of the UI currently taking place at UIQ Technology.

2.3 Motivation

The main motivation for this thesis is the UIQ Technology desire to improve their testing process and especially the test-case selection part. As it is, later in the thesis written, a rather large amount of test-cases is present in every project. The project that was chosen for experimenting in this thesis can be seen as rather small with only around 4000 test-cases but we would like to remind that this project was ongoing while the thesis was executed. The amount of test-cases was base lined to a certain point and will grow further as the project moves on. Combined with the fact that multiple test runs are conducted the amount of test-cases rapidly increases as they get further into the projects. Even though the company is undergoing process of automation in some areas, there is an expressed need to a method that will reduce amount of testing with maintained quality, a method that is not a full automation of testing since the company already is working on some automation for other areas. UIQ Technology is also constantly undergoing a procedure of improvement of the documentation, which in turn leads to more possibilities for conducting methods that will reduce amount of testing.

2.4 UIQ Technology specific

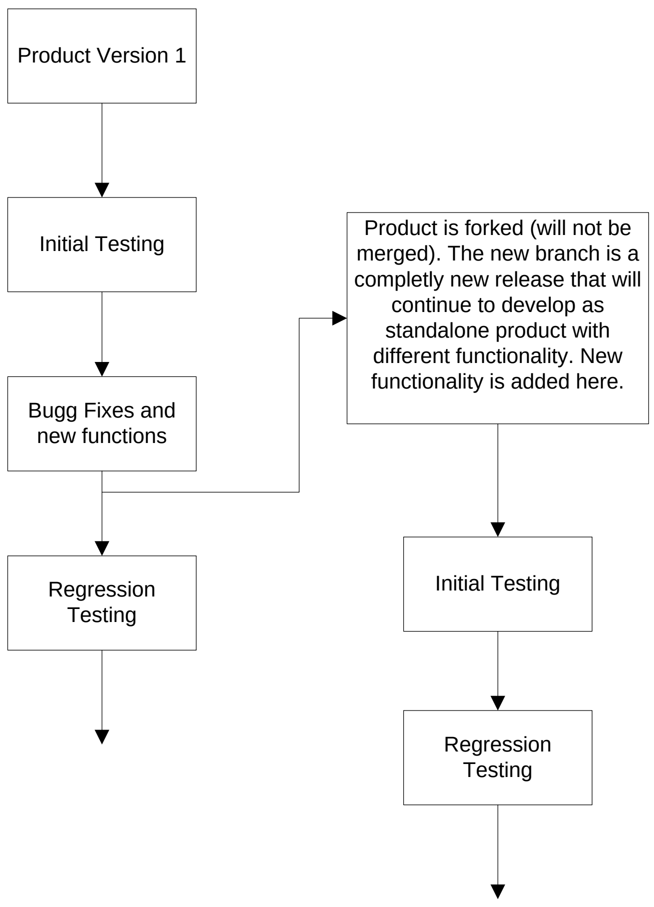
In this chapter, some special expressions for UIQ Technology and this thesis are explained. It also holds explanations to what the company desires in form of methods, tools and solutions for the above-mentioned issues. Reason for explanation is that those objects that are special for UIQ Technology are crucial to have knowledge about in order to fully understand the thesis.

2.4.1 Initial testing

The concept of the expression “initial testing” is very hard to understand. Therefore, this section is dedicated to explaining the concept of the expression. This expression is used in the thesis and needs full understanding in order to understand the thesis.

First, what initial testing does not mean is the process of creating test-cases. The expression be misinterpreted as the process of test-case generation that, in this case, we are not talking about. The meaning of the expression “initial testing” in this thesis is the first testing done after development. To strengthen the meaning of “initial testing” even more, we refer to Figure 1 below.

Figure 1: Initial testing



Product Version 1
Initial Testing
Bugg Fixes and new functions
Regression Testing
Product is forked (will not be merged). The new branch is a completly new release that will continue to develop as standalone product with different functionality. New functionality is added here.
Initial Testing
Regression Testing

Picture 1 adds more explanation to the meaning of the expression “initial testing”. What is happening in the diagram is that an initial version of the program is created followed by first “initial testing”. As it is previously mentioned, it is the first testing done after development. Beyond this point bug fixes and regression testing are tasks that are performed. Initial testing however is used again only if, as in the picture, software is forked and very new version is created. In the picture, it is stated that the newly created fork is not meant to merge with the original line later on and that new functionality is added. The meaning is that all the new functionality that is added will trigger a new “initial testing”. Same thing would happen if the company uses parallel development. A product A and a product B would then co-exist. At the start of both projects initial testing is done.

In this thesis expression, “initial testing” is used to describe first testing after development. As one part of the problem definition, the test-case handling at initial testing is considered in this thesis. In rest of the thesis, the expression “initial testing” will be used without explanation. The approach for the initial testing can however without modification be used for regression testing as well as we will describe more in detail later.

2.4.2 FS-Trace

Expression FS-Trace in this thesis refers to the functional specification trace that is used at UIQ Technology for tracking of functions currently embedded in a certain project. FS-Trace consists of a web-based client that communicates with a database holding all the information regarding the functions. The information that can be viewed can be e.g. test-cases coupled to a certain function and procedures to how to execute each one of the test-cases found in a function. The FS-Trace is, in the thesis, used for data collecting, sorting and comparing and is therefore a very important expression to understand. Another very important aspect of the FS-Trace is the tags used for notation of the functions. All the functions are written in following format *FS.someGlobalFunction.functionInsideGlobalFunction.subFunction* This approach to notation is later on used for batch making. It is written more about this later on in the thesis.

2.4.3 Standpoints

UIQ Technology has some clearly defined standpoints that this chapter describes. Reason for bringing those up is that the reader needs to be familiar with those standpoints to understand some of the decision taken in the thesis.

SP1. For the researched and later on proposed method there should be no tools requirement posed on the method. UIQ Technology currently develops most of their used tools by themselves and do not wish to incorporate any new tools into the system. Even though this standpoint the thesis does include some basic studying of tool-based methods, mostly for the writers own knowledge.

SP2. The researched and later on proposed method should have good potential for later automation. As we previously wrote UIQ Technology are already working on implementing automation so this desire was rather obvious. If the method proves successful, it should be possible to automate it in line with some other areas at company. To achieve high degree of automation is seen as one of the sub-goals for the thesis.

3 TESTING METHODS

This chapter describes the different testing methods that were researched while doing the literature study planned in the methodology of the thesis. The literature study covers both testing methods that handle the initial testing but also methods that handle regression testing. The most interesting methods found during the literature study are described and in the later subchapters the two methods that stood ground for the final solution proposal are thoroughly noted down.

3.1 Introduction

Software testing is big business nowadays; there is several different ways to approach a testing process so that it fits the testing needs of a certain company. How to choose which method suits your company can be as hard as development of the product itself. The basic principle of the initial testing is to verify the software against the requirements that the customer and the developing company agreed upon [2]. There are different methods on how to perform this phase of the product life cycle. The aim of all methods is to cut down on the number of the test-cases that is tested, but sustaining a good-enough quality on the end product. The literature study described in this chapter is aimed at gaining knowledge about the area at the same time as searching for possible candidate as solution for the issues presented for the thesis.

3.2 Selective testing

Since the use of software has increased over the last decade, so has the size of the software. This obviously leads to an increase in testing performed on the software products. The big software products contain a lot of functions, operations and behaviours, and all of them should be tested [19, 25]. However companies, similar to UIQ Technology, have close to ten thousand test-cases for their products. As a result there is no feasible way of testing all those test-cases without extensive budget and lead time impact.

This is where selective testing enters software business. The goal of this approach is to select test-cases that will give a good coverage of the product functionality in a way that will guarantee a “high-enough”-quality. The selective method exists in a lot of varieties, depending on what the company’s focus is. If the product is going to be used in real-time environments, e.g. airplanes or hospitals, then the focus lies towards reliability and durability. While in our case, working with UIQ Technology, the focus is more towards the visual appearance, and functionality. How the test-cases are selected is also very different from method to method. Some methods use discrete-mathematics, while others use test developers judgments to select what test-cases should be performed.

3.3 Initial selective testing

This section presents different methods that can be used during initial testing of a product for test-case selection. The concept of initial testing, is described earlier in the report.

3.3.1 Different methods for selective testing

The goal of the literature study for this section is to find method that could be used by UIQ Technology during their initial testing phase. The methods presented may not always be the best suited for UIQ Technology's current situation. Therefore the methods presented will be mapped against the standpoints stated in section 2.4.3.

3.3.1.1 Axiom-Based test case selection strategy

This method uses a research by Frankl et al [15] that presents an approach to testing of object-oriented programs. Frankl et al try to implement a way of testing entire classes using data abstraction. Their idea is that the classes are the natural units to tests, and that the testing should check if a sequence of data "puts an object of the class under test into the *correct* state". Frankl et al constructed a set of testing tools that was named ASTOOT (A Set of Tools for Object-Oriented Testing), which embodied the idea of classes as natural units to test. Frank et al believed that when designing test-cases to test certain method inside a class you shifted the focus away from data abstraction, the interaction between operations within a class.

The ASTOOT was designed only to handle algebraic specifications. The algebraic specifications are created by modelling a specification with heterogeneous word algebra. Example from [16] is used to make this transformation clearer. Let u_1 and u_2 be two terms of an Abstract Data Type (ADT). Further on, let s_1 and s_2 be respective sequence of operations to the terms u_1 and u_2. Frankl et al [15] state that two terms are equivalent if they can be transformed into one another using axioms as rewriting rules. If s_1 and s_2 produce different objects while u_1 and u_2 are equivalent, and error has been found. ASTOOT has tools that apply axioms transformation of u_1 to u_2. It then uses a driver generator to automatically derive test drivers that check and execute s_1 and s_2 corresponding to u_1 and u_2. The last step is comparing the results of the execution.

Chen et al [16] found some discrepancies with the Frankl et al suggestions. The major one was that transformations using axiom rules is uni-directional. This means that if u_2 that is derived from u_1 using axiom rules is not equivalent to u_1. Chen et al also state that Frankl et al use a selection technique that is based on two case-studies without any theoretical proof, and therefor cannot guarantee any proof of effectiveness. In order to improve the selection of Frankl et al method they define few rules that should be applied in order to have a sound mathematical proof of the method. Chen et al state that u_1 and u_2 are equivalent only if both of them can be rewritten on the same ground term (term without variables) and reach a unique normal form. A term is said to be in normal form if no axioms are applicable [16]. Compared to Frankl et al, Chen's method is based on theoretical facts and mathematical proof, which is stated to be more efficient in selection of tests.

The advantage of this method is its mathematical roots. It provides a strong qualitative measurement on what should be tested with mathematical proof. But the main advantage was also a disadvantage when the method was presented UIQ Technology. The method was too complicated and too academic according to them. The need of algebraic specification would cause a rewrite of their current specifications, which is something they did not have any intentions of doing. And as stated in 2.4.3 UIQ Technology was not interested in incorporation of any new tools.

3.3.1.2 Annotated classification trees

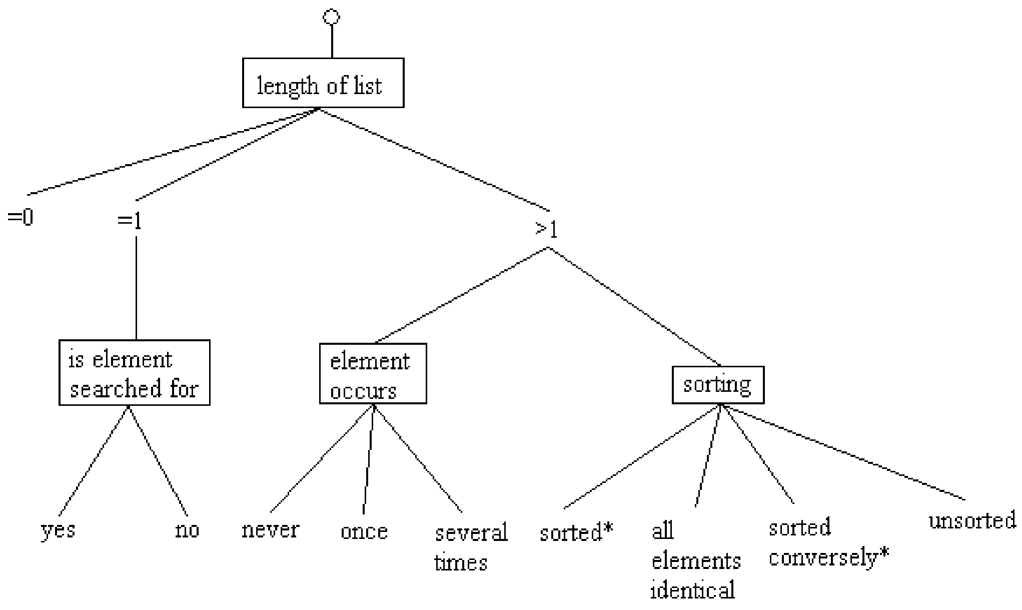
The method presented by Yu et al is based on evolving the classification-tree method (CTM) to support generation, selection and prioritization of test-cases. The CTM was introduced by Grotchmann and Grimm, and has been used in many industrial development projects. In order to describe the functionality of the method an example from Yu et al report will be used. See Table 1 below.

Table 1: Categories and choices [17]

Category (Classifications)	**Associated Choices (Classes)**
length of list	=0, =1, >1
is element searched for	yes, no
element occurs	never, once, several times
sorting	sorted*, all elements identical sorted conversely*, unsorted

A simple function that counts number of occurrences of a certain element in a list and then returns the amount is used as the function that is to be tested. Analysis of the function should show the categories (classifications) of said function. To each of the classifications there are Associated Choices (Classes) connected to. Test-cases are generated by combining the different categories. But some combinations are not feasible, e.g. if the length of the list is 1 the sorting does not apply. Yu et al state that when number of combinations is large the job of manually identifying unfeasible connections between categories is costly and error-prone. The next step is to specify constraints for all the associated choices. Generating the test-cases can be conducted with a tool that is designed for CTM specifications. Another way of generating test-cases is by drawing the classification tree and setting some rules that should be followed during the test-case generation.

Figure 2: Classification tree [17]



The advantages presented by Yu et al, focus on the visual aspects of the method. They state that the graphical appearance of the method makes it easy to understand and use. According to one of their sources inexperienced users to CTM, did not have any problems deriving test-cases from the classification-tree, even without a tool.

In order to apply test-cases selection and prioritization to the CTM, Yu et al adapt new annotations to the nodes of the classification tree. The three annotations added are selector expression, occurrence tag and weight tag.

Selector expression: this annotation incorporates selectiveness of what classifications can be combines, so that unfeasible classifications are not combined to generate malicious test-cases.

Occurrence tag: the reasoning behind this annotations is if the tester deems it necessary that a certain test-case is to run x amount of times in the test suite. The tester can then set the occurrence tag value.

Weight tag: the weight tag shows the priority of a certain classification, and can be used to sort classifications based on their priority.

After presenting the added annotations, Yu et al use five steps for the test preparation process using a tool. The tool can automatically generate legitimit test-cases due the added annotations. It also allows the tester to prioritize the test-cases and select test-cases to be run, both manually or by deciding the occurrence level of test-cases that should be run.

This method would give UIQ Technology a framework for generating, selecting and prioritizing their test suite. According to Yu et al, the framework is close to fully automatable. But the method relies heavily on the supporting tools to be as efficient as possible, and as we already mentioned new tools is not something that is wanted by UIQ Technology at this time.

3.4 Regression Testing

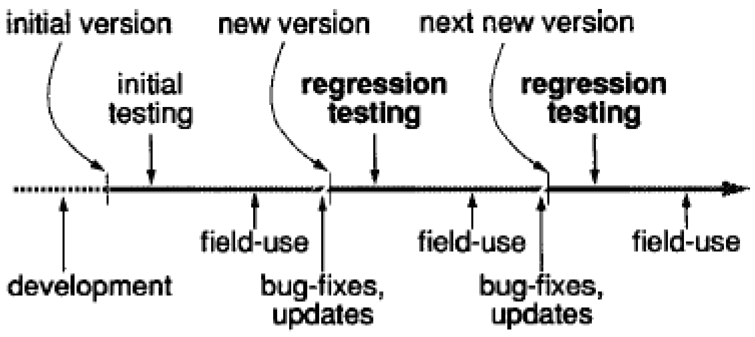
This chapter covers the area of regression testing. A small introduction to the area of regression testing and its basics is written in the beginning of the section. Later we will more thoroughly describe content of selective regression testing along with few different methods to perform this task.

3.4.1 Introduction

Regression testing is an essential activity to any software development to ensure software quality [4]. There are studies showing that more than 50% of development effort in software lifecycle is spent on maintenance [10]. Large parts of those 50% are spent on testing. It is here regression testing proves to be important. By cutting down on the amount of retesting, the time, effort and costs of software can be drastically lowered [10]. As regression testing can repeat itself several times during the lifecycle of a project, cutting down amount of testing in the regression phase is even more important. This however may not be done risking the quality.

The main aim of software regression testing is to, as defined [12], establish the confidence that software acts and performs according to the specification it is based on [12], after modifications to the software. Regression testing is also supposed to assure developers that modifications and additions to the code have not adversely affected unchanged portions of the software [3]. The flow of software can be seen as found in report by Agrawal, Horgan, Krauser and London [14]. Our opinion is that “field-use” even can be replaced by internal company use. Nevertheless, the picture shown in Figure 3 clearly shows when and why software regression testing is used.

Figure 3: Test-flow order [14]



It is widely acknowledged that software regression testing is a highly important activity of software maintenance [3, 4]. Furthermore, software development today tends to be object oriented with high usage of third-party software. Here software regression testing can, and is, used to ensure that third-party software does not cause faults when integrated with the rest of the software [3].

There are two different ways to conduct software regression testing. One method is the *retest all* approach. It does not need further explanation. Changes, modification or additions to the software trigger a regression testing of all software. Even though this method is highly effective, it uses, and mostly, exceeds resources and allocated time. The alternative method to retest all approach is the *selective regression testing* [3]. There are several different methods that perform selective regression testing, but all the methods aim towards same goal: *"reducing amount of time and resources used to retest modified or enriched software by selecting a subset of existing software test-suite [3]."*

3.4.2 Different methods for selective regression testing

Rothermel and Harrold [3] provide an extensive comparison and evaluation of different software regression techniques, provided as Appendix A. The comparison itself is based on four common categories for all methods: inclusiveness, precision, efficiency and generality. It is important to mention that in the beginning of the report Rothermel and Harrold [3] write that most of the methods for selective regression testing are code based as it is now [10, 14, 25, 26, 27, 28, 29, 30], there are however few that do their selection, based on the software specification [31, 32].

3.4.2.1 Slicing method

The first method that was researched more about was the so-called "slicing" method. It has been given a grade of safe, but with limitations, by Rothermel and Harrold [3]. Binkley [13] describes the method as close to 100% accurate. The basic idea of the method is to slice the software into small entities. After that, test-cases are executed and it is noted down what test-cases that execute certain slice(s). To illustrate the approach of the method the simple example below is provided:

Consider following pseudo code as the source code to be traversed:

```
Input(a,b)
If(a>b)
    Write "a is bigger"
If(a<b)
    Write "a is smaller"
Else
    Write "equal size"
```

Complementary to this code test-cases are made as following:

Table 2

Test-case	Input	Output
	a,b	
T1	5,1	"a is bigger"
T2	1,5	"b is bigger"
T3	1,1	"equal size"

The "slice" for test-case T1 will look as the red-marked part of the code:

```
Input(a,b)
If(a>b)
    Write "a is bigger"
If(a<b)
    Write "a is smaller"
Else
    Write "equal size"
```

Later on, incase modifications are made all slices affected are selected for regression testing e.g. if statement "a is bigger" would change into "a is bigger and therefore a winner" test-case T1 would be selected for regression testing. However if something changes outside T1 slice it will not be retested. This is the main reason that the method receives a "safe but with limitations" grade by Rothermel and Harrold [3]. It will simply never check for possible bugs that a modification of the software might have caused outside of the slice. However, for the thesis, even larger concern is how those slices are made. To create those slices you are forced to traverse the source code [13] and make the slices manually, as it is illustrated in example above. In our research material [3, 13, 14] this is applied to small C programs consisting of, at most, few hundred lines of code. UIQ Technology produces much larger software than that, counted in lines of code. This method is rejected as valid approach for our thesis. It is simply too academic and can not in any efficient way be applied to a company of UIQ Technology magnitude. The method can however possibly be applied to unit testing but as stated in 2.2.1 we were not supposed to cover this area of testing. It was also agreed that future investigated methods that where based on source code would be rejected unless they used a very abstract view of the code to select software regression test-cases.

3.4.2.2 Firewall method

The next method investigated is the so-called “firewall” technique. This method failed to prove valid for our thesis with the statement: “*Their technique determines where to place a firewall around modified code modules” [3]*. We did however not reject it directly but researched it fully. Results showed that even if this method uses a bit more abstract view on the software [31, 32], in form of modules, you are still required to look at the source code to build up those modules that should be monitored. The method is however suitable to use at integration testing level and at unit test level [3]. The firewall method has also been implemented and measured in [35]. The work was however, done using a tool called “Test Manager”, which is not wanted by the definition of the thesis. The results of the report [35] also showed that e.g. database usage of the firewall technique is a rather expensive task. The report also stated that method inability to handle large amounts of data might be discouraging. As for example, we refer to the prior subchapter about slicing and the example listed there. Instead of making slices, entire code modules can be put under monitoring e.g. entire example in 3.4.2.1 can be put in one code module. The firewall technique will detect changes inside the firewall but will not check for errors outside the firewall. The method was presented to UIQ Technology but the need of tool support combined with the low abstraction of the method made it a non-valid candidate for a solution.

3.5 Summary of testing methods

As it is previously mentioned, there is a need to use a very abstract level of selecting test cases for both initial testing as well as for regression testing. As it is shown in next sections, in both cases (initial and regression testing) it is clear that a possible solution to the problem is prioritization of test cases based on test specification.

As it is previously written in this chapter, most of the, by the literature study, covered methods had some characteristic that made them obsolete for a possible solution. Usage of heavy algebraic formulas and tools was e.g. not wanted by UIQ Technology. The methods using algebraic formulas are simply too complex and tool-based solutions are not wanted at all since the company mostly uses their own developed tools.

When it comes to regression testing the common characteristic that ruled out most methods was the access to source-code requirement to be able to track changes. Most methods that we wrote about and researched about require good and thorough knowledge about the source-code. They also require, in some cases, traversing trough the source code in order to be able to perform regression testing with the created modules of code. This was not accepted by UIQ Technology since they, as we previously mentioned, have very few test developers with in-depth knowledge about the source code. It is also stated in 2.2.1 that thesis should assume a level of functional/acceptance testing at high level of abstraction which rules out heavy usage of source code.

The initial test-case selection and the regression test-case selection should be done as it is described in section 3.6-3.8. Intention behind giving the proposed methods separate section is to, in depth; explain why the methods were chosen as a base for further research.

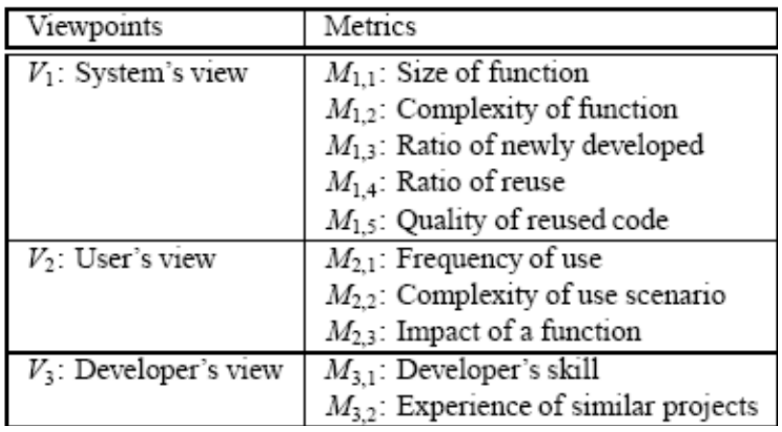
3.6 Test selection methods chosen as a base for the solution proposal

The following two section hold information from the literature study that has been mentioned previously in this chapter. The two methods described in the following two sections are written in separate chapters to highlight that those methods are the base for the solution proposal. Both presented methods use prioritization of test-cases with respect to the specification.

3.7 Selective method for initial testing

The method chosen for further research is based on prioritization of the test-cases, to find the most crucial faults in a product. The idea behind selecting this method is that it is rather simple to understand and implement. It also gives good customization possibilities and it was also found suitable for UIQ Technology testing process in discussions with our supervisors. The basic idea behind this method is that the test developer looks at the system from different views: system, user and development view. Within the views there are metrics which are weighed, See Table 3. The weighing of the metrics together with using a scalar formula, the test-case gets a priority.

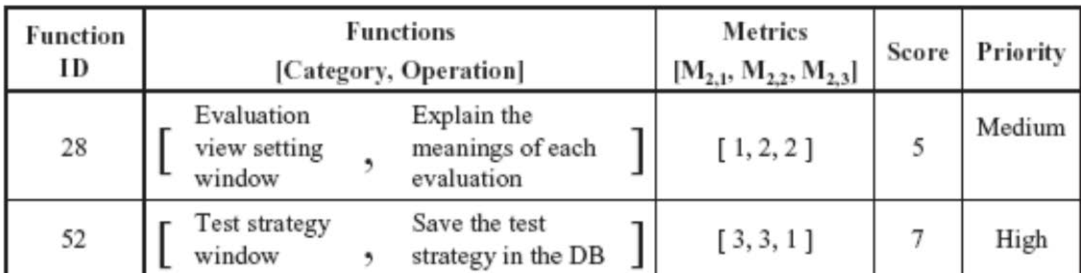
Table 3: Metrics for evaluation. [2]



Viewpoints	Metrics
V_1: System's view	$M_{1,1}$: Size of function $M_{1,2}$: Complexity of function $M_{1,3}$: Ratio of newly developed $M_{1,4}$: Ratio of reuse $M_{1,5}$: Quality of reused code
V_2: User's view	$M_{2,1}$: Frequency of use $M_{2,2}$: Complexity of use scenario $M_{2,3}$: Impact of a function
V_3: Developer's view	$M_{3,1}$: Developer's skill $M_{3,2}$: Experience of similar projects

The principle of the method is that the developer decides what viewpoint he will use to weigh the test-cases. The test-cases are then viewed from the functional perspective, a.k.a. black-box. In the example here, the developer is viewing the test-cases from a user's viewpoint. That means that metrics, **frequency of use**, **complexity of use scenario** and **impact of a function** are used. The metrics then have set values that can be assigned to them, e.g. 1 – 9. The test-developer goes through the functions list and weighs all three of the metrics according to the scale that has been set. See the Table 4 below.

Table 4: Prioritizing functions [2]

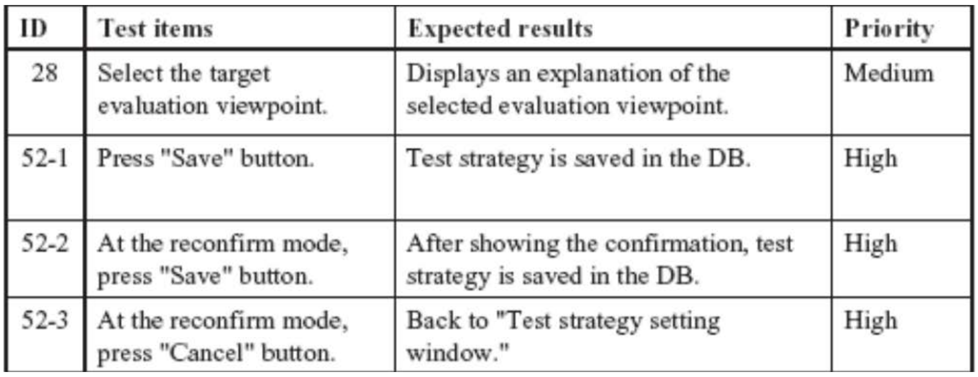


Function ID	Functions [Category, Operation]	Metrics $[M_{2,1}, M_{2,2}, M_{2,3}]$	Score	Priority
28	[Evaluation view setting window , Explain the meanings of each evaluation]	[1, 2, 2]	5	Medium
52	[Test strategy window , Save the test strategy in the DB]	[3, 3, 1]	7	High

Figure 5 presents the two functions that have been weighed. The column on the right end is the priority that the function receives after the metrics have been added. The borders are set by the test-developers before the weighing starts. For example, if the score is between 4 and 6, the function has a medium priority.

After the prioritization has been completed the extraction of the test-cases can commence. The method says that 100% of the high priority test-cases, 50% of the medium and 25% of the low priority test-cases should be executed. The test-case priorities are received from the function that is tested by them, see Table 5. As Table 5 illustrates, if a function even has more then one test-case, all the test-cases associated with the specified function, receive the same priority.

Table 5: Test-cases inherit priority [2]



ID	Test items	Expected results	Priority
28	Select the target evaluation viewpoint.	Displays an explanation of the selected evaluation viewpoint.	Medium
52-1	Press "Save" button.	Test strategy is saved in the DB.	High
52-2	At the reconfirm mode, press "Save" button.	After showing the confirmation, test strategy is saved in the DB.	High
52-3	At the reconfirm mode, press "Cancel" button.	Back to "Test strategy setting window."	High

3.8 Selective method for regression testing

After conducting a rather extensive literature study choice for a method to work with, for this thesis solution, fell on Chen, Probert and Sims work [4]. First of all their method is a black box based software regression selection method. It is based on the specification of the software, which is exactly what this thesis could benefit from. The second big advantage is that the method we pick for selective regression testing is rather similar to the method for initial test selection. This will later in the thesis be proven an advantage.

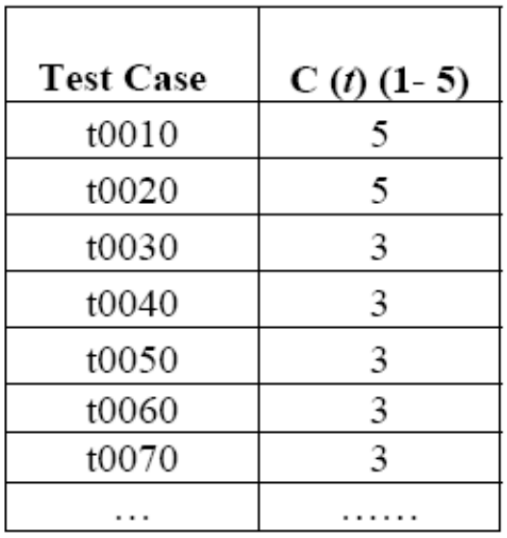
Basic idea of their proposal is to give different metrics for all test cases in certain software. The metrics that Chen, Probert and Sims were using for their proposal where cost, severity probability and risk exposure. Their cost estimation was graded on a 1-5 scale where one was low cost and five was high cost. Furthermore, cost was determined on two different factors:

- The consequence of a fault as seen by the customer, that is, losing market share because of faults

- The consequences of a fault as seen by the vendor, that is, high software maintenance cost because of faults [4]

A table with costs for each test case looks something as the following tables from their thesis [4] see Table 6. In Table 6, cost of one is the lowest cost and five is the highest cost.

Table 6: Cost of test-cases [4]

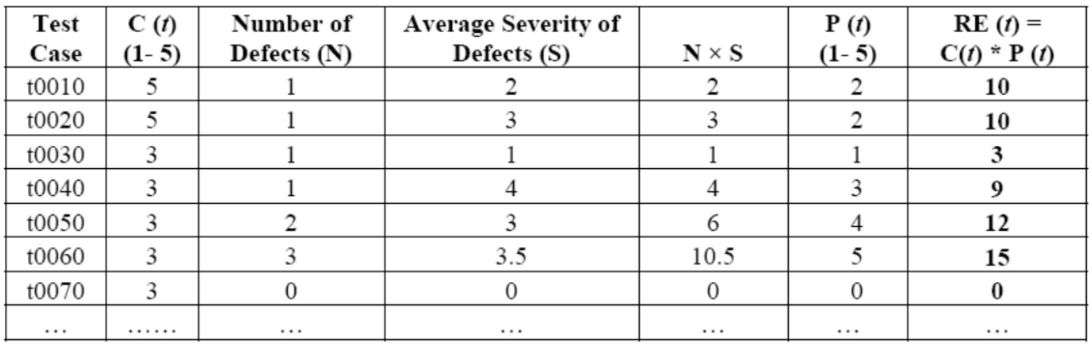


Test Case	C (*t*) (1- 5)
t0010	5
t0020	5
t0030	3
t0040	3
t0050	3
t0060	3
t0070	3
…	……

After the costs have been set it is time to determine severity of defects P(t). This is done by multiplying Number of Defects N, with Average severity of Defects S (NxS). After this is done, cost-estimates are combined with severity-estimates to build up a new value called Risk Exposure (RE).

The result looks as Table 7:

Table 7: Final result [4]



Test Case	C (*t*) (1- 5)	Number of Defects (N)	Average Severity of Defects (S)	N × S	P (*t*) (1- 5)	RE (*t*) = C(*t*) * P (*t*)
t0010	5	1	2	2	2	**10**
t0020	5	1	3	3	2	**10**
t0030	3	1	1	1	1	**3**
t0040	3	1	4	4	3	**9**
t0050	3	2	3	6	4	**12**
t0060	3	3	3.5	10.5	5	**15**
t0070	3	0	0	0	0	**0**
…	……	…	…	…	…	…

Last step of this method would be to select test-cases for regression-testing, based on the value RE. As you can see this is not a method that guarantees perfect regression testing rather providing a way to regression test the most important parts of the system first. Important parts of the system being such, that they cause a very negative impact on the software from a certain point of view. This can also be compare with the method for initial test-case selection. Both methods base their selection at some kind of prioritization, risk prioritization or as in regression test-case selection case cost prioritization (assessment). The similarity of the methods will be used to form the final solution. This will be brought up further in the thesis.

4 CURRENT METHOD FOR TESTING AT THE COMPANY

UIQ Technology has built an interactive database for their testing process. It contains their test-cases which are categorized according to what parts of the software they belong too. Ideally, they all have traceability back to functional requirements that spawned mentioned test-cases. The reason word ideally is chosen is because some of the test-cases are designed to test non-functional requirements, and some are designed for user scenarios, which means they cover several functional requirements, and the database does not include that information. One thing all of the test-specifications contain is a use description. This shows how often a user or the system is going to go use a certain function/feature.

The "frequency of use" description is what the test-developers use when they decide what test-cases should be run. They use four grades to determine this: always, often, rarely and never.

When it is time to do a new test-run the developers, choose the not tested "always"-functions, some "often"-functions, and so on. One of the problems, as we discuss is that they do not have clear boundaries using this method. As it was found out during the discussions with the company reference, the test-cases selected for one test run, often are tested in the following test runs, due to their frequent usage. This fact can lead to retesting of less important test-cases over and over again while more important test-cases are omitted.

The test-leaders also choose test-cases that they think are important for the integrity of the software based on their previous knowledge. This poses another threat to quality guarantee, which we will discuss in section 4.2

4.1 Concerns with current method

As mentioned in report by Tomaszewski et al. [9], beyond a certain number of components developers usually put remaining components in a random order. This is not a statement that this happens at UIQ Technology, but it is assumed that it does. Even when doing prioritization during the implementation of the method we felt these symptoms during the last stages of the prioritization.

Another issue presented by Tomaszewski et al [9] is that the code introduced in the later stages, e.g. modifications, is the most fault-prone code during the development. They found in their research that 37% of the researched software was code that was added as modification, later in development. These 37% of the code, contained 62% faults found in the released software. Another software system that was researched had 44% of the code added as modifications, and this contained 78% of the faults found. This shows how important regression testing is. As it is stated in the introduction the regression testing is currently done in an unstructured way at UIQ Technology, and this poses a quality risk for the software. The company needs a method to decide what needs to be tested to give best coverage the product. They need to be able to decide what test-cases that are crucial for product functionality and what test-cases that can be omitted due to their low impact.

4.2 Analysis of the current method

This section will cover an analysis of the current testing method at UIQ Technology. It covers important aspects, for the thesis point of view, like test-case selection and execution.

As it is mentioned the test developers use a simplistic way of determining what test-cases should be run. There are several problems associated with UIQ Technology method of prioritization and selection of test-cases, and this heritages from how the test-cases are assigned their “frequency of use”-value. During the thesis it was made clear that this value does not always represent the truth. This poses problems for the company. Since the values are not always correct, that leads to test-cases that might not be important being run more times than they actually should. This not only wastes resources on testing redundant functions but can also overlook the testing of important functions.

Tomaszewski et al [9] state that after 15% of the code has been analyzed that the gain from using prediction model is largely increased compared to using expert estimation. They also mention that the time a developer has spent on a project does not raise his or hers prediction of faults. This statement shows that gut feeling does not belong in test-case selection process.

During their research, Tomaszewski et al [9] found that two of the statistical prediction models outperformed the estimations done by the experts. This is yet another proof that using a prediction model would benefit UIQ Technology testing process.

5 SOLUTION PROPOSAL

This chapter describes the method writers propose as a possible solution to the issues proposed for the thesis. After in next section writing a short introduction information about the method, customization of the method and a step by step approach will be written.

5.1 Introduction of the proposal

As it is written in the report, several issues need to be addressed by this thesis. In addition, the solution for those issues has to follow some guidelines. Few of those guidelines are level of abstraction, automation and tool support. In section 3.5-3.8, methods for initial test-case selection as well as method for regression test-case selection are presented and this chapter discusses further development of those methods in order to make them acceptable as the solution for the thesis.

5.2 Main method for solution proposal

As it is written earlier in the thesis, the two methods that were found interesting for possible solution for the thesis are rather similar in approach. Both methods use categorization as test-case selection. This categorization can be e.g. based on prioritization based on risks or cost assessment of the product. This method is in this thesis picked as a possible solution to the issues UIQ Technology wants solved. Another reason for choosing this method as a possible approach in the thesis is that Tomaszewski et al [9] mention that a simple model is more likely to sustain stability over several releases. This statement supports the choice of our method. It is very easy to understand, and we feel should be easy to incorporate. Since UIQ has several releases of their software, that means that they also will benefit from using a simple method for their testing process.

Evolution based system faults heritage from the modified code that is added in later stages of the system development [9]. This demands good testing coverage by the regression method used by the developer. The method that is proposed, groups up the functions of the system into batches[5]. If a new code is added or old one modified in any way, the test-leader can see what batch the modified code belongs to and order the testing of that said batch. The functions that are in same batch are interlaced, so the faults that are introduced by the new/modified code are contained to the batch of said function. This gives the test-leaders guidance on what needs to be tested to get good coverage on the new/modified code. It was previously mentioned that similarity of the methods for initial test-case selection and regression test-case selection are similar and that this will be used by the thesis. The approach to this will be merging of methods to simplify the method itself.

5.3 Customization

Having decided on the method to use, we noticed that it needed some tweaking in order to be useful to UIQ Technology current process without too much altering.

[5] Concept of batches together with creating of those will is explained in chapter 6.3

5.3.1 Customization of metrics

To be able to use the method mentioned in 3.7 as a solution for the thesis some customization was needed. First decision however was to choose point of view that should be used for prioritization of test-cases. As it is mentioned in 2.2.1, the thesis is supposed to operate on functional/acceptance level of testing with aim at customer satisfaction. The decision was taken to use customer point of view for prioritization of functions. Since no real customers could be interviewed due to time constraints on the thesis, authors own experience was used for the prioritization. Both authors of the thesis are active users of mobile phones leading to the conclusion that their prioritization should be valid for the thesis.

Since user point of view has been selected for the prioritization, metrics for this view were assessed. Case-study of UIQ Technology was conducted to determine what metrics can be used. The aim was to at least find an approach to use the metrics stated in the original method [2] but also reuse some of the old documentation found in the company documentation during the case-study. Good guidance was provided on this matter from Hirayama et al [1, 2]. It was found that UIQ Technology documentation supports usage of metrics stated in the original method [2] see table 8.

Table 8: Metrics

Metrics	Values
Complexity	1-10 functions - Low 11-30 functions - Medium >30 functions - High
Impact	High Medium Low
Frequency of use	Very often Often Occasionally Rarely Never

It was also stated that metric frequency of use could be reused from the current documentation at the company but with few modifications. Modifications included removing the always level and instead adding the very often level as well as adding never level to the grading. Reason for removing always level was that we could not find one single function that was always used by a normal user so that value needed redefinition. On the other hand, it is possible to find functions that are never used leading to addition of a new grading, never.

5.3.2 Customization of regression test-selection method

It is mentioned in previous sections that merging of the two methods found in 3.6-3.8 was planned in the thesis. Reasoning behind this approach is to simplify approach of the method but also to reuse some data gained in the initial test-case selection. During the initial test-case selection, prioritization is conducted on the functions. This leads to the fact that most important parts of the system are already prioritized and that this should be reused later on in the development. As it is rather extensive to prioritize entire systems, the idea about reuse of the prioritization done during the initial testing, was discussed.

The regression test-case selection has, as we previously wrote, approximately same issues as the initial test-case selection. UIQ Technology has no structured way to select test-cases for regression leading to several problems. It was decided due to factors of similar methods proposed and due to nature of the problem to reuse the prioritization from the initial test-case selection on the regression testing. In other words, the method that was described in section 3.8 is merged into method for initial test-case selection so that prioritization approach from initial test-case selection is same approach that should be used for regression test-selection thereby there is no need to redo the prioritization. Another discovery was made after decision of merging was taken. Due to nature of data collected during case-study, it was proposed to the company that selection of test-cases for regression can be backtracked to the batches made in the initial test-case selection. Tracing modified functionality to entire functionality batches should lead to improved regression testing in terms of coverage. Every change made to the system will trigger regression testing of an entire batch that contained the test-case and therefore ensure none unintentional bugs were introduced to the function. It was also concluded that UIQ Technology's own tools could be modified for this functionality and that this thesis would not go deeper into that subject.

6 IMPLEMENTATION OF OUR METHOD

This section covers implementation of the proposed solution. The experiment was done in order to be able to determine the accuracy, functionality and general usability of the proposed method. First, a short introduction of the project the experiment is done on is presented. Thereafter a step by step description of the model is shown to give a better overview. Other discussion topics found in the chapter are automation possibilities, batch making and the implementation of method in terms of work required.

6.1 Introduction of the system

In order to be able to compare the proposed method to the current method used by UIQ Technology, we had to compare the test results extracted from the company documentation based on both current and proposed method. The idea was that if we looked at the company current approach to testing we would execute X amount of tests finding Y amount of failures. While executing our own method containing approximately same amount of tests (X) and finding, hopefully, at least same amount of fails (Y) but with different spread. What we were hoping for is that our method would find high risk failures at higher rate.

The system we used for comparison is an ongoing project, containing around 4000 test-cases. It is important to mention that this project was still growing as we were examining it, so we base-lined it roughly two weeks after test-start, the test case amount is expected to grow by a large amount. All changes to the system beyond this time where ignored and all tests conducted on the base-lined version of the system. Another important aspect of the system is that even if around 4000 test cases does not sound much, most of the test cases, if not all, are run multiple times.

The first step of the method is built up of examining the system and making our previously mentioned batches.

6.2 Step by step

This contains a short overview of the activities of the proposed method. All the steps will be further discussed in separate chapters.

The method can be described with three basic steps:

1. Create Batches (Section 6.2)
2. Prioritization (Section 6.3)
3. Execution (Section 6.4)

6.3 Creating of batches

The concept of batches is of same importance as expressions like FS-Trace and initial testing and therefore it needs further explaining.

Batches are groupings of functions. After data from the case-study was extracted decision was fast taken that prioritizing all the functions separately would not be applicable. Also because of the regression test solution, it was decided to create the so-called batches. The notation used in the FS-Trace by UIQ Technology was of great help while making those batches. As we previously mentioned a batch tag can look something like following:

FS.someGlobalFunction.functionInsideGlobalFunction.subFunction

The dot notation of the tags was used for creating of batches since it was in the case-study discovered that most batches followed a standard that was applicable to our thesis. For example, a tag of a certain function in the FS-Trace was FS.Agenda.Viewing. This should be seen as function to view agenda entries. Agenda is the main function and viewing is the sub function. This was just an example, most of the batches actually had even more sub-categories. After the case-study was finished decision was taken to create batches by following rules:

1. Start at the left side of the tag (at the "FS")
2. Move two dots to the right (in FS.Agenda.Viewing this would mean moving just before the word Viewing
3. Group up function by the following word after second dot

The reasoning behind this was that if batches were made like this functions were grouped up in such a way that interlacing between them was minimal. This condition was a prerequisite for the solution of regression testing were we needed batches where we almost can guarantee they are not interlacing with other batches.

6.4 Prioritization

As the batches are done they should be prioritized to the previously mentioned point of view, in this thesis user point of view. As you can read in 5.3.1 the final prioritization value consist of three different values, complexity, impact of a function and frequency of use, added together into one value.

First value to assign, as proposed in our method, is the complexity of the functions. Given values of 0.0 to 1.0 all batches were traversed with a macro in Microsoft Excel. Impact of a function value can however, be more complex to determine. To determine impact of a function we need to actually know what all functions do, which is not clear all the times. To overcome this problem we were used a test mobile phone to check functions on. This mobile was a Sony Ericsson m600 and contained UIQ software. Impact of a function prioritization is also done using numbers from 0.0 to 1.0. Assigning of values to impact of a function was proven to be the most time consuming process of all value assigning. We often ended up in long discussions about how serious a failure in a certain function would be. Last value we had to set was frequency-of-use. After some extensive discussion we decided to assign values here using customer point-of-view. We realized that this view is what we can do with good results since we both are mobile phone users. It is however important to say that even though customer point of view sounds like one single view it is not. While discussing we agreed that the customer might be everything from a teenager to a C.E.O. of a certain major company. Their respective usage of a mobile phone will most probably differ a lot. We decided here to put ourselves in the middle of those two extremities. This way we would cover as much as possible of important functions in a mobile phone for all different kinds of users.

After finished prioritization the batches can be grouped up in categories of high, medium and low priority batches. To do this classification, all values from prioritization are multiplied by 3 and then added together e.g. (complexity*3 + impact*3 + frequency*3). Reason to multiplication with 3 is to get more manageable numbers. The border values were set as following: Low priority 0 to 2,9, medium priority 3-5,9 and high priority 6-9. This gave following grouping of batches.

- **High priority batches 86**
- **Medium priority batches 141**
- **Low priority batches 85**

The comparison on methods described later is based on defect detected by the each method involved. Therefore, it was needed to transform the batches into raw test-case amount. The resulting amounts are as following:

- **High priority test-cases 54,4%**
- **Medium priority test-cases 27,9%**
- **Low priority test-cases 17,7%**

The numbers gained here give a good indication of how the system is prioritized. It was intentional to prioritize many test-cases as high to be able to guarantee none critical function would be left out of testing. Proposed method [2] writes that 100% of high priority, 50% of medium priority and 25% of low priority test-cases should be executed. In the case of this thesis that means executing around 3000 test-cases, or 78% of original amount It is important to remember that this is just one single execution of testing and that the amount of not-tested test-cases will grow rapidly as UIQ does multiple test runs during a project lifetime.

6.5 Test results with current method at the company

As we previously mentioned we need data on how UIQ current process behaved to be able to compare it to our proposed method. We did encounter a big problem here however. Extracting those test-cases along with all needed data was a highly time-consuming effort. We agreed that there was no possible way to check all test-cases that were in the system and after discussions with our supervisor Kennet Henningsson we decided to use sampling. It was decided that we should take a certain period of time, containing certain amount of test-cases and assume that we can mathematically transform the results onto the rest of the system. As we will explain later this is not a critical issue.

To extract results of the testing conducted we defined the sampling span for the data collection. This is also the base for test-case selection, while executing the company's current method the only selection that was done was to pick same test-cases that were originally picked by the company. It was decided to look at the beginning of the testing and choose a time span for which the test-case execution results should be executed. The amount of sampled test-cases was about 600.

Next step in the process was to investigate those 600 test-cases and determine if they failed or passed their first execution. Reason for the choosing first execution was that we wanted to see what happened first time ever the test-case was executed. Choosing later execution point would probably provide us with faults that were introduced to the system at a later point meaning the results would not reflect the first testing done.

A rather interesting problem was encountered here, many of the test-cases where not connected to the FS-trace at all. Those test-cases were mostly UI-Specification ones so they were naturally not stated in the functional specification. This however did not cause any bigger problems since we agreed that those test-cases can be seen as rest of the test-cases in terms of comparison, only that they would require more manual work to compare with our results. The non-coupled test-cases got removed from further investigation. Looking up all data about those 600 test-cases that made the sampling area, data about defect spread detection was gathered. Following results represent what spread the detected defects had in respect to the 3 different categories we defined earlier:

- **High priority failures 71,62%**
- **Medium priority failures 18,2%**
- **Low priority failures 9,5%**

As we previously mentioned there is few test-cases that are not coupled to the FS-Trace. In order to be able to provide fair comparison to the proposed method it was determined that about 60% of the test-cases where coupled to the FS-Trace and therefore the proposed method should execute around same amount, 400 test-cases.

6.6 Test results with proposed method

This chapter provides information about execution of the proposed method on the company test-cases. Test-case selection was done according to the method proposal meaning highest priority batches (containing test-cases) were picked to be executed first. Amount of batches that should be executed was determined by as we explained in previous section amount of executed test-cases by the company's current method. This means that batches containing approximately 400 test-cases were picked for experiment.

Nine of our batches that where prioritized as highest possible proved to not be run at the company at all. Explanation we received was that those batches contained functions, as calling with the phone and various other network dependent functions. Those test-cases were never executed at UIQ but together with their customers at customer location. The reason we missed this fact was that this was rather obvious to the people at UIQ. However, this obstacle was overcome. Since all high-prioritized batches will be executed at one time or another, according to proposed method, next "in line" batches were picked for execution. The amount of high-prioritized batches was big enough for this approach.

Total of 8 batches were selected for the execution part. Same approach was applied here with researching the outcome of the first instance of execution for each test-case in the selected batched. The detected defect spread was as following:

- **High priority failures 100%**
- **Medium priority failures 0%**
- **Low priority failures 0%**

Here it is clear that the method is performing according to the previously mentioned goal of finding the high-risk defect first. Furthermore, it is worth to mention that amount of total defects found was approximately the same for both current and the proposed method.

7 EVALUATION OF THE METHOD

This chapter covers the evaluation of the method that has been proposed in the thesis. It starts with a small discussion about how the method is to introduce to the existing data and documentation at UIQ Technology. Further, results of the methods are compared and evaluated.

7.1.1 How was it to introduce

Introducing the method was, at least for us doing it the first time, rather extensive work. There was a lot of data that needed to be collected, sorted and filtered before it could be used by the method and this took quite a bit of time. We do however feel that large parts of this work can fully, or partially, be automated for future, therefore leading to less effort for same result. In discussions with Kennet Henningsson, we agreed that only parts that can not be automated with ease are the impact of a function and frequency of use values. Those values can however, be assigned by people writing the tests. These persons will mostly have a good overview over the part of the system they are writing test-cases for and should therefore be able to prioritize those values with good accuracy. This part of the method can however be done in numerous ways. To ensure quality of prioritization the company can let several different teams do the prioritization is one example of an approach for this.

7.2 Comparison of the methods

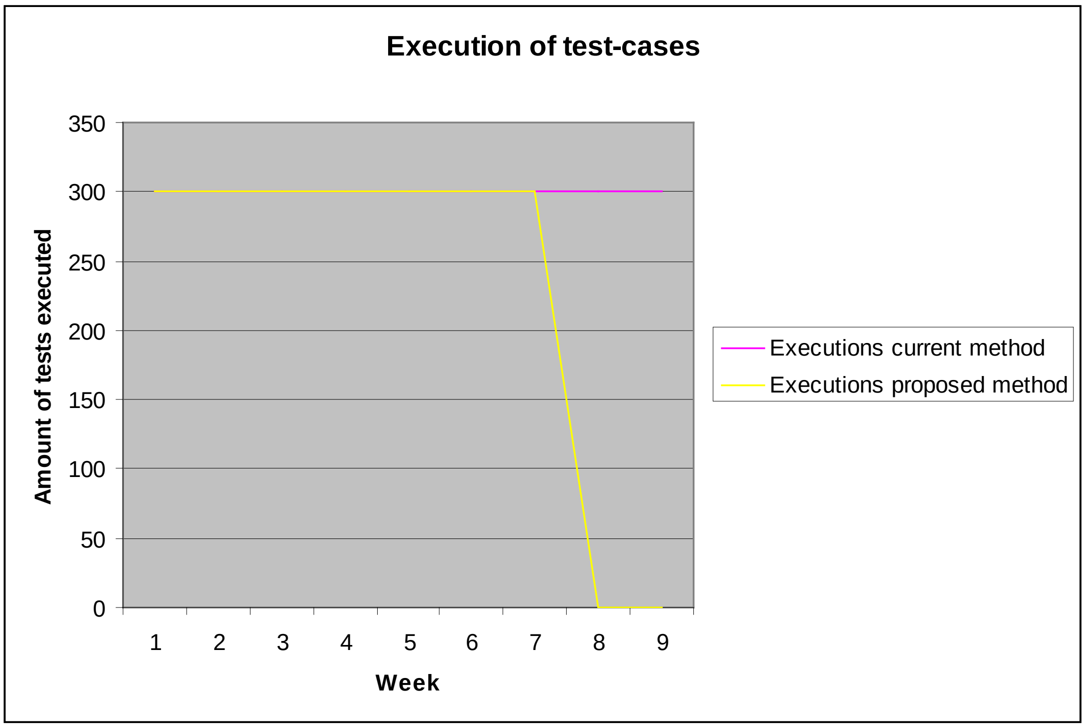
In this section, we will compare the results we got from collecting and executing the data from the selected project. Our intention is to give a good overview over the gains and losses of our proposed method if it would be used by UIQ technology.

7.2.1 Comparison

As we previously wrote in the report, both methods did find about the same number of defects in the system when executing a certain amount of test-cases. The proposed method however, found only high-priority test-cases while current method applied by the company had a wider spread amongst high-, medium- and low-priority test-cases.. This "trend" will continue with our method as long as there are high prioritized test-cases to execute due to the fact our method always executes high prioritized test-cases first. Comparing those two methods from this point-of-view it is clear that our proposed method has an advantage since it will find critical defects first e.g. defects that will badly hurt the system if not removed. We do admit that even current company method might do that, but that is based on circumstances and is not really to be trusted at. We feel that having a guarantee to find critical errors first is worth a lot for the company.

As testing process continues, the methods will differ after certain amount of time. We wrote previously that out method executes 100% of high-prioritized test-cases, 50% of medium prioritized and 25% of low prioritized test-cases. This leads to the fact that not all test-cases will be executed in one test-run. The company will still be able to say that the system is "good enough" since test-cases that are not executed will not cause any critical errors. To illustrate this further following diagram is drawn:

Diagram 1: Execution of test-cases over time



In this case, we assume that company executes 300 test-cases per week. As you can see, both methods will follow the same pattern until week 8. By this point, our method will have executed all test-cases that are necessary in order to guarantee good coverage while the current method used by the company will continue testing. We do also feel that another very important advantage of our method is that the company will get a definite point in time when to stop testing. As we previously mentioned testing in its current form is done in an unstructured way. With our method, this is not necessary and company will at all times know when to stop testing. In addition, as we previously mentioned the proposed method will decrease amount of executed test-cases with 22% without sacrificing the quality.

7.2.2 Deviations

As we previously mentioned we had to make some adjustments and tradeoffs while collecting and examining the data from the selected project.

The first adjustment of data that we described was found while making the batches for our method. In the FS-Trace we used for collecting the data there were some functions with a very strange tag. An entire batch that held 24 test-cases was not usable due to notation errors. All the tags ended with a "??" thus making them either uncompleted or simply malicious. We brought this up in a discussion with Kennet Henningsson and we agreed that we would simply mark all batches looking like this and remove them from further data analysis.

Next big trade-off we had to make while executing our method was the exclusion of non FS-Trace coupled test-cases. As you have been able to read in previous chapters we had a quite large amount of test-cases not being connected to the FS-Trace. Test-cases that were not coupled were either UI-specification test-cases or they were not included in the FS-Trace. After discussions with Kennet we decided to remove those cases from the thesis. The main reason for this was that when executing our own method we will only be using FS-Trace. Therefore no test-cases based on e.g. UI-specification will be included. To keep data in consistent state we decided to only use FS-Trace coupled test-cases and we also agreed, together with Kennet, that values we get can most probably be mathematically transferred to the non-coupled cases later.

The last deviation, we think should be mentioned; origins from execution of our own proposed method. As we have described before our method selects highest prioritized batches and executes those first. We soon however discovered that most of the highest prioritized batches where not executed at all by UIQ. Amongst batches there where not executed by UIQ we found batches for making a phone call or playing a sound signal on incoming call. This was quickly taken to Kennet and we got a fairly obvious explanation. Those methods where not executed by UIQ test-department simply because they where executed at other location together with their customers at the customer site. This situation did not however cause any major changes. Since we had a large amount of batches prioritized as highest we simply moved on and picked next batch “in line” to be tested.

8 VALIDITY OF THE THESIS

We found few validity threats during this research that might have a negative effect on this thesis. In this chapter we will describe those and also describe what we feel can be done to overcome those threats.

During the prioritization phase we used our own experience with mobile phones to set the priorities. This can be validity threat since we have a certain view on mobile-phone usage. To negate personal experience the company can use broader selection of focus group. The focus group in itself can contain testing engineers and real users to spread out the prioritization values. Using both engineers and end-users, should remove extreme values such as, over prioritizing a certain function. We discussed this issue with Kennet before even conducting the prioritization and we agreed that using focus groups or different point of views might solve this problem and provide a more accurate prioritization.

In order to have a reasonable amount of test-cases to compare, we had to use sampling. This is a big validity threat because; defect density can be different outside the chosen sample. Given the time for this master thesis there was no time to do anything else but sampling. So in order to overcome this validity threat the company should execute the method on full test-suite. This only needs to be done once however to ensure that the method is valid for a full test-suite.

As we wrote earlier in the report we had to remove test-cases from the experiment due to them not being coupled to the FS-Trace. We discussed this issue with Kennet and this action was approved. We do however feel that this is a validity threat to not include test-cases that are e.g. in UI-Specification. To overcome this validity threat there is a need to execute our method on all test-cases, not only those coupled to the FS-Trace. There was also certain amount of test-cases that were badly documented. Those test-cases were also removed from the experiment. To overcome this, the documentation should be controlled in a better way to ensure that all test-cases are properly documented and can be used by the method. We do realise that this might be a lot of work for UIQ but since our proposed method is highly dependant on traceability, they would need to improve in some areas of documentation.

Another threat to the validity of this thesis is that all this work is conducted on only one system/project. We used this project because it was by far the best documented project UIQ could provide to us. This project was also described, by our company contact, as representative for how UIQ are conducting their work at the moment. To overcome this threat the method should be executed on few other systems with approximately the same output. UIQ is adopting a new way of documentation, specially the FS-Trace part. This should lead to the fact that they will get more systems later on to execute our method on and test if it is performing well independent of the system we executed it on.

Finally we would like to write that we feel that this thesis is valid if it is seen in a proper way. This thesis should be used as a step in the right direction by the UIQ. It merely shows what can be done with a fairly simple approach but it does need more research and tailoring before it can be used.

9 CONCLUSION

The goal of this thesis was to find a suitable test-case selection method for UIQ. The prerequisites of the method were that the method should be easy to incorporate and use in an already existent development process. It was also highly sought after that the method was close to fully automatable. The following questions were used as guidance for finding a suitable method.

1. Is there a process that could be suited to fit UIQ's regression testing?
    a. What amount of test-cases that should be rerun would this process result in compared to the amount of test cases rerun today?
    b. How will the final quality of the product be, compared to the quality of the product as it is today?

2. Is there any way to do initial test-case selection so it decreases number of test-cases run while maintaining same quality?
    a. How would this work practically?
    b. Is this method feasible to use for a company like UIQ?
    c. What are the advantages and disadvantages of such process?

Besides the questions, consultation with Kennet Henningsson was done on weekly basis during the initial weeks, to get feedback on what is feasible to use and to incorporate.

The results in chapter 7.2 show that even though our method only executes about 78% of the initial amount of test-cases it still manages to find same amount of defects in the system as current testing method thus maintaining at least same quality as current testing process. We do realize that it will not find all low prioritized defects, which can be seen as a disadvantage, but we will find all high prioritized defects first in the testing process. Together with Kennet we agreed that this is a fair tradeoff to do. We do also want to mention once more, that with our proposed method UIQ will get a way to decide when to stop testing, which is a big advantage. We feel that the questions we asked in the beginning of this thesis were answered by our research.

Finally we would like to write that we do not feel that we have finished this method in any way. It still need more researching and work before it is fully applicable.

10 FUTURE WORK

Future work with this thesis is needed to further improve but also prove usefulness of the proposed method for test-case selection. Few of the propositions we have for further work are:

1. Executing the method on an entire project in order to ensure its validity when it is used on whole project instead sampling.
2. It would be interesting to execute the method on more than one project to ensure that it performs in a good way unbound of the project it is applied on.
3. Researching the idea this thesis presented about regression testing and find out if it works well the way it is proposed here.
4. Automating as much as possible of the method, so that real time and resource usage can be measured.

ACKNOWLEDGEMENTS

The authors would like to thank UIQ for providing necessary material for conducting this master thesis. We would like to thank Karin Petermann at Human Resources at UIQ for being initial contact with UIQ and arranging for us to conduct this thesis and we would also like to thank our advisors Kennet Henningsson (UIQ) and Håkan Grahn (BTH) for the help and support they provided during this thesis.

11 REFERENCES

[1] M. Hirayama, T Yamamoto, J. Okayasu, O. Mizuno and T. Kikuno, *A Selective Software Testing Method Based on Priorities Assigned to Functional Modules*, IEEE, 2001, 259-267

[2] M. Hirayama, T Yamamoto, J. Okayasu, O. Mizuno and T. Kikuno, *Elimination of Crucial Faults by a New Selective Testing Method*, Proc. Of the 2002 Int. Symposium on Empirical Software Engineering, 2002, 1-9

[3] G. Rothermel and M. J. Harrold, *Analyzing Regression Test Selection Techniques,* IEEE Transactions on Software Engineering, 22 (1996), 529-551

[4] Y. Chen, R.L. Probert and D. P. Sims, *Specification-based Regression Test Selection with Risk Analysis,* Chillarege Press, 2003, 1-14

[5] S. Elbaum, A. G. Malishesky and G. Rothermel, *Test Case Prioritization: A Family of Empirical Studies*, IEEE Transactions on Software Engineering, 28 (2002), 159-182

[6] S. Elbaum, A. G. Malishesky and G. Rothermel, *Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization,* IEEE, 2001, 329-338

[7] M. Hirayama, T Yamamoto, J. Okayasu, O. Mizuno and T. Kikuno, *Generating Test Items for Checking Illegal Behaviors in Software Testing,* IEEE, 2000, 235-240

[8] W.E. Wong, J.R. Horgan, S. London and A.P. Mathur, *Effect of Test Set Minimization on Fault Detection Effectiveness,* Software – Practice and Experience, John Wiley & Sons, 28 (1998), 347-369

[9] P. Tomaszewski, J. Håkansson, H.Gråhn and L. Lundberg. *Statistical Models vs. Expert Estimations for Fault Prediction in Modified Code – an Industrial Case Study,* 213-239

[10] Y-F. Chen, D. Rosenblum and K-P. Vo, *TestTube: A System for Selective Regression Testing,* IEEE, 1994, 211-220

[11] J. Laski and W. Szermer, *Identification of Program Modification and its Application in Software Maintenance,* IEEE, 1992, 282-290

[12] I. Granja and M. Jino, *Techniques for Regression Testing: Selecting test Case Sets Tailored to Possibly Modified Functionalities,* Pontifical Catholic Uni. Of Campinas

[13] D. Binkley, *The application of program slicing to regression testing,* Elsevier Science, 1998, 583-594

[14] H. Agrawal, J.R. Horgan, E.W. Krauser and S.A. London, *Incremental Regression Testing,* IEEE, 1993, 348-357

[15] R-K. Doong and P.G. Frankl, *Case Studies on Testing Object Oriented Programs,* ACM, 1991, 165-177

[16] T.H. Tse, F.T. Chan, H.Y. Chen, *An Axiom-Based test Case Selection Strategy for Object-Oriented Programs,* 107-114

[17] Y.T. Yu, S.P. Ng and E.Y.K Chan, *Generating, Selecting and Prioritizing Test Cases from Specification with Tool Support,* Proceedings of the Thirds Int. Conf. On Quality Software, 2003,

[18] M. Clermont and D. Parnas, *Using Information about Functions in Selecting Test Cases*, ACM, 2005

[19] B. Beizer, *Black-box testing: techniques for functional testing of software and systems,* John Wiley & Sons, NY, 1995

[20] K.Y Cai, *Software defect and operational profile modelling,* Kluwer Academic Publishers, 1998

[21] D.M. Cohen, S.R. Dalal, J. Parelius and G.C. Patton, *The combinatorial design approach to automatic test generation,* IEEE Sofwater, 26 (1996), 83-88

[22] P.D. Coward, *A review of software testing*, Information and Software Technology, 30 (1988) 189-198

[23] Y.K. Malaiya, *Antirandom testing: Generating the most out of black-box testing,* In Proc. 6th International Symposium on software Realibility Engineering, 1995, 86-95

[24] I. Sommerville, *Software Engineering,* 4th edition, Addison-Wesley, MA, 1992

[25] S. Bates and S. Horwitch, *Incremental Program Testing Using Program Dependence Graphs,* Proc. 20th ACM Symp. Principles of Programming Langugages, 1993, 384-396

[26] P. Bendusi, A. Cimitile and U. De Carlini, *Post-Maintanance Testing Based on Patch Change Analysis*, Proc. Conf. Software Maintanance, 1988, 352-361

[27] D. Binkley, *Reducing the cost of Regression Testing by Semantics Guided Test Case Selection,* Proc. Conf. Software Maintenance, 1995, 251-260

[28] K.F Fischer, *A Test Case Selection Method for the Validation of Software Maintenance Modifications,* Proc. COMPSAC, 1977, 421-426

[29] K.F. Fischer, F. Raji and A. Chruscicki, *A Methdodology for Retesting Modified Software,* Proc. Nat'l Telecommunication Conf., 1981, 1-6

[30] R. Gupta, M.J. Harrold and M.L. Soffa, *An Approach to Regression Testing Using Slicing,* Proc. Conf. Software Maintanance, 1992, 299-308

[31] H.K.N. Leung and L.J. White, *Insights into Testing and Regression testing Global Variables,* Software Maintanance, 2 (1990), 209-222

[32] H.K.N. Leung and L.J. White, *A Study of Integration Testing and Software Regression at the Integration Level,* Proc. Conf. Software Maintanance, 1990, 290-300

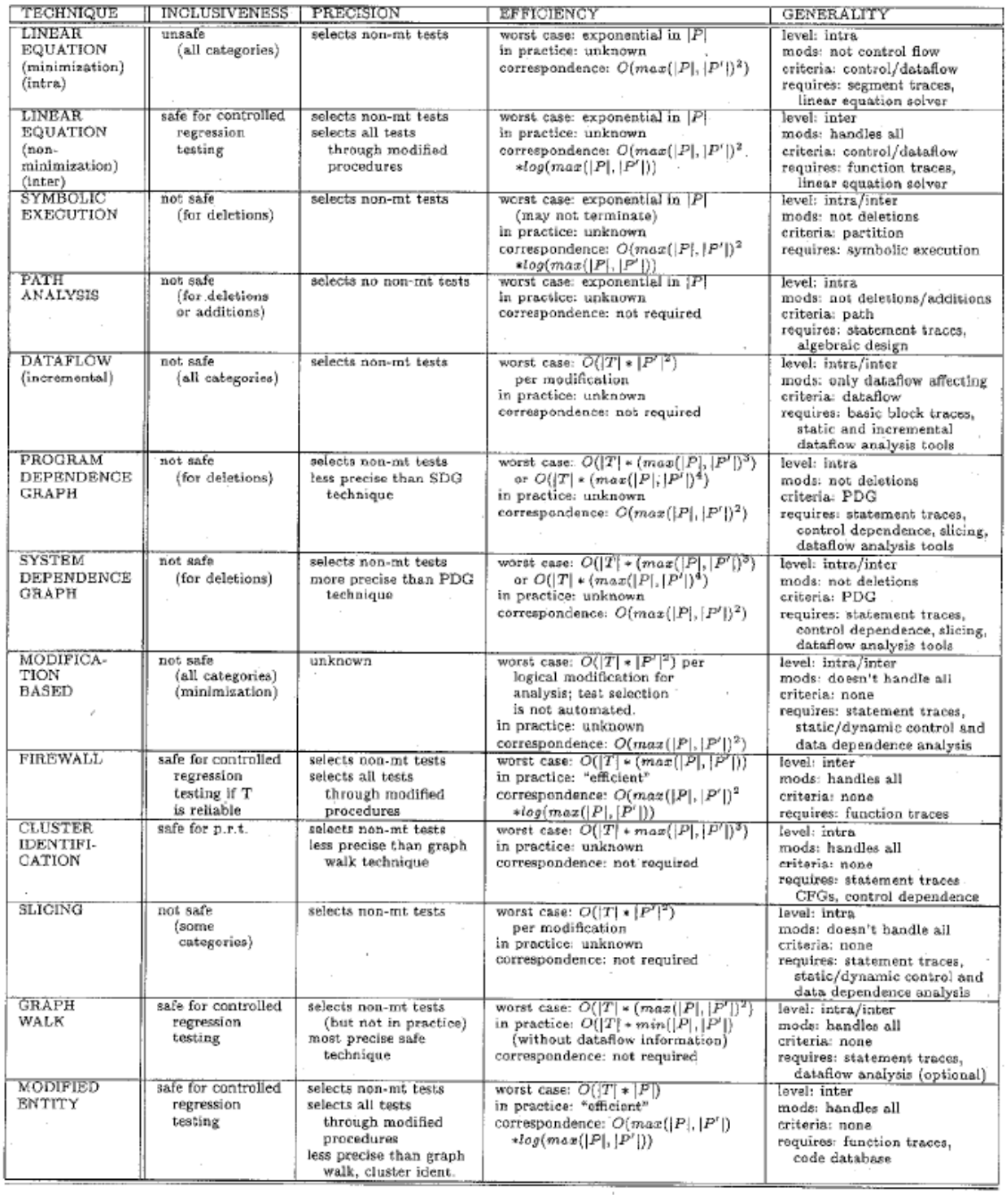
[33] M.J. Harrold, J.A. Jones, T. Li and D. Liant, *Regression Test Selection for Java Software,* Proceedings of the ACM Conf. on OO Programming, 2001

[34] R.W. Miller, C.T. Collins, *Acceptance Testing,* XPUniverse, Juli 2001

[35] White, L.J. Narayanswamy, V. Friedman, T. Kirschenbaum, M. Piwowarski,P.Oha,M. *Test Manager: A regression-testing tool,* Dept. of Comput. Eng. & Sci., Case Western Reserve Univ., Cleveland, OH,1993

12 APPENDIX A

Summary of software regression test-case selection methods [3]



TECHNIQUE	INCLUSIVENESS	PRECISION	EFFICIENCY	GENERALITY
LINEAR EQUATION (minimization) (intra)	unsafe (all categories)	selects non-mt tests	worst case: exponential in $\lvert P\rvert$ in practice: unknown correspondence: $O(max(\lvert P\rvert, \lvert P'\rvert)^2)$	level: intra mods: not control flow criteria: control/dataflow requires: segment traces, linear equation solver
LINEAR EQUATION (non-minimization) (inter)	safe for controlled regression testing	selects non-mt tests selects all tests through modified procedures	worst case: exponential in $\lvert P\rvert$ in practice: unknown correspondence: $O(max(\lvert P\rvert, \lvert P'\rvert)^2$. $*log(max(\lvert P\rvert, \lvert P'\rvert))$	level: inter mods: handles all criteria: control/dataflow requires: function traces, linear equation solver
SYMBOLIC EXECUTION	not safe (for deletions)	selects non-mt tests	worst case: exponential in $\lvert P\rvert$ (may not terminate) in practice: unknown correspondence: $O(max(\lvert P\rvert, \lvert P'\rvert)^2$ $*log(max(\lvert P\rvert, \lvert P'\rvert))$	level: intra/inter mods: not deletions criteria: partition requires: symbolic execution
PATH ANALYSIS	not safe (for deletions or additions)	selects no non-mt tests	worst case: exponential in $\lvert P\rvert$ in practice: unknown correspondence: not required	level: intra mods: not deletions/additions criteria: path requires: statement traces, algebraic design
DATAFLOW (incremental)	not safe (all categories)	selects non-mt tests	worst case: $O(\lvert T\rvert * \lvert P'\rvert^2)$ per modification in practice: unknown correspondence: not required	level: intra/inter mods: only dataflow affecting criteria: dataflow requires: basic block traces, static and incremental dataflow analysis tools
PROGRAM DEPENDENCE GRAPH	not safe (for deletions)	selects non-mt tests less precise than SDG technique	worst case: $O(\lvert T\rvert * (max(\lvert P\rvert, \lvert P'\rvert)^3)$ or $O(\lvert T\rvert * (max(\lvert P\rvert, \lvert P'\rvert)^4)$ in practice: unknown correspondence: $O(max(\lvert P\rvert, \lvert P'\rvert)^2)$	level: intra mods: not deletions criteria: PDG requires: statement traces, control dependence, slicing, dataflow analysis tools
SYSTEM DEPENDENCE GRAPH	not safe (for deletions)	selects non-mt tests more precise than PDG technique	worst case: $O(\lvert T\rvert * (max(\lvert P\rvert, \lvert P'\rvert)^3)$ or $O(\lvert T\rvert * (max(\lvert P\rvert, \lvert P'\rvert)^4)$ in practice: unknown correspondence: $O(max(\lvert P\rvert, \lvert P'\rvert)^2)$	level: intra/inter mods: not deletions criteria: PDG requires: statement traces, control dependence, slicing, dataflow analysis tools
MODIFICATION BASED	not safe (all categories) (minimization)	unknown	worst case: $O(\lvert T\rvert * \lvert P'\rvert^2)$ per logical modification for analysis; test selection is not automated. in practice: unknown correspondence: $O(max(\lvert P\rvert, \lvert P'\rvert)^2)$	level: intra/inter mods: doesn't handle all criteria: none requires: statement traces, static/dynamic control and data dependence analysis
FIREWALL	safe for controlled regression testing if T is reliable	selects non-mt tests selects all tests through modified procedures	worst case: $O(\lvert T\rvert * (max(\lvert P\rvert, \lvert P'\rvert))$ in practice: "efficient" correspondence: $O(max(\lvert P\rvert, \lvert P'\rvert)^2$ $*log(max(\lvert P\rvert, \lvert P'\rvert))$	level: inter mods: handles all criteria: none requires: function traces
CLUSTER IDENTIFICATION	safe for p.r.t.	selects non-mt tests less precise than graph walk technique	worst case: $O(\lvert T\rvert * max(\lvert P\rvert, \lvert P'\rvert)^3)$ in practice: unknown correspondence: not required	level: intra mods: handles all criteria: none requires: statement traces CFGs, control dependence
SLICING	not safe (some categories)	selects non-mt tests	worst case: $O(\lvert T\rvert * \lvert P'\rvert^2)$ per modification in practice: unknown correspondence: not required	level: intra mods: doesn't handle all criteria: none requires: statement traces, static/dynamic control and data dependence analysis
GRAPH WALK	safe for controlled regression testing	selects non-mt tests (but not in practice) most precise safe technique	worst case: $O(\lvert T\rvert * (max(\lvert P\rvert, \lvert P'\rvert)^2)$ in practice: $O(\lvert T\rvert * min(\lvert P\rvert, \lvert P'\rvert)$ (without dataflow information) correspondence: not required	level: intra/inter mods: handles all criteria: none requires: statement traces, dataflow analysis (optional)
MODIFIED ENTITY	safe for controlled regression testing	selects non-mt tests selects all tests through modified procedures less precise than graph walk, cluster ident.	worst case: $O(\lvert T\rvert * \lvert P\rvert)$ in practice: "efficient" correspondence: $O(max(\lvert P\rvert, \lvert P'\rvert)$ $*log(max(\lvert P\rvert, \lvert P'\rvert))$	level: inter mods: handles all criteria: none requires: function traces, code database