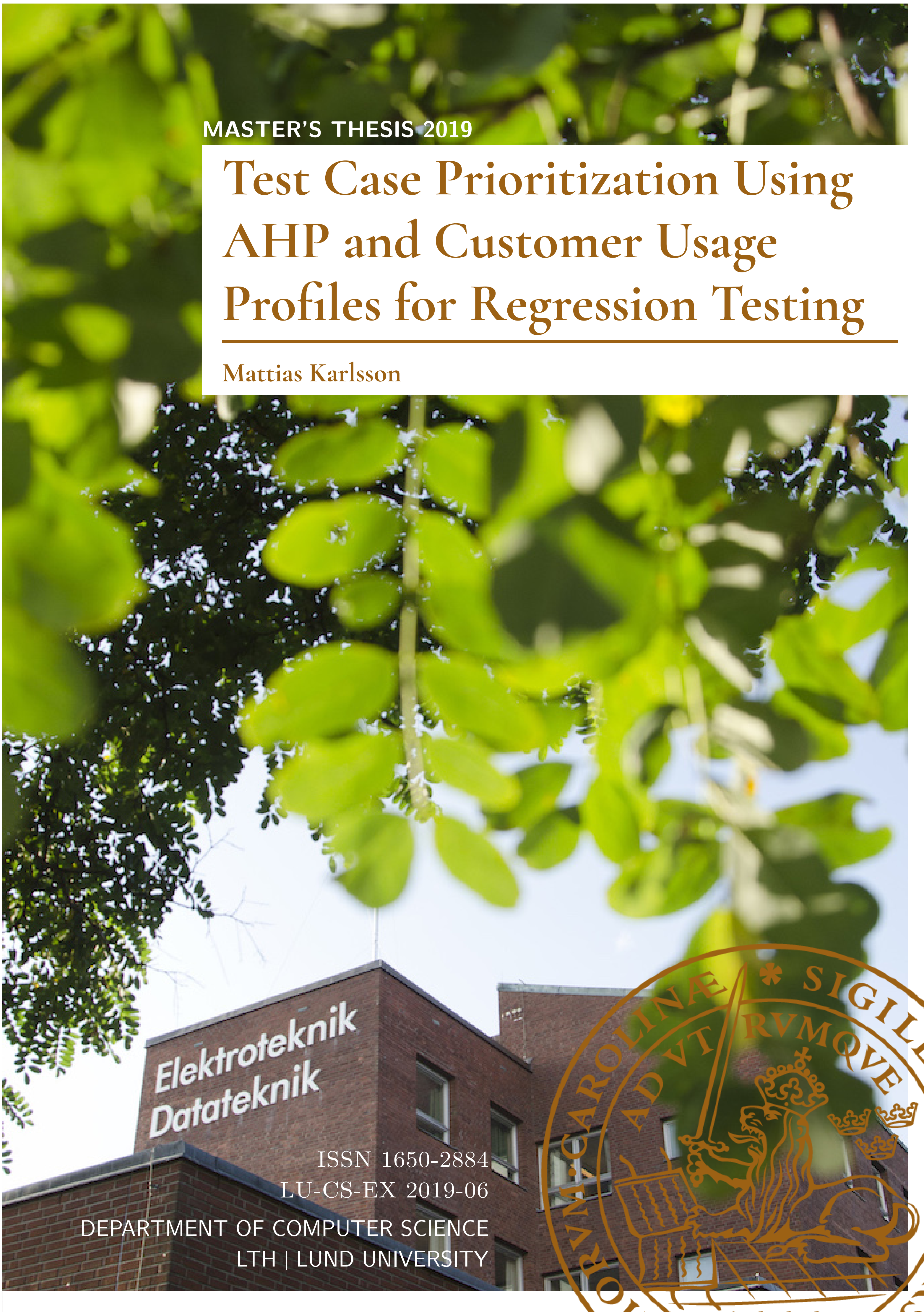

MASTER'S THESIS 2019

Test Case Prioritization Using AHP and Customer Usage Profiles for Regression Testing

Mattias Karlsson

ISSN 1650-2884
LU-CS-EX 2019-06

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2019-06

Test Case Prioritization Using AHP and Customer Usage Profiles for Regression Testing

Mattias Karlsson

Test Case Prioritization Using AHP and Customer Usage Profiles for Regression Testing

(A Mixed Methods Study at ASSA ABLOY Global Solutions)

Mattias Karlsson
`Elt14mka@student.lu.se`

June 17, 2019

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Per Nordbeck, `per.nordbeck@assaabloy.com`
Markus Borg, `markus.borg@cs.lth.se`
Sahar Tahvili, Ericsson AB, `sahar.tahvili@ericsson.com`

Examiner: Per Runeson, `per.runeson@cs.lth.se`

Abstract

In rapidly changing markets, there is a need to be able to quickly adapt to the changes and continuously release new software updates. This puts a high focus on reducing the time needed to ensure the correctness of the previously tested functionality, while still maintaining a high level of quality. In this thesis, we conducted mixed method research to develop and evaluate a tool and a process to reduce the amount of time spent on regression testing at ASSA ABLOY Global Solutions. This was accomplished by employing the Analytic Hierarchy Process (AHP) in combination with fuzzy values for the evaluation of test cases, where one of the main criteria employed is code coverage in combination with customer usage information. Our approach reduced the number of test cases needed to reach 90% of the total decision coverage of the product source code from 258 to 63 (75.6%), corresponding to a reduction of test execution time by 54%. Furthermore, when instead measuring decision coverage of the source code actually executed in a solution deployed at a customer, we obtained a 94.6% reduction of test cases needed to reach 90% of the total decision coverage. We were also able to discover possible redundancy in the testing and the need for additional test cases to cover untested code. This resulted in the creation of a maintenance process based on the information found from our tool. We believe that our work on using advanced test prioritization in combination with code instrumentation is a first important step toward more efficient software testing at ASSA ABLOY.

Keywords: Test Case Prioritization (TCP), Code Coverage, Fuzzy Analytical Hierarchical Process (FAHP), Regression Testing, Multi-Criteria Decision Making (MCDM)

Acknowledgements

Special thanks to my three supervisors, Per Nordbeck at ASSA ABLOY Global Solutions for all the support and for inspiring me to always strive for more. Markus Borg from LTH/RISE for the invaluable and continuous support throughout the duration of this thesis, for always being there for discussions and feedback. Sahar Tahvili from Ericsson AB, for also supporting me along the way and always being there for all my doubts and questions, as well giving invaluable feedback on my work.
I would like to extend an extra thanks to all my colleagues at ASSA ABLOY Global Solutions for all the help I received along the way and always being there to discuss different problems, solutions and thoughts I had along the way.

Contents

Chapter 1
Introduction

1.1 Background

Software development is a very competitive business where there is a need to be able to quickly adapt to the changing circumstances in the market. To enable this, a lot of the software development that is conducted today is done with an agile approach, developing the products in shorter iterations with continuous interaction with the customers during development. This makes it easier to adapt to the changing needs of the customers and reduces the time needed to release products to the market.

One of the key components when performing iterative development is to integrate the changes made to the code as soon as they are tested, i.e. Continuous Integration (CI). CI is a development practice that requires integration of the code into a shared repository as soon as possible [30]. CI is commonly used in practice to handle rapid software changes. When changes are made to projects an automated regression testing suite is run to check that no new faults are introduced to the code [18]. In a survey on 442 developers, 70% found that employing CI helped them find bugs earlier and made them less worried about breaking the build [18].

For every increment of new functionality or alterations of previous functionality added to the existing system, there is a risk of introducing regressions in the previously tested code. In order to deal mitigate this, companies do not only test the new additions to the code but also test previously tested functionality to ensure the correctness of said functionality. This is done to make sure that the addition or alteration to the code did not introduce new faults to the system. The practice of retesting code to ensure the correctness of previously tested functionality is called regression testing and is a vital phase of software development to build trust in that the software works as expected [5].

In addition to running the automated test suites for each commit, it is common to employ further testing before the release of new software. Some of the tests involved in release testing might not be possible to automate and as such must be done manually.

As the employed test suites grow they tend to increase in complexity and could consume excessive amounts of time and resources for execution and maintenance of the test suite [6][34]. This could nullify the benefits of reusing old test suites instead of creating new ones. Rerunning all test cases could become infeasible as it may consume excessive amounts of time due to the increasing complexity of a growing test suite. Thus it would be beneficial to be able to select a subset of test cases and/or re-prioritize the execution order to quickly detect possible faults in the code [18]. The problem at hand is to decide which test cases to run, and in what order they should be executed, such that we can still ensure the correctness of the functionality and as quick as possible detect possible faults.

There are numerous ways and metrics aimed to measure the adequacy of a test suite. A common metric is the number of requirements covered by the test cases in the test suite. Another one is the structural coverage, i.e the amount of code covered by the testing suite [7] [19] [12]. A way to test the quality of a test case is with mutation testing, where faults are introduced into the source code to see if the test cases are able to catch them.

There are various methods and algorithms to employ for the prioritization of test cases based on either a single or multiple criteria. In more complex scenarios it could be hard to properly judge the relative priority of a test case based on a single criterion, and these algorithms are generally used to only maximize the structural coverage. Many legacy systems tend to contain a considerably large amount of code that is rarely or never executed during usage of the system. This means that it is hard to really gauge which test cases are actually exercising something that is used.

In this thesis, we explore using a Multi-Criteria Decision-Making(MCDM) Decision Support System (DSS) for the prioritization of test cases in regression testing. One criterion we explore in the MCDM is usage data from instrumented code deployed at a customer site, i.e., data on which parts of the source code actually is executing in the field. Our goal is to increase the efficiency of the regression testing practices at ASSA ABLOY Global Solutions.

1.2 Case Description

This thesis was conducted at ASSA ABLOY Global Solutions (AAGS), which is the global leader in door opening solutions. AAGS was founded in 1974 and has been an industry pioneer and world leading brand since then, AAGS develops security solutions for a wide range of scenarios, e.g. hotels, cruise ships, elderly care, and student housing [3].

AAGS needs to be able to quickly respond to changes in the market or based on the needs of their customers, and for this agile development methods are employed in the development at AAGS. AAGS also employs CI in their development, such that they could quickly detect possible faults and adapt to possible changes. In accordance with CI, AAGS employs automated regression testing for each commit to the shared repository during development. The release process at AAGS also includes manual testing as well as running beta tests at chosen beta test sites.

1.3 Purpose

The purpose of this master thesis is twofold, first to develop a new tool for prioritization of test cases, based on acquired usage data. And secondly a process tailored for regression testing at AAGS. The intermediary goals for achieving this are listed below:

- **G1** Examine current regression testing processes
  - How is regression testing performed?
  - Is there any prior work done for optimization of regression testing activities?
- **G2** Collect Test Case information
  - Collect coverage statistics of both manual and automated test cases
  - Collect test case execution time and current priority
  - Collect customer code usage
- **G3** Prioritize test cases
  - Find out how the collected data could be used in prioritization of test cases
  - Find which criteria are best suited for the prioritization of test cases at AAGS
  - Find out how data regarding existing hardware- and software configurations can be used in combination with data regarding what is really used, to prioritize what to test and in which order.
- **G4** Implement G3 in the CI pipeline
  - Develop a tool that extracts this data and sets the priority of test cases.
  - Suggest how this information could be integrated and used in existing processes.

1.4 Contributions

This thesis contributes to AAGS by reducing the amount of time taken for regression testing, which in turn will save both money and resources when developing new features, fixing bugs or adding functionality to existing systems. In addition to reducing the amount of time needed for the regression testing practices, this thesis also serves as a way to assert and enhance the quality of the testing activities at AAGS. This is done by proposing a process of how the results of this thesis could be used to review the existing test cases and find out what is missing when testing. A tool for the prioritization of test cases was developed during this thesis and the information from the tool could be used when evaluating the test suite to increase the quality and efficiency of the testing. The results of this thesis are not tailored specifically to AAGS but can serve as a general purpose method for automating prioritization of test cases when performing regression testing. The thesis also serves as a basis for further research into regression test optimization, automation of test case prioritization and the use of user data to correlate which parts of the system that should have higher priority.

1.5 Outline

The rest of the thesis is structured as follows, Chapter 2 contains the background to the problem where we go through all the topics that are important for this thesis. And we present work that is related to the work presented in this thesis, chapter 3 contains the research methodology used for this thesis. In chapter 4 we present the results of the case study, chapter 5 contains the design science outcome and in chapter 6 we present the results from the static validation of our proposed solution. Chapter 7 consists of the discussions, threats to validity and future work, chapter 8 concludes the thesis.

Chapter 2
Background

In this chapter, we elucidate the topics central to understanding the work presented in this master thesis. The chapter also serves as the foundation for the development of methods employed in his thesis.

2.1 Agile Software development

The Agile Alliance defines the agile methodology as [1]:

> *The ability to create and respond to change. It is a way of dealing with, and ultimately succeeding in, an uncertain and turbulent environment.*

IEEE defines agile in software development as [21]:

> *Software development approach based on iterative development, frequent inspection, and adaptation, and incremental deliveries, in which requirements and solutions evolve through collaboration in cross-functional teams and through continuous stakeholder feedback.*

As one of the core principles in agile software development is to be able to as quickly as possible deliver a working solution, adaptability is considered a vital concept. To achieve this, agile software development premiers being able to respond to changes over creating detailed and long term plans. In addition, agile software development prioritizes having working solutions over comprehensive documentation. This is coupled with a close connection to customers and focusing on individuals and interactions over processes and tools [1][4].

In conventional development, regression testing is performed at the end, which allows for more time to perform testing. In agile software development, regression testing takes another form; it is performed at the end of each increment of the project. This means it is crucial that it does not delay the next increment too much, as it would counteract some of the benefits of employing agile methods [41]. When working with rapid releases, there is a need to narrow the testing scope and focus on the test cases that are most important due to time constraints [31].

To mitigate the threat of delaying the release of a product or increment, regression testing in agile development is often automated to an as high degree as possible [41]. However, there are some scenarios that need to be manually tested due to various reasons e.g. it could be infeasible to automate some tests, and safety-critical functionality may need to be ensured manually [24]. Due to the time-consuming nature of manual testing, there is a risk of delaying increments if the testing is inefficient. Therefore the regression test suites need to be maintained such that they are kept up to date and not clogged with test cases that are considered redundant and new additions to the code are covered.

2.2 Code Coverage

Code coverage is a white-box testing[1] measurement of the degree to which the code has been exercised during run-time, and is often used as an adequacy measurement, where companies do not release software which has not been tested until they have reached a certain degree of code covered. There are numerous applications of code coverage, e.g. finding out if you have tested enough, testing what you intended to test and debug purposes.

Coverage based metrics are some of the more common metrics used when measuring the adequacy of a test suite [19]. A program or procedure with a high coverage percentage signifies that it has had more of its source code exercised during testing. Which in turn implies that is has a lower chance of containing undetected software bugs compared to a program or procedure with a lower test coverage percentage.

There are several types of code coverage, Figure 2.1 depicts the main coverage criteria employed when evaluating to which degree a test or test suite has exercised the source code [30].

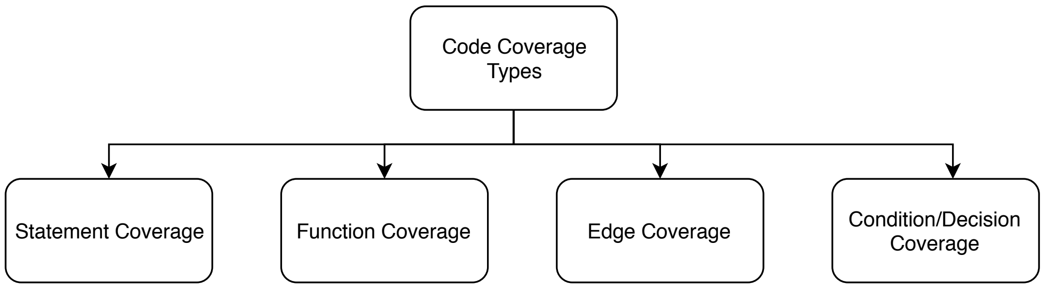

Figure 2.1: Code Coverage types

The Function coverage criterion controls if all functions or routines in the code have been executed, Statement coverage assesses if all statements have been executed, Edge coverage is about the path taken during execution, checks if every edge in the control flow graph has been evaluated. Condition-Decision coverage appraises if every branch of the control structures such as Boolean expressions have been evaluated, i.e. to true and false [19] [16] [30].

[1]white-box testing is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. (see Glenford [30])

The most commonly used metric is statement coverage even though it is usually considered to be the least effective in finding faults, edge/branch coverage is considered to subsume statement coverage as it also covers all the branches of the program. And Condition-Decision Coverage subsumes edge/branch coverage as it also covers each sub-condition [19] [30].

When employing code coverage as a way to measure the adequacy of a test suite, caution must be taken as code coverage is naive. Code coverage only shows that the region of the code have been visited, not how and in what way. A high level of code coverage does not equal a high test quality, thus code coverage should be used in combination with other methods [13].

2.3 Code instrumentation

In order to be able to measure the code coverage the information regarding what parts of the code that is being exercised needs to be gathered. This could either be done manually by analyzing the code and calculating the coverage, but it is more common to use tools that does this automatically. These tools apply code instrumentation when evaluating the code coverage of the system under test [32]. Code instrumentation is the practice of adding extra instructions to the code for monitoring purposes. These instructions serves as measuring probes which enable the collection of runtime information but do not affect the behavior of the system other than the performance [20]. With code instrumentation, it is e.g. possible to monitor a product's performance, diagnose errors, write trace and coverage information about the covered regions of the code. Instrumentation of the code could be performed on either on source code or on byte code and could be performed either statically or dynamically [32].

2.4 Regression Testing

Regression testing is a way to verify that previously existing functionality works as intended and to elevate confidence in the correctness of the altered system. This is accomplished by re-testing the previously tested functionality [2][34][36][11]. Furthermore ensuring that any modification that is done to a certain part of the system should not result in the introduction of new errors to the unaltered system. Some of the common goals of regression testing are preventing fault slippage and ensuring the correctness of the previously tested code [29][33].

There could be a multitude of reasons to why performing regression testing is necessary, we've listed some examples below [2] [33]:

- Changes made to requirements that result in the modification of the code.
- Additions to functionality.
- Changes to the environment e.g. changing database.
- When there is a performance issue fix.
- When there is a defect fix.

In conclusion, whenever there have been any changes to the environment or the system itself there is a need to perform regression testing to build confidence in the correctness of the functionality in the system.

2.5 Regression Test Selection (RTS)

The intent of test case selection is when given a test suite $\boldsymbol{T}$ to find a subset $\boldsymbol{T'}$ which still meets the requirements of that given test suite for the corresponding program $\boldsymbol{P}$. Rothermel et al. [33], defined the (RTS) problem accordingly:

Given: The original program P, the revised version of P, P' and a test suite, T.
Task: Determine $\boldsymbol{T'} \subseteq \boldsymbol{T}$, for the modified version P'

Test case selection is divided into two different directions, to either select all the available test cases for the given system, i.e. the Re-Test all approach, or only select those that cover sections that are affected to any degree by the changes made to the code [25]. Retesting all test cases might not be an appropriate method when the system grows and the number of functions and code expands, due to the increasing amount of resources spent during execution of the test suite [27].

2.6 Test Case Prioritization (TCP)

Test Case Prioritization serves as another method for increasing the efficiency of a given test suite. In contrast to Regression Test Selection, it does not run the risk of removing or omitting important test cases. Instead, the intention of TCP is to schedule test cases such that higher priority test cases are executed ahead of lower priority ones according to some chosen criterion [11][34][5]. The purpose of applying TCP is to achieve a goal earlier in the testing, e.g. a high of coverage as soon as possible or running previously failed test cases first [11].

The test case prioritization problem was formally defined by Rothermel et al. [36] as:

Given: T, a test suite, PT, the set of permutations of T, and f, a function from PT to the real numbers
Task: Determine $\boldsymbol{T'} \in \boldsymbol{PT}$ such that $(\forall T'')(T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')]$).

Where $\boldsymbol{PT}$ represents the set of all possible orderings of T, f is a function when applied to any of the orderings yields an *award value* that one can utilize to select the most efficient ordering.

2.7 Hybrid Approach

To further increase the efficiency of a test suite, a combination of methods could be employed. A hybrid approach is a combination of both RTS and TCP in an effort to further improve the efficiency and to reduce the cost of the test suite. Several researchers are working on such approaches and have proposed numerous algorithms for them [23] [36]. It still runs the risk of omitting important test cases when selecting test cases, but if done properly the test suite could gain a major increase in efficiency compared to no prioritization or only using one of the methods.

2.8 Analytic Hierarchy Process

When considering test suite optimizations methods, it is important to establish the relative worth of each test case based on what the goal of the optimization is. Depending on the situation it could suffice with employing a single criterion for the evaluation, but in more complex scenarios it could be hard to properly judge the priority of a test case based solely based on one criterion [40] [38] [15].

When applying more than one criterion for the evaluation of alternatives, the test suite optimization problem becomes a Multi-Criteria Decision-Making problem (MCDM). In MCDM problems, one must first define the relative importance of the different criteria used for the comparison, such that one can properly weigh one attribute against another attribute for the evaluation of test cases [38]. When dealing with MCDM, it is common to apply a Decision Support System (DSS) to help with solving the MCDM problem [10].

The Analytic Hierarchy Process (AHP) is a DSS for MCDM problems, where AHP decomposes the decision-making problem into a hierarchy such that the decision-maker can focus on a limited number of items at a time [22] [38]. AHP is considered a powerful tool for calculating weights to solve an MCDM problem, and is especially well suited for complex decisions that involve the comparison of decision elements which are difficult to quantify [40] [22].

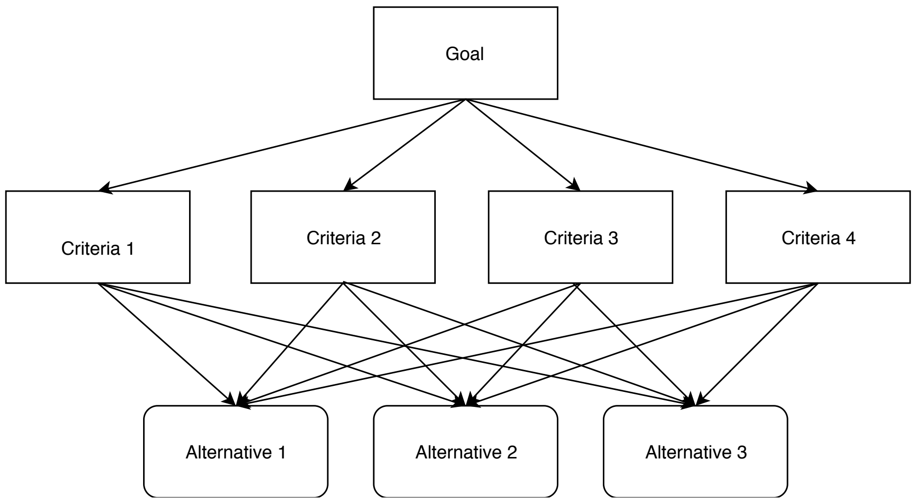

Figure 2.2: Structure of AHP

As can be seen in Figure 2.2, the AHP hierarchy consists of three levels, i.e. the goal, criteria and alternatives. The levels of AHP could be regarded as phases:

1. Goal identification
2. Criteria identification and assessment of found criteria
3. Alternative assessment and application of found weights

Phase 1 consists of deciding on what the general goal is, the decision to be made. In phase 2, one must identify which criteria are important for the comparison of alternatives, and decide the relative worth of each criterion compared to the rest. The output of phase 2 is a set of weights to be applied to the assessment of alternatives based on the different criteria. In phase 3, the order of importance is decided by first assessing each alternative based on each criterion individually and then applying the found weights.

The calculation of the weights for the criteria is performed by developing a pairwise comparison matrix for the criteria. In the matrix, the importance of each criterion is judged in relation to every other criterion.

The pairwise evaluation is performed with the help of a 9-point scale, an example is illustrated in table 2.1. The results of the pairwise comparison are stored in the comparison

Factor	Factor weighting score			Factor
	More importance than	Equal	Less importance than	
C1	9 8 7 6 5 4 3 2	[✓ 1]	2 3 4 5 6 7 8 9	C2
C1	9 8 7 6 [✓ 5] 4 3 2	1	2 3 4 5 6 7 8 9	C3
C2	9 8 7 6 [✓ 5] 4 3 2	1	2 3 4 5 6 7 8 9	C3
C2	9 8 7 6 5 4 3 2	1	2 3 4 [✓ 5] 6 7 8 9	C4
C3	9 8 7 6 5 4 3 2	1	2 [✓ 3] 4 5 6 7 8 9	C4
C4	9 8 7 6 5 4 3 2	1	2 [✓ 3] 4 5 6 7 8 9	C1
C..	… … … … … … … …	…	… … … … … … … …	C..

Table 2.1: Pairwise comparison

matrix as illustrated in table 2.2.

Factor	C1	C2	C3	C4
C1	1	1	5	3
C2	1	1	5	3
C3	1/5	1/5	1	1/3
C4	1/3	1/3	3	1

Table 2.2: The resulting comparison matrix

Furthermore, to make sure that the evaluation of the different criteria does not contradict each other, a consistency analysis is performed. This is done through first calculating the Consistency Index (CI), which is the measurement of the degree of inconsistency. Where $CI = (\lambda_{max} - n)(n - 1)$, λ_{max} is the principal eigenvalue of the comparison matrix and n is the number of elements.

2.9 Fuzzy Values

When making decisions in a complex environment it is often hard to put an exact number on the relative importance of one alternative compared to the rest. A way to mitigate this is by employing fuzzy sets instead of discrete values to make it easier for the decision maker to judge the alternatives [22] [40]. This allows for more room for error when making comparisons and is better suited for when it is hard to set precise quantified values.

A triangular fuzzy number (TFN) can be defined as a triplet of values, M=*(l,m,u)* where *l,m,u* are real numbers and *l* indicates low bound, *m* is modal and *u* represents a high bound [40].

Fuzzy number	Description	Triangular fuzzy scale	Domain	$m_A(x)$
9	Very High	(7,9,9)	7 ≤ x ≤ 9	(x - 7)/(9 - 7)
7	High	(5,7,9)	7 ≤ x ≤ 9	(9 - x)/(9 - 7)
			5 ≤ x ≤ 7	(x - 5)/(7 - 5)
5	Medium	(3,5,7)	5 ≤ x ≤ 7	(7 - x)/(7 - 5)
			3 ≤ x ≤ 5	(x - 3)/(5 - 3)
3	Low	(1,3,5)	3 ≤ x ≤ 5	(5 - x)/(5 - 3)
			1 ≤ x ≤ 3	(x - 1)/(3 - 1)
1	Very Low	(1,1,3)	1 ≤ x ≤ 3	(3 - x)/(3 - 1)

Table 2.3: The Fuzzy Scale of Importance

Using table 2.3, we can translate the linguistic values used to evaluate the alternatives to triangular fuzzy numbers (TFN).

2.10 Related Work

We identified the following areas to hold the highest relevance to the work presented in this thesis:

- Code coverage based test case selection and/or prioritization
- Decision support systems for test case evaluation
- Multi-Criteria Decision-Making

The purpose of TCP is to order test cases in an order that would satiate the goal of the prioritization with least amount of cost, commonly done by weighing different costs against values. However, it is common to first select a subset of the test cases for the prioritization.

There are several studies on how to increase the efficiency of a test suite by selecting and/or prioritizing test cases. A lot of these involve techniques that are based on applying a single criterion for the evaluation, but more and more techniques are based on multiple criteria instead [15].

Li et al. performed an empirical comparison of five different metaheuristics algorithms for test case prioritization based on code coverage [28]. Beena and Sarala [5] proposed a technique for combining TCS with TCP based on code coverage, where they first select which tests to keep and then in which order the selected test cases should be executed. The proposed approach was deemed very effective in reducing the cost and time spent on regression testing.

Wong et al. identified TCP as a single objective optimization problem [45]. They ranked test cases based on their increasing cost versus additional coverage. Ahlam et al. performed test case prioritization for regression testing based on Ant Colony Optimization [2]. Beszdes et al. performed code coverage-based regression test selection and prioritization in WebKit, where they applied a change based selection of test cases and then prioritized them for further minimization [6]. Through various studies where code coverage has been used as a single criterion for prioritization of test cases, Rothermel et al. [36] [35] and Elbaum et al. [11], identified branch coverage as the most critical coverage criterion.

When testing large and complex systems, it could be hard to correctly estimate the importance of a test case based on a single criterion, as there could be multiple factors affecting the efficiency [40]. When considering the cost and value for a test case, there could be different views of what is considered as costs or value [15]. Examples of costs could be, e.g. test execution time, test setup time or simulation costs, examples of values could be, e.g. code based coverage or non-code based coverage such as requirement coverage [15].

Multi Objective Regression Test Optimization (MORTO) is considered more suitable for complex scenarios as it considers multiple criteria for the evaluation of test cases [15]. However, adding multiple criteria for the evaluation of test cases adds another problem for the evaluation. In order to apply these criteria, one must first judge the relative importance of each criterion compared to the rest.

When facing MCDM problems, it is common to apply a DSS for the evaluation of different alternatives as judging the alternatives based on multiple criteria could become rather complex [10]. Wang et al. performed test case selection using multi-criteria optimization; one of the criteria employed was statement coverage, where they proposed a greedy algorithm to solve the multi-criteria optimization problem [44]. Walcott et al. proposed a time-aware test case prioritization where they balance time and code coverage [43]. Zhang et al. presented a similar approach to the test case prioritization problem [46]. Kabir et al. compared the DSS AHP to FAHP [22], where they found that the application of fuzzy values proved to be convenient when making decisions in a fuzzy environment where there could be deviations in the decisions of the decision makers. This makes it better suited in environments where it is hard to apply crisp values for the decisions.

Tahvilli et al. proposed a Multi-Criteria Decision Making(MCDM) approach to the prioritization of test cases, where they chose to apply FAHP as a DSS for the evaluation of test cases [40]. Their work established that it is viable to apply FAHP for the prioritization of test cases and found that:

> *Our approach enables to rank test cases based on a set of criteria and is particularly necessary when there is a limitation (due to resource constraints) in selection and execution of test cases for a system and it is not possible to run all the test cases.*

They also found that the approach is not limited to any specific set of criteria:

> *In different systems and contexts users can have their own set of key test case properties based on which prioritization is performed.*

Tahvili et al. performed an industrial case study where they selected test cases based on test case dependencies, they then prioritized the selected test cases based on multiple criteria. Their chosen method for the prioritization of test cases was based on using FAHP as a DSS [39].

The work presented in this report is based on the solution proposed by Tahvili et al. [40], where we employ FAHP as a DSS for the prioritization of test cases. However, whereas Tahvili et al. employed requirement coverage as one of the main criteria for the evaluation, we instead focus on the amount of code that is covered by each test case. Our approach also takes the information regarding what parts of the source code that is executed by the customer.

And uses the found information for the code coverage evaluation of the test cases, to increase the priority of the test cases that are covering something that is used by the customers.

Chapter 3
Research Method

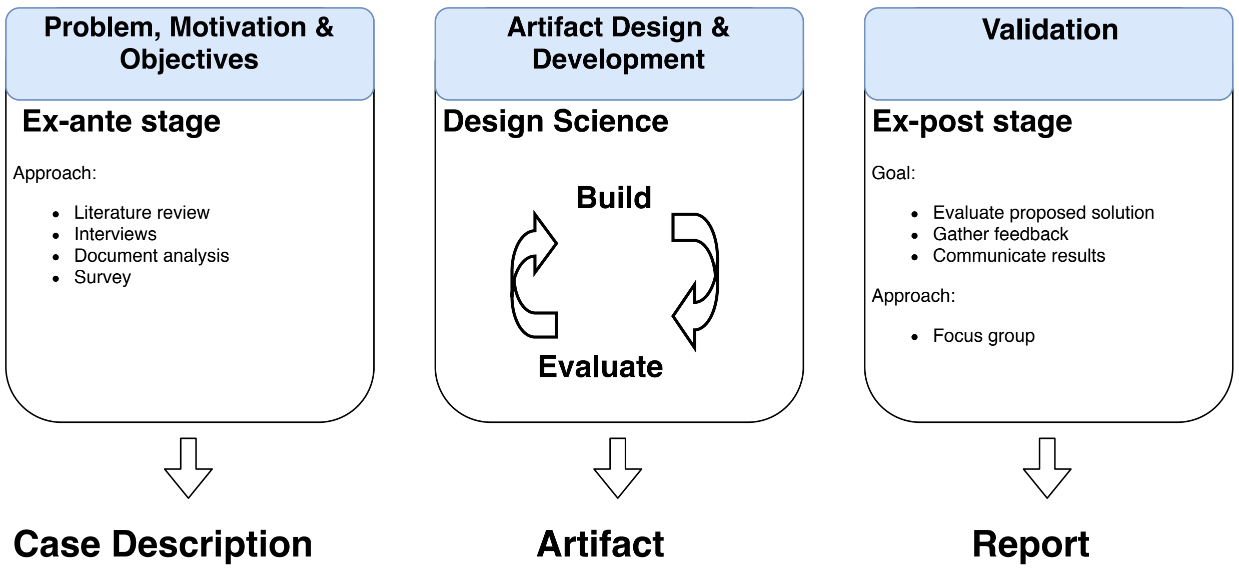

Figure 3.1: Overview of the research methodology used for this thesis

The overall structure of the work in this thesis is based on the model created by Costa et al. [8], where the method is comprised of three stages, see Figure 3.1.

Each stage in the chosen model produces some results, the ex-ante stage produces a case study report; this is where we prepare for the design science stage by reviewing literature, what kind of information that is available at AAGS and gather information from the testing department. In the finishing steps of the case study, we propose a possible solution based on the found information. The design science stage is the main stage for the development, the results of this stage are the artifacts that will be used by AAGS when they perform regression testing. In the final stage, the ex-post stage, we perform a static validation of what was produced in the design stage, both as a way to evaluate our work as well as the feasibility to employ the artifacts in the daily work of AAGS testing department.

An overview of the work conducted for the development of the tool and proposed process can be seen in Figure 3.2, where the case study covers the initial study and information gathering which is then used for the development of a tool and a process in the design science stage. The implementation of the tool and process and continuous evaluation of said tool and process belongs to the design science stage, and the evaluation of the work conducted for this thesis is done in the ex-post stage, see Figure 3.1.

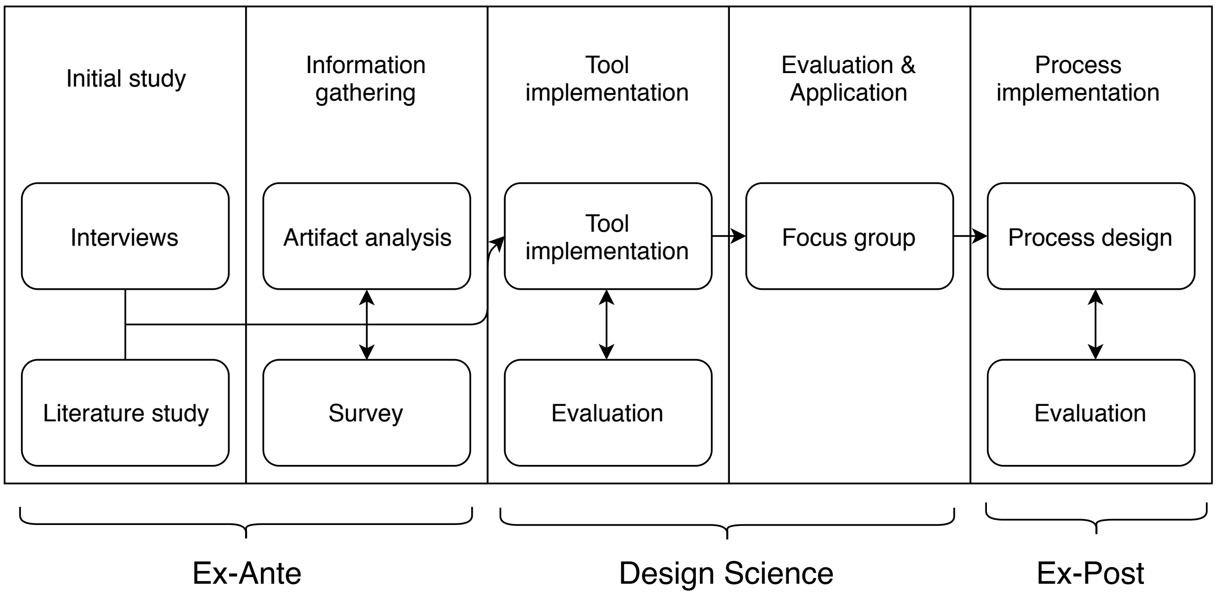

Figure 3.2: Overview of the work conducted

3.1 Ex-ante: Case Study Research (CSR)

CSR is considered as a suitable research methodology for software engineering research as it studies contemporary phenomena in its natural context. But even though CSR is considered flexible, good planning is crucial for its success [37].

A plan for CSR should contain the following elements [37]:

- Objective - what to achieve?
- The case - what is studied?
- Theory - frame of reference
- Research questions - what to know?
- Methods - how to collect data?

- Selection strategy - where to seek data?

We started by examining previous studies on regression testing to get a better understanding of the problems and challenges related to regression testing, as well as possible criteria that could be employed when evaluating the test cases, test processes and selection of test cases. However, there is no guarantee that the found information is directly applicable for AAGS, to ensure the problem relevance and applicability of the found information for AAGS specific needs. We need to dive into the specific situation at AAGS, to find the problems and challenges of AAGS regarding regression testing, and information that could be used when evaluating the testing efforts of AAGS, as well as understanding the overall idea and thoughts on the currently employed strategy for regression testing.

For this, we conducted interviews to gather information from the testing department regarding the current strategy, perceived problems and challenges related to regression testing. After the interviews, we analyzed the artifacts from previous test runs as well as other information pertaining to the testing efforts of AAGS to find out what information could be found and used for the development of a new process. This was then followed by surveying the testers with a questionnaire regarding the performance of test cases based on specific criteria.

3.1.1 Definition

The main objectives of this case study were to examine the current prioritization strategy of AAGS, gather information regarding the regression testing practices and gather the information that could be used when designing a solution for increasing the efficiency of the regression testing at AAGS. This was done as a preparatory step such that we could develop a tool that sets the priority to the test cases based on the available information.

Development process at the company

The main focus for the work done during this thesis is AAGS software system Visionline along with the components in that ecosystem, the Visionline system was created under another name DC-One in 1997 and got its current name in 2006. As the system has evolved over time, more and more functionality has been added, and with this, new test cases have been added to cover the new functionality. The creation of test cases is based on the view of every major component of the system, which means that there are several test cases that probably cover the same parts of the system. As there are several parts of the system, AAGS has developed a test suite for each of the major components, and even though they serve different purposes they are still a part of the same system and are tested with a similar approach but with different acceptance criteria. Another problem when testing for AAGS, is that it has become harder to actually know what is used by the customers, and as such knowing what should have a higher priority when testing. In order to increase the probability to find potential issues that would affect the larger portion of customers, there is a need to know what is actually used and what is more commonly used.

AAGS employs an agile methodology in their development and has three major releases planned for each year, as well as a possible minor release if the need should arise. For each release of the software, the previously tested functionality needs to be retested such that no regressions are introduced into the system. When performing regression testing AAGS use unit tests, automated tests with robots and manual testing. The bottleneck for AAGS when performing regression testing is manual testing, which is deemed as time consuming.

Ideally, AAGS would like to rerun all test cases to ensure the correctness of the system, but rerunning the whole regression testing suite has become too time consuming and thus costly. Rerunning all test cases in AAGS test suite would consume four weeks of work for one tester, thus AAGS either needs to wait for four weeks or allocate more resources to shorten the amount of time needed. In addition to possibly delaying a release, this also affects the feedback loop for the developers as it takes more time to find possible faults, fix them and rerun the failing test cases again.

Research questions

The research questions for this exploratory case study were:

> **RQ1** How is regression testing done at AAGS? What are the perceived problems/challenges?

In order to get a clear view and a deeper understanding of the regression testing practices at AAGS as well as perceived problems and challenges, we need to find out how and by whom regression testing is performed today. This serves as a base for understanding the prioritization needs of AAGS in addition to finding out the problems related to regression testing at AAGS.

> **RQ2** What is the current regression testing strategy?

To understand how we could increase the efficiency of AAGS regression testing activities, we need to investigate how AAGS currently works with regression testing.

> **RQ3** What kind of test case information is available? How can the found information be used for the evaluation of test cases?

In order to be able to prioritize test cases according to some criterion, we need to investigate what kind of data are available and what data we could use for the evaluation.

Unit of analysis

The unit of analysis is the regression testing activities in an agile, incremental development process for complex legacy system software development. The unit of analysis in the study is specifically regression testing performed by one function group at AAGS.

Methods and selection strategy

We chose to gather information from multiple sources, and as such, we employed three different methods for the collection of data. This was done in order to limit the effects of one interpretation of one single data source, and possibly strengthen the validity of our findings if the same interpretation could be drawn from multiple sources [37].

We first employed interviews to gather information regarding the current regression testing practices of AAGS, their prioritization strategy, and perceived problems and challenges related to regression testing. We then analyzed the documents related to the testing practices of AAGS in order to collect data regarding what could be used when evaluating the testing efforts of AAGS as well as individual test cases. We finished the collection of data with a questionnaire regarding the evaluation of test cases based on the found data and possible criteria that could be employed for the evaluation of test cases.

3.1.2 Interviews

The interviews were designed with a semi-structured approach, to allow for flexibility and possibility to add questions based on the answers from the interviewees [37]. The reason for conducting these interviews was to determine how regression testing is currently performed, what the current strategy is based on and map out possible challenges and problems perceived by the testers.

The structure of the interview guide consisted of background information about the interviewee, overall testing approach, views, and understanding of the current strategy. Possible challenges and problems related to regression testing and overall thoughts on the regression testing practices at AAGS, the questions used during the interview series can be found in Appendix A.

When selecting participants for the interviews we chose to focus our efforts on the persons involved in testing at AAGS, such as testers, test developers and test managers. In order to find out if there were any general methods employed for the regression testing practices at AAGS, we also chose to interview two testers that were involved with another product.

The process of conducting the interview consisted of the following steps [37]:

1. Create a consent form and interview guide
2. Conduct interviews and record the discussion
3. Transcribe the recording
4. Analyze the results and draw conclusions

As transcribing is a time consuming process, we decided to limit the interviews to 15 minutes, leaving one and a half minute per question. Which should be long enough to answer the questions without superfluous details and discussions regarding the questions.

3.1.3 Document analysis

We analyzed various documents related to the regression testing practices of AAGS to find what kind of data that were available and what could be used when evaluating test cases.

When selecting which documents to analyze, we chose to focus our efforts on documents relating to the test cases used in the regression testing, as well as any document detailing the regression testing practices at AAGS. The primary documents chosen for the analysis were the test case specifications and test case tracking tables, where all information regarding the test cases and historical outcome of the test cases were kept.

We analyzed the documents with a combination of the hypothesis-generating approach and hypothesis-confirming approach [37], where we had some hypothesis from the interviews and literature which we wanted to confirm as well as possibly creating new hypotheses based on the found data.

3.1.4 Survey

After reviewing the literature, related work and what kind of data that could be found regarding the test cases, we chose to employ a survey as a method of gathering subjective evaluation of the performance of the test cases based on the expertise of the testers. The criteria chosen for the evaluation were based on findings from previous work done by Tahvili et al. [39]. These criteria have shown good results when employed for increasing the efficiency of the testing at Bombardier tech. We employed the survey as a way to gather further information regarding the performance of the test cases based on the expertise of the testers. Each tester evaluated each test case based on the found criteria. The evaluations was done with the help of linguistic values instead of numbers. This was done in order to make it easier for the testers to evaluate the test cases, as it could be hard to put exact values on the performance based on the chosen criteria. The evaluation of the test cases was based on the five point scale of fuzzy values detailed in section 2.9, an example of the questionnaire can be seen in table 3.1.

Test Case ID	Time efficiency	Cost efficiency	Fault detection probability	Requirements Covered
T10001	Medium	Medium	Medium	1
T10002	Low	Medium	Medium	1
T10003	Low	Low	Low	1
T10004	Low	High	Medium	1
T10005	Medium	High	High	1
T10006	Medium	Low	Medium	1
T10007	Very high	High	Low	1
T10008	Low	Low	Low	1
T10009	Medium	Medium	High	1
T10010	High	High	Low	1

Table 3.1: Questionnaire for evaluation of test cases

3.2 Design Science Research (DSR)

Design science creates and evaluates artifacts intended to solve identified organizational problems, where the artifacts created when employing design science can vary depending on the problem. Design science is done with an iterative approach, continuously building and evaluating the artifacts until it satisfies the requirements and constraints of the problem it was intended to solve, see Figure 3.3 [17].

The design science research methodology of this thesis is primarily based on Hevner's guidelines regarding design science in information system research, see table 3.2 [17].

To properly explore the problem relevance, see guideline 2 in table 3.2, we conducted an exploratory case study in the ex-ante stage of this thesis as shown in Figure 3.1. The exploratory case study also served as a way to find out what could be used for the design and

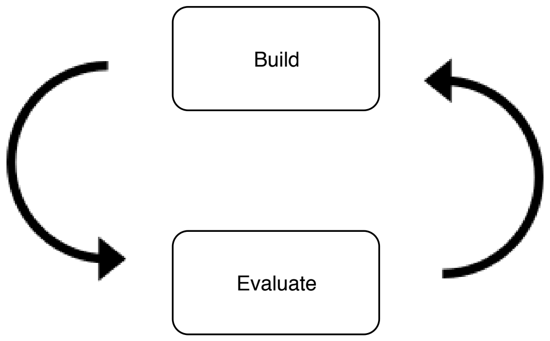

Figure 3.3: Design science development cycle

Guideline	Description
1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Table 3.2: Hevner's guidelines for design science.

implementation of a tool that could prioritize test cases based on the found data. In addition to finding out what was available, we also gathered additional data from the testing department of AAGS based on the information found from the literature study. In accordance with guideline 3, we continuously evaluate the proposed tool and process, by employing focus group meetings and having frequent interactions with the testing department of AAGS. When performing DSR, the research contributions must be clear and verifiable, see guideline 4 in table 3.2, which is done by thoroughly presenting the research approach and results. We meet guidelines 5 and 6 through the exploratory case study, where we study previous work as well as analyze the available data for the proposed solution.

When employing design science as a methodology, the work needs to produce a viable artifact in some form as well as be communicated effectively, see table 3.2 guidelines 1 and

7. The work presented in this thesis resulted in the creation of a process for automating the evaluation and prioritization of test cases when performing regression testing. For the evaluation and prioritization of test cases, an implementation of the proposed solution was created. The design of the resulting process and implementation of the tool were based on the exploration of the problem areas as well as previous work done in the field of regression testing.

The artifact consists of two parts, the main part is an implementation of a tool that could evaluate test cases based on several aspects and artifacts. The second part is the process of employing the tool, how they can utilize the findings of this thesis in the daily work of testing as a way of increasing the efficiency of testing efforts.

3.2.1 Build

All information gathering and examination of the current regression testing practices were done in the exploratory case study, and a possible solution was constructed based on the found information.

In the design science stage we focus on building a DSS tool for the evaluation of test cases based on the data gathered in the exploratory case study. The information gathered in the exploratory case study mainly relates to the testing practices of AAGS, test case performance as well as information regarding previous test runs. In addition to the evaluation of the test cases done by the testers in the case study, we also need to gather information about what parts of the software is used by the customers and what parts of the code that is covered when testing, i.e., structural coverage.

When correlating the testing done at AAGS with what is actually used by customers we can find out which test cases that hold a higher priority when testing, as they are covering something that is being used by the customers. As AAGS employs automated testing, we also need to examine what is covered by the automated tests, such that we do not elevate the priority of a test case based on how much it covers only, but instead how much new code that is covered. As shown in Figure 3.4, we want to find the test cases with the highest amount code that is being used by the customers but has not already been covered by automated test efforts.

The proposed solution can be summarized by the following steps:

1. Gather code coverage data from the testing environment and customers, then analyze the correlation between test case code coverage and user code coverage

2. Create a DSS tool based on FAHP for the prioritization of test cases

3. Create a process for how the DSS tool can be used at AAGS

In the first step, we need to gather code coverage data from each individual test case involved in the regression testing at AAGS in addition to the automated test coverage. To find out how the testing of AAGS compares to the usage of the users we also need to gather information regarding what parts of the code that is actually executed in the field.

In order to collect information regarding what is exercised in the code, we need to employ code instrumentation to the source code, such that we can find out what is actually exercised. After collecting code coverage from testing and usage from the customers we need to analyze

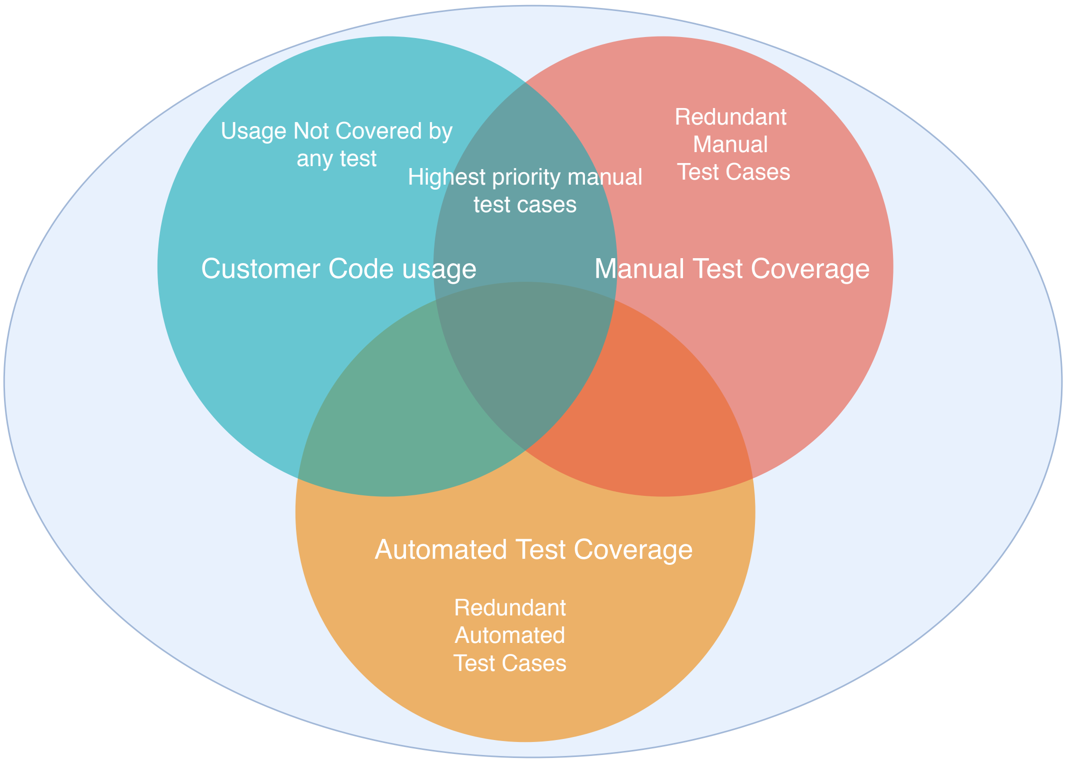
Usage Not Covered by any test
Highest priority manual test cases
Redundant Manual Test Cases
Customer Code usage
Manual Test Coverage
Automated Test Coverage
Redundant Automated Test Cases

Figure 3.4: Code Coverage of the System Under Test

the found information and determine how much of the system each test case covers and how much of the usage they cover. This could be summarized by the following steps:

1. Collect the information regarding what has been executed during run time
2. Find out which parts of the source code that has been executed during run time
3. Extract the information and store it for further analysis.

The extracted data can then be used to compare the customer usage of the system with the test coverage, such that we can find how much code each test casea covers that is also used by the customers.

In the second step, we develop a tool where we apply AHP as the DSS to solve the MCDM problem when evaluating the test cases, that combines the information found in the first step with the information from the document analysis for the prioritization of test cases. In the final step, we create a process of how the tool could be used by the testing department of AAGS to increase the efficiency of their testing efforts. The final step is performed through a focus group meeting, where we discuss how the tool can be used together with the testing department of AAGS.

3.2.2 Evaluate

When employing design science, the problem relevance and design of the artifact need to be continuously evaluated, as stated in the guidelines in Table 3.2 [17].

We chose to evaluate the process based on the performance of the proposed solution as well as how it was perceived by the testing department, thus incorporating both an analytical dynamic analysis and an analytical static analysis [17]. This was done in order to get a more complete view of the performance as well as the feasibility to employ the proposed solution in the existing processes at AAGS. In addition to this, we worked in close contact with the testing department such that we could adapt the solution based on their feedback.

Performance comparison

The analytical dynamic analysis is done by comparing the proposed solution to the current strategy of AAGS as well as random order prioritization as a point of reference. The comparison is compromised of two parts, first the amount of code covered per test case and the amount of code covered per time unit.

In order to properly gauge the efficiency of each method, we accumulate the amount of code covered based on time and amount of test cases needed. This is done such that each test case in the given order only adds the amount of code that is not covered by previous test cases or automated efforts. In addition to this, we compare the methods both with respect to the system wide testing as well as the customer usage of the system.

Focus group meeting

The analytical static analysis was performed by conducting a focus group meeting, where the testers could evaluate the preliminary results as well as the feasibility to use and maintain the proposed solution. This was done in order to get feedback from the practitioners to further improve the proposed solution, as to involve the testers that will be using the tool as early on as possible [14].

The meeting started with a presentation of the proposed solution, preliminary results and what was required to be able to use and maintain the proposed solution. Before the meeting we sent material to the elicited participants such as they could contemplate on their own before the meeting. This was done to increase the efficiency of the meeting as well as possibly get more input from the participants. The format of the meeting was a mix of direct questions to be discussed first in smaller groups and then in the larger group followed by an open discussion regarding the process of employing the proposed solution. The execution of the focus group consisted of the following steps [37]:

1. Prepare the material and gather the preliminary results
2. Send out the material to the elicited participants before the meeting
3. Conduct the focus group meeting and record the discussion
4. Transcribe the recording
5. Analyze the results and draw conclusions

3.3 Ex-Post: Static validation

As proposed by Gorschek et al. [14], we conducted an evaluation meeting as a static evaluation of the resulting tool and process. The intention with the static validation was to find out how the proposed process could be integrated into the release process at AAGS in addition to getting feedback on the proposed process itself. As an extension of how to use the proposed process, we also wanted to figure out how AAGS could use the found information to increase the efficiency of the testing efforts.

The elicited participants for the focus group meeting held various positions at AAGS. The format of the meeting was a directed discussion around the main points of interest followed by open discussions regarding the proposed process and integration into existing processes.

We initiated the focus group meeting with a presentation of the proposed process based on the findings from the previous meeting. The execution of the focus group consisted of the following steps [37]:

1. Prepare the material and results
2. Conduct the focus group meeting and record the discussion
3. Transcribe the recording
4. Analyze the results and draw conclusions

Chapter 4

Ex-ante: Case study results

In this chapter, we present the results from the exploratory case study that was conducted at AAGS. The purpose of the case study was to explore the problem relevance and find out what kind of information that was available for the evaluation of test cases. The outcome from the case study forms the foundation for the design science stage, and thus we gather as much information as possible regarding the performance of the test cases. Such that we know what could be used when prioritizing test cases for regression testing at AAGS.

4.1 Interviews

The purpose of the interview series was to establish how regression testing is performed at AAGS and gain insight into the possibly perceived problems and challenges related to regression testing.

We interviewed eight persons that are involved with testing at AAGS, the interviewees hold different positions but all have some relation to the testing practices at AAGS. The persons interviewed formed a nonuniform group in regards to positions at AAGS, academic background, amount of time at AAGS and work experience, see table 4.1. Due to the high variance in the background, better coverage of the company practice could be drawn from the interviews [9].

4.1.1 Observations

There are several actors involved in the testing at AAGS, and the testing is done at different levels and with different methods. The main body of testing performed at AAGS is either manual testing, unit tests or automated tests done with the help of robots. In addition to the methods mentioned above, AAGS also employs exploratory testing as much as possible to detect faults that might slip through the main body of tests when regression testing. Af-

Interviewee	Role	Experience / Years
1	Test manager	8-9
2	Test tools development lead	10+
3	Test consultant	4-5
4	Test consultant	6-7
5	QA tester	10+
6	Test developer	10+
7	Test leader	10+
8	Test developer	10+

Table 4.1: Interviewees

Priority	When	Description/Comments
1	Every release	Main/Central/Core functionality, Showstoppers
2	Most releases	Not the same showstopping effect as in priority one, Secondary functionality
3	Major releases	Less common functionality, Test all functions, "Cool to have" functionality
4	Major releases/If there is an issue	Corner cases, Custom tailored solutions, Exhaustive testing,

Table 4.2: The current testing strategy

ter performing regression testing in-house with passing results, AAGS sends their release candidate to a select group of customers for beta testing before the official release.

The most prominent problem according to the testers was coupled with the manual testing, where the main issue is that the manual testing is considered inefficient and time consuming. In addition to being time consuming, the test suite was considered old and possibly outdated, and the testers feel that a lot of test cases might be testing the same thing. The test cases in the test suite were created with a component based perspective from the beginning, where all the original test cases were created to test some functionality of the lock case component of the system. As time went on, the different components of the system got their own test suite based on their perspective of the system, even though a lot of test cases could be testing the same thing. The sentiment of the test manager, who created the test suite was that it should have been constructed the other way around, seen from a system wide perspective instead of a component perspective.

One of the issues involved was the strategy behind the current prioritization (**RQ2**). We have summarized the answers related to this question in table 4.2. The strategy pertains specifically to the manual regression testing performed at AAGS, as the automated tests are run each time before a commit to the shared repository.

We have summarized some of the perceived challenges/problems with the current regression testing at AAGS in the following list:

- Knowing if tested enough
- Manual testing consumes too much time
- Knowing if a test case is redundant
- Knowing if we are testing what customers actually use
- What is worth automating?
- Are some of the manual test efforts already covered by automated testing?

One of the problems that were deemed as extra important was knowing the quality of the testing, and if they are testing enough. This has been hard to properly gauge as they do not know what is actually exercised during testing, and what is actually used by the users. The consensus of the interview series was that the current way of testing is considered inefficient, as they feel that a lot of test cases might be testing the same or similar things.

4.1.2 Conclusions

The answer to **RQ1** is that there are several parties involved in the regression testing at AAGS, it is done both with automated tests and manually, with the addition of exploratory testing as much as time allows for. The most prominent problem found was the currently employed strategy is considered inefficient, and that the manual testing consumes too much time. The answer to **RQ2** can be seen in table 4.2, where the test cases are divided into four categories based on what kind of functionality they are testing and what priority it holds. The consensus of the interviewees was that the idea behind the current strategy for prioritization of test case was perceived as outdated and inefficient. The interviewees felt that there was a lot of similar test cases and that some even might be completely redundant. For this, there is some ongoing work to increase the efficiency of the test suite by revising the test cases involved as well as trying to figure out which ones that can be a part of the automated efforts. However, the problem when revising test cases, figuring out what is already automated, and what can be automated, is the knowledge about what is actually covered by each test case. The value of automating a test case is based on if something is gained by automating it, as there is no reason to just add test cases if they do the same thing or test something that is not important. If its already covered by other automated tests or if it does not cover anything that is actually used by the customers, then there is little value in automating it. Testing what their customers actually use was deemed as important, but right now there is no way to actually know what is being used other than asking the customers what they are using and how.

4.1.3 Limitations

As we were not allowed to record all interviews, we wrote down the answers of the interviewees that did not want to be recorded directly, the rest of the interviews were recorded and then transcribed for further analysis. After transcribing the interviews we gave the interviewees an extra chance to reflect upon their answers and add or detract something if there were any misconceptions. This was done in order to collect data from as many as possible as not all were comfortable with being recorded, and the extra chance to reflect upon their answers often led to added reflections.

4.2 Document analysis

When performing the document analysis we found that there was no document that explicitly detailed the current prioritization strategy for the unit under analysis. We discovered different documents with information pertaining to the test cases, how they were built and what they are testing. One of the documents contained test case tracking tables, which consisted of the priority of each test case, estimated execution time, results from previous runs, test identification, names, and overall comments. Another document contained test case specifications, which detailed preconditions, how to set up a test case, test instructions, the priority, test case id, name, acceptance criteria, and expected results as well as postconditions and purpose. This document also detailed the information regarding the format of the test cases, naming rules, requirements, and test objectives as well as overall purpose.

As discovered during the interviews, the documents detailed that the priority of a test case ranged from priority 1 through 4 and that each part of the system had its own set of test cases. Reviewing the test case specifications, we discovered that the steps to execute the test case were the same between several test cases, but the acceptance criterion pertained to specific components and thus differed between the test cases.

The documents pertaining to test cases, i.e. the specifications and tracking tables, were deemed useful for the creation of a tool, as they hold relevant data regarding the test cases. We did not find any information pertaining to the performance of the test cases, except the estimated time it takes to execute the test cases and the outcome history of the test cases. As there were no specified high-level requirements for the test cases, the acceptance criteria of the test cases are could be seen as the requirements instead, [26]. Each test case had one acceptance criterion instead of multiple, which implies that it would not be beneficial to employ the number of requirements covered as a criterion for ranking test cases.

4.3 Questionnaire

By studying literature we found that the following criteria have been shown to yield good results when employed for the prioritization of test cases [40] [39] [36]:

- Requirement coverage (C_1) constitutes the number of requirements covered by the test case.
- Time efficiency (C_2) is used to represent the time spent for the execution of a test case, a higher value means that the test case is less time-consuming.
- Cost efficiency (C_3) is used to evaluate the time and cost spent on implementing a test case and setting up the test environment for the execution of said test case.

- Fault detection probability (C_4) is used to specify the average probability of detecting a fault by the test case and is based on the fault-proneness of the region of code exercised by the test case.
- Code coverage (C_5) constitutes the amount of code covered by each test case.

Participant	Role	Experience / Years
1	Test Consultant	4-5
2	Test Consultant	6-7

Table 4.3: Survey participants

The first four criteria founded the base for the survey used when evaluating the test cases, an example of the survey can be found in appendix 3.1. Code coverage needs to be measured and thus it is not included in the questionnaire. When compiling the results from the survey we found that three out of the four criteria used in the questionnaire were found relevant for AAGS as well, as the answers varied from test case to test case. But the fourth criterion (C_4), requirements coverage was not deemed relevant for AAGS, as it would not yield any difference in the priority of a test case for AAGS. This confirmed our hypothesis from the document analysis, that each test case was created to test a specific functionality or part of the system, with only one acceptance criteria. Thus we decided to remove requirement coverage as a criterion as it would not add anything to the evaluation of test cases and instead just add unnecessary overhead.

We decided to employ the Condition/Decision Coverage criterion for the code coverage evaluations and comparisons. When studying related work, we found the branch/decision coverage criterion was identified to be the most important coverage criterion [36] [11] [45]. While the Condition/Decision coverage criterion is more thorough as it also covers the boolean sub-expressions [30]. When evaluating the coverage level, the evaluations could be based on both correlations with customer usage and the amount of code that is not already covered by automated testing. This is done to be able to maximize the efficiency of the test suite depending on the testing situation. This in turn means that the criterion could yield different results for the same test case depending on the scope of a specific test session.

4.4 Synthesis

The perceived problems with the current strategy for prioritization of test cases were numerous, ranging from structural problems to test suite maintenance problems. One of the most prominent problems perceived by the testing department was that the strategy was developed a long time ago and based on the perspective of the components instead of a system wide perspective. This resulted in a set of test case sets, where each set of test cases only considers their perspective of the system with no respect or regard to what is already tested by the other test sets. This in combination with little to no review of older test cases when adding on new ones or between releases has lead to an inefficient testing of the system, where the manual testing efforts consume too much time. In addition to that the testing efforts of AAGS and what is used by the customers may no longer match.

To address research question **RQ3**, we performed the document analysis and surveyed the testers regarding the performance of the test cases based on the identified criteria. In the document analysis, we analyzed several documents pertaining to the performance and historical outcome of the test cases, in addition to the test specifications detailing how and why they are run. The document analysis yielded good results in terms of finding valuable information and confirming the hypotheses from the interviews regarding the prioritization and structuring och test cases.

From the literature study, we found a set of criteria for the evaluation of test cases [39], but after reviewing the results from the survey we discovered that the requirement coverage criterion was not suitable for AAGS. This is due to the fact that each test case that AAGS employs for their regression testing only cover one criterion each. Thus it would not benefit any prioritization scheme to involve this criterion for the evaluation. To conclude we propose the following criteria to be used for the prioritization of test cases:

- Code coverage (C_1)
- Time efficiency (C_2)
- Cost efficiency (C_3)
- Fault detection probability (C_4)

4.5 Objective for the design science phase

AAGS would like to reduce the amount of time needed to ensure the correctness of previously tested functionality without increasing the risk for possible slippage when regression testing. Rerunning all the current test cases each release consumes too much time, and as the system under test is rather large and complex, it could be hard to evaluate the test cases based on a single criterion. This means that the test cases need to be evaluated based on multiple criteria to properly judge the performance of the test cases. For this we have identified four criteria to employ for the evaluation of test cases, creating an MCDM problem to solve evaluating test cases based on multiple criteria.

The evaluation of test cases based on the answers from the survey and the document analysis needs to be complemented with coverage data. For this, we need to collect coverage information from both the testing efforts at AAGS and information about the system usage of the customers such that we can find out how much each test case covers that is also used by the customers.

The main objective for the design science stage is to develop a DSS tool based on FAHP, that can prioritize the test cases based on the data found during this case study and the code coverage capabilities of the test cases. In addition to creating a DSS tool, a process of employing the tool and incorporating it into the existing processes of AAGS needs to be developed.

Chapter 5

Design science outcome

In this section, we detail the outcome of the design science stage. The chapter consists of a build and an evaluate section, where we first describe how we developed and implemented our solution and then the results from the continuous evaluation.

5.1 Build activity

Based on the findings from the exploratory case study, we decided to implement a DSS tool based on FAHP for the MCDM problem presented when evaluating test cases based on multiple criteria.

The tool collects the test case names and test case performance from the test case tracking tables found during the document analysis and the results of the questionnaire. In addition to this, we combine customer usage data with test code coverage to find out which test cases that cover the most of the customer usage. The code coverage criterion applied is the Condition/Decision Criterion, where we compare the coverage of the test cases compared to the customer usage and see how many of all the conditions, decisions and outcomes are covered. The criteria identified during the exploratory case study, see section 4.4, are then used when applying AHP for calculating weights which in turn yield the priority for the test cases when ordered according to the found weights.

An overview of the code coverage extraction and evaluation process can be seen in Figure 5.1, where we collect information from both the testing efforts of AAGS and the usage data from the solution deployed at the customer site. Each test case is executed in isolation, such that the relevant information could be extracted and stored test case by test case. Then the information collected from the testing efforts of AAGS is compared to the customer usage to find out how much unique coverage each test case covers that also is covering something that is used, see Figure 5.1.

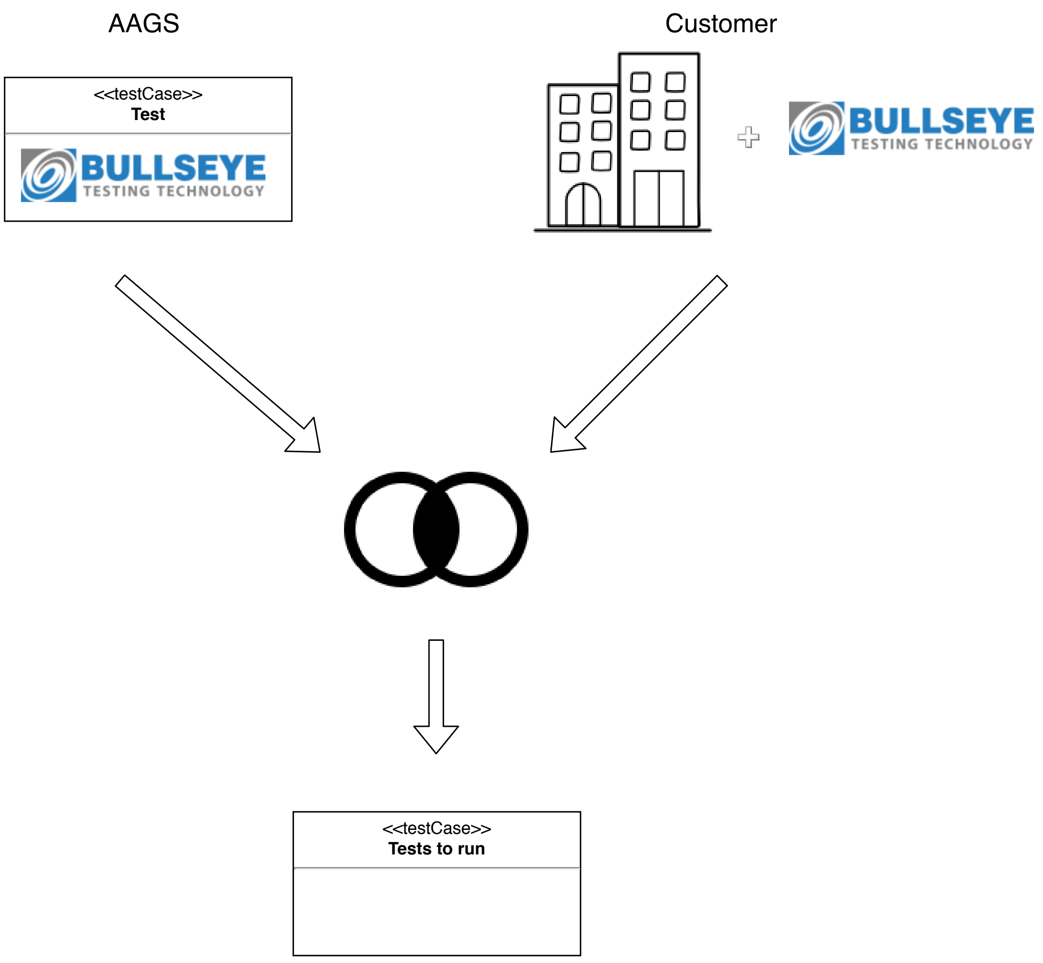

Figure 5.1: An overview of the code coverage extraction and evaluation process

5.1.1 Data extraction

In order to collect code coverage information from the testing efforts of AAGS and customer usage, we used a code instrumentation tool called Bullseye. Bullseye is a code coverage analyzer tool that writes information about what is exercised during run-time to a coverage file [42]. Bullseye is compiled with the source code during build time and then the special build with Bullseye is installed where you want to monitor what parts of the software is used. To gather information regarding what is used by the customers we deploy a version of the system that was compiled with Bullseye and collect the results after four weeks, in order to reflect the normal usage of the system.

When applying Bullseye as the code instrumentation tool, Bullseye stores all the information pertaining to what was exercised during run-time in a specified coverage file. But as the format of the information in the coverage is made for the graphical user interface of Bullseye, we first need to extract the information into a more accessible way such that we can compare the results from different runs.

This was solved by writing a Python script that searches through the coverage file and extracts the rows that were evaluated during run-time and to which extent they were evaluated. We then parse the coverage files and extract the data regarding what is exercised during run-time; this data is then stored as data-sets for analysis. The steps to collect the relevant information can be seen below:

1. Compile code with Bullseye
2. Deploy and run code for four weeks in order to reflect the normal usage of a customer.
3. Extract the information from Bullseye coverage files for parsing and collection of relevant information
4. Store the collected information

5.1.2 Test Case Code Coverage

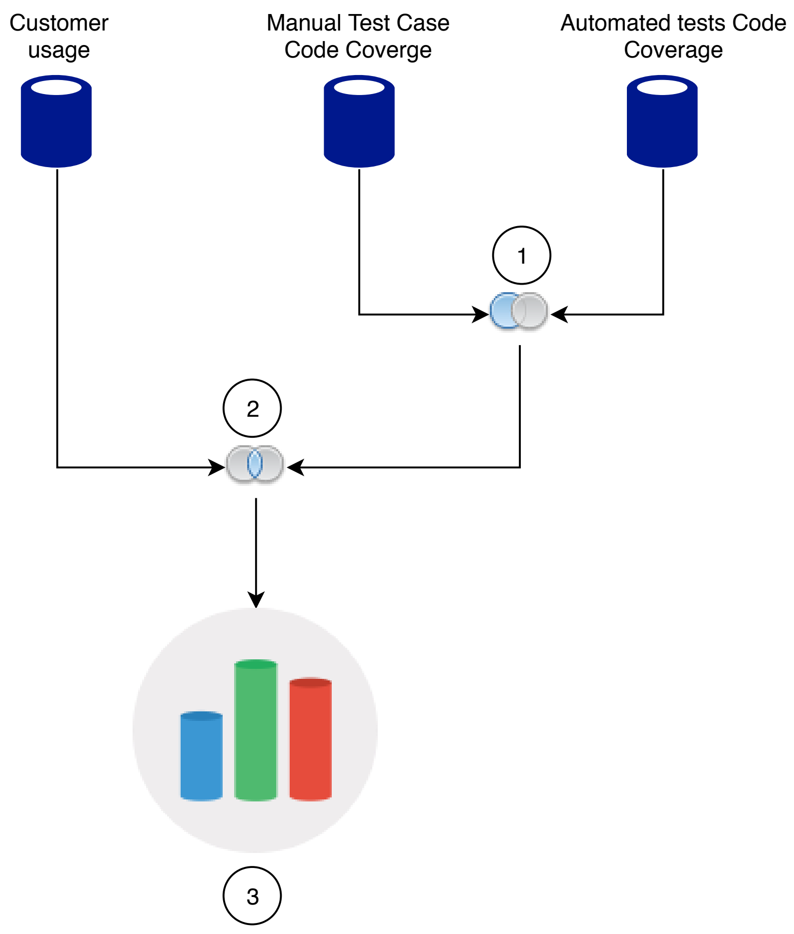

Figure 5.2: Process of calculating the amount of code covered by each test case

The code coverage capabilities of each test case are based on the amount code that is covered by each test case that is not covered by automated testing and then compared to the target, which could be either a subset of the customers or the whole system. We gather the amount of code covered by each test case by analyzing the data-sets containing coverage information that was extracted from the testing environment and users.

The process of finding out how much code is covered by each test case is illustrated in Figure 5.2, the process consists of either two or three steps depending on the scope of the testing. Step 1 consists of performing a left excluding join between a test case and the automated tests, this is done to find out how much that is covered by the test case but not already covered by the automated tests. In step 2 we perform an inner join between the customer usage and the test case to find out what is covered by the test case that is also used by the user. The second step is only applied when regarding the users if the scope for the test session is based on the whole system we skip directly to step 3, where we collect the results and store them for further analysis.

To find out how much is covered by all testing efforts, we go through a similar process, but instead of looking at the coverage of each test case one by one, we accumulate the coverage of all testing efforts at AAGS. This is performed almost the same way as collecting the individual test case information, but the left exclusive join performed at step 1, see Figure 5.2, is replaced by a full outer join.

The two methods of extracting information are done in the same step in the tool, such that we can check the performance of each individual test while finding out the total percentage that is covered.

5.1.3 Test case evaluation

The test cases are evaluated based on the four criteria that were identified in the exploratory case study, see section 4.4. In order to evaluate each test case based on the identified criteria, we must first find the relative importance of each criterion compared to the rest. This is done with the help of AHP, where we perform manual pairwise comparisons between the chosen criteria to evaluate how important each criterion is compared to another, see table 2.1 [22].

The process of evaluating test cases can be summarized in the following steps:

1. Gather all relevant information for the evaluation
2. Prepare the information for the application of criteria weights
3. Apply the calculated criteria weights to the performance evaluations of the test cases
4. Sum the rows and divide by the number of columns to collect final weight
5. Order the test cases according to the size of the final weights in descending order.

In the first step, we collect all the relevant information regarding the performance of test cases, i.e. criteria weights, code coverage and the values from the document analysis and survey.

Test Case ID	C1	C2	C3	C4
TC_1	MED	MED	HIGH	HIGH
TC_2	HIGH	MED	LOW	LOW
TC_3	LOW	LOW	LOW	MED
TC_4	VERY LOW	HIGH	VERY HIGH	VERY LOW

Table 5.1: Example of test case evaluations with Fuzzy values

Test Case ID	C1	C2	C3	C4
TC_1	5	5	7	7
TC_2	7	5	3	3
TC_3	3	3	3	5
TC_4	1	7	9	1

Table 5.2: Example of test case evaluations with Numerical values

In step two, in accordance with FAHP, we need to prepare the data such that it fits the 5 point scale in table 2.3 that was used for the evaluation of test cases, see Section 2.9. This is done by defuzzyfying the fuzzy linguistic values into their numerical counterparts and fitting the coverage data to the same scale, see Table 5.1 and Table 5.2. The defuzzyfication consists of translating the fuzzy linguistic values to their numerical counterparts, see table 2.3. We do not use the judgment of the testers to gauge the code coverage capabilities of the test cases, but instead rely on the information found from the execution of test cases and customer usage with Bullseye.

Step three consists of applying the found criteria weights on the judgments made by the testers, which then yield the weighted value which is used for the prioritization of the test cases. From the example in Section 2.8, from the comparison matrix, see Table 2.2, we get the weights detailed in Table 5.3. The resulting weighted evaluations of the test cases can be seen in Table 5.4.

Criteria ID	Criteria Weight
C1	0.43
C2	0.37
C3	0.13
C4	0.07

Table 5.3: Normalized criteria weights

Test Case ID	C1	C2	C3	C4
TC_1	2.2	1.85	0.91	0.49
TC_2	3.08	1.85	0.39	0.21
TC_3	1.32	1.11	0.39	0.35
TC_4	0.44	2.59	1.17	0.07

Table 5.4: Test case evaluations after application of criteria weights

In step 4, we calculate the sum of the weighted values, test case by test case and divide the found value by the number of criteria employed to get the final value of the test cases compared to the rest, as shown in Table 5.5. The order which test cases should be run is then found by ordering the test cases according to the final value in descending order.

Test Case ID	Weighted Value
TC_1	1.36
TC_2	1.38
TC_3	0.79
TC_4	1.06

Table 5.5: Final weighted values of the test cases

5.2 Evaluate activity

In this section, we describe the evaluate part of the design science stage of this thesis. In order to ensure that the proposed solution fulfills the needs of AAGS, a process of evaluation was established. The evaluation process consists of two parts, one part where we compare and evaluate the performance of the proposed solution to other well-known methods. The second part is where the process of working with the implementation and integrating the proposed solution is evaluated by a focus group meeting held with the testing department.

5.2.1 Performance evaluation

To gauge the efficiency of the proposed solution, we compared the performance of our solution, i.e. FAHP to the current prioritization strategy employed by AAGS. To evaluate the performance of the proposed solution compared to a third method, we decided to evaluate the performance of random order prioritization. When evaluating random order prioritization, we randomized the order of test cases and took the average performance of 100 different runs for the comparison. This was done in order to reduce the risk of extreme values, where the random order prioritization could perform either badly or well compared to the average. The main aspects of the comparisons between the different prioritization strategies were coverage versus the number of executed test cases needed.

The comparison between different strategies is performed over all test cases that are used when performing regression testing for the chosen system at AAGS. The following plots show the cumulative code coverage obtained from running the automated test suite and then the manual test cases. This was done in order to elucidate possible duplicate testing between the manual test suite and automated testing, as well as highlighting the real added value by each test case. But as we include all test cases in the comparison, the total amount of coverage will be the same for all methods. Because of this we chose three points for the comparisons, 90%, 99%, and a point close to the total amount covered.

Coverage Testing from the perspective of the entire Software System

In Figure 5.3 we can see the accumulated percentage of the total system coverage when performing regression testing versus the number of test cases needed. In Table 5.6, we present how many test cases are needed to reach certain levels of the total test coverage of the system.

The current strategy used by AAGS needs 258 test cases to reach 90% of the total test coverage, and random order prioritization needs 192 test cases. Our proposed method using

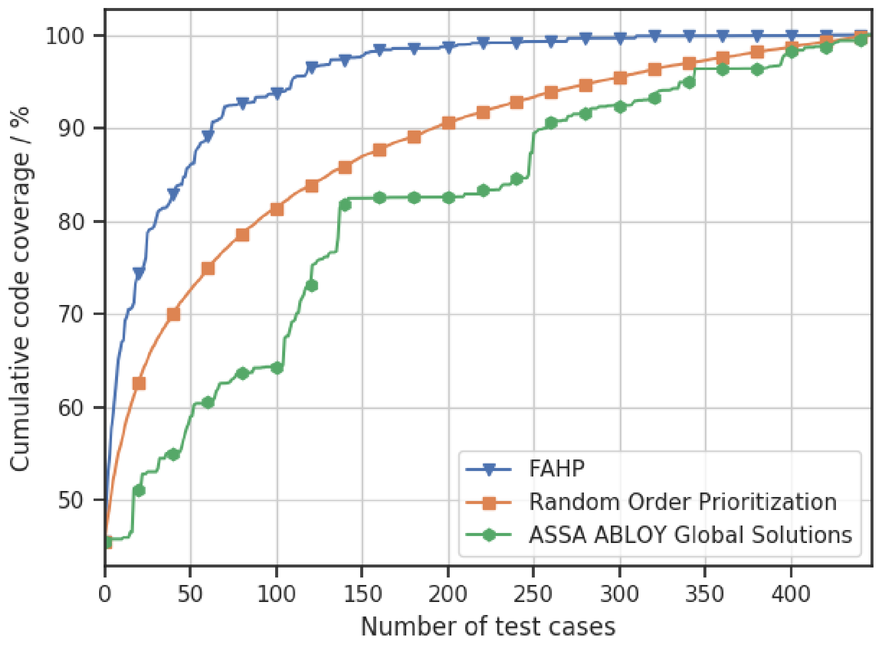

Figure 5.3: Cumulative code coverage as the number of test cases increases.

Coverage/%	AAGS	Random	FAHP
90	258	192	63
99	425	410	212
99.9	442	445	367

Table 5.6: Number of test cases needed to reach the coverage level of the system.

FAHP only needs 63 test cases to reach the same percentage. Using our proposed method, 212 test cases are needed to reach 99% code coverage, while the currently employed method and random order prioritization both need over 400 test cases to reach 99%.

Coverage / %:	AAGS	Random	FAHP
90	628	578	289
99	1225	1205	784
99.9	1284	1298	1115

Table 5.7: Execution time in minutes to reach the coverage level of the system.

In order to further evaluate the performance of the proposed solution, Figure 5.4 shows the execution time instead of the number of executed test cases. When considering the time needed to cover certain percentages, the differences are less clear. With the currently employed method, it takes the testers 628 minutes to reach 90% of the total coverage, while using our method, it takes the testers 289 minutes to reach the same level of coverage. For

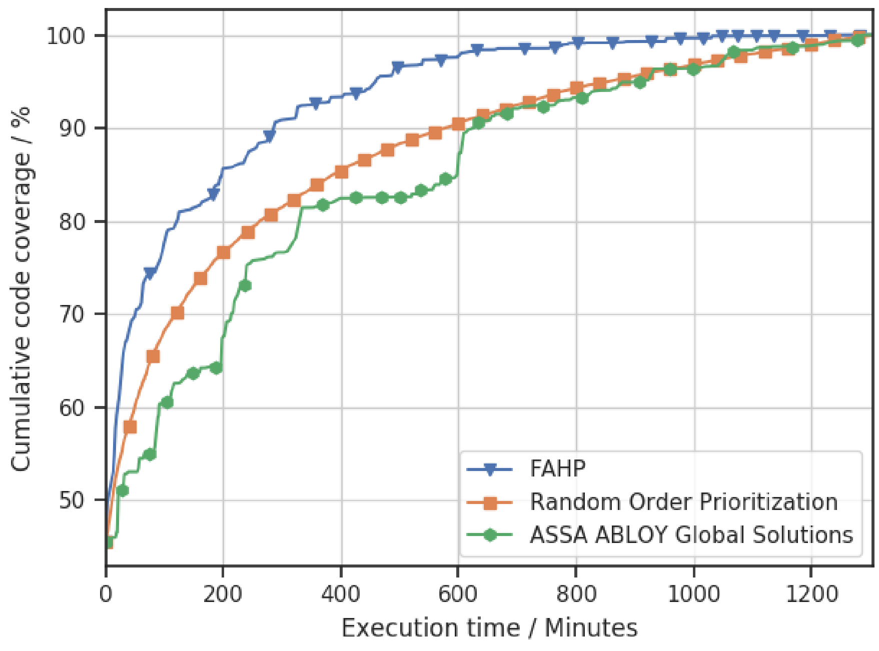

Figure 5.4: Cumulative code coverage percentage of the total test suite system coverage

reaching 99% of the total test coverage, the times are closer together, where there is only 20 minutes difference between random order prioritization and the method currently employed by AAGS. As can be seen in Figure 5.4, when over 90%, the methods perform almost the same, but FAHP still increases the level of coverage a bit quicker than the rest. Our proposed method based on FAHP needs less than half the time of both the currently employed method and random order prioritization.

Coverage testing from the perspective of what source code is executed by the customer

Coverage / %	AAGS	Random	FAHP
90	112	24	6
99	308	226	48
99.9	425	417	137

Table 5.8: Number of test cases needed to reach the coverage level.

In Figure 5.5, we can see the accumulated coverage with regards to the source code actually executed at the customer site. As shown in table 5.8, AAGS' currently employed method needs 112 test cases to reach 90% of the total coverage of the customer usage. However, random order prioritization can reach the same level of coverage in 24 test cases and our proposed method can reach it with 6 test cases. As the percentage increases, random order prioritization get closer and closer to the same performance as AAGS current method, while FAHP continues to outperform both of them. Where random order prioritization and AAGS

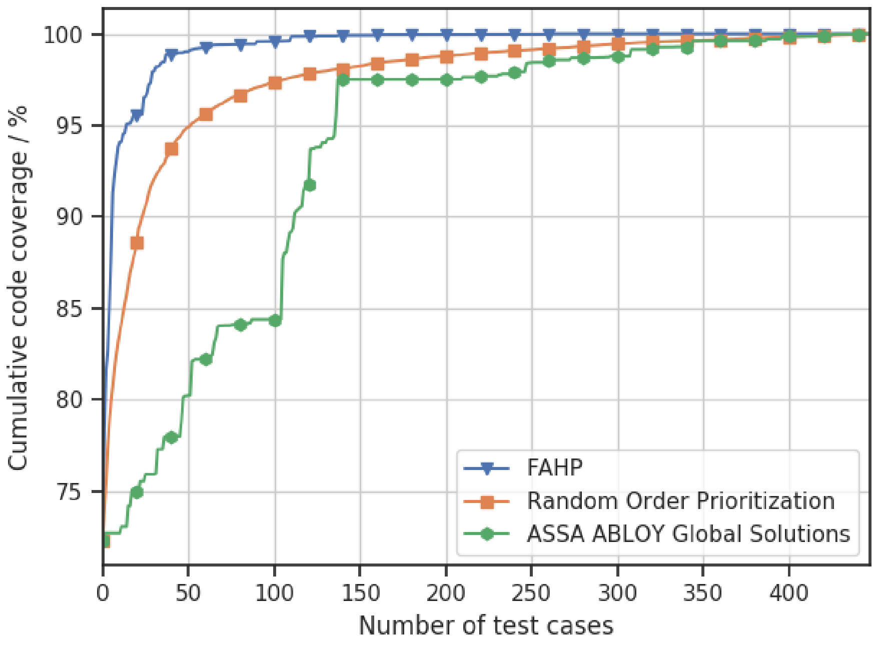

Figure 5.5: Cumulative code coverage for the source code executed by the customer

need more than 400 test cases to cover **99.9%** of the total test coverage of the customer's usage, our method can reach the same level of coverage in 137 test cases.

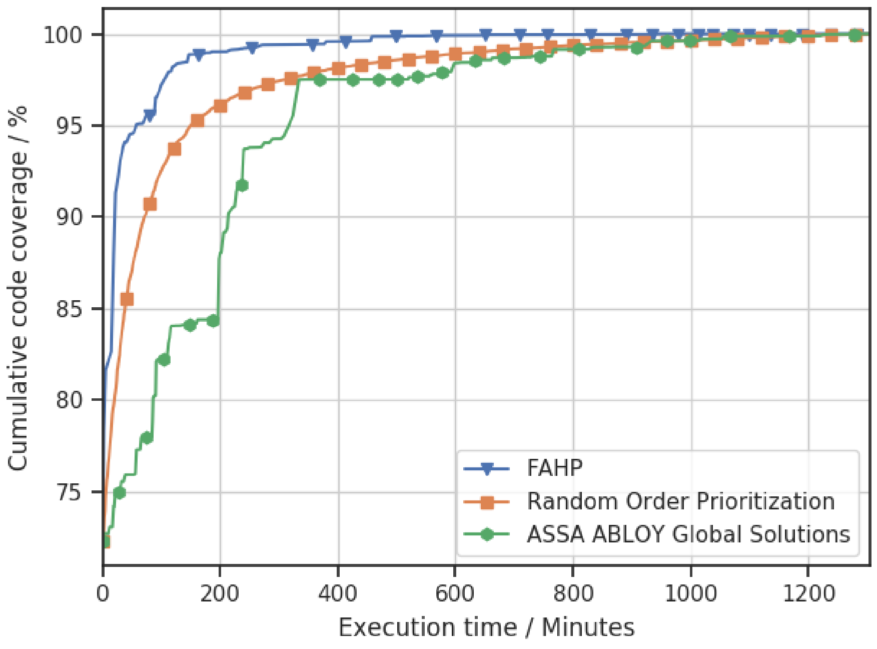

Figure 5.6: Cumulative code coverage for the source code executed by the customer

Coverage / %	AAGS	Random	FAHP
90	214	72	22
99	767	638	186
99.9	1225	1183	559

Table 5.9: Execution time in minutes to reach the coverage level of user utilization of the system.

For the comparison between the methods based on the amount of time needed to reach the chosen levels of coverage, out proposed method reaches **90%** coverage about ten times faster. And compared to random order prioritization, the proposed method takes less than a third of the time to reach the same level of coverage. However, the difference is less extreme when reaching **99%** and **99.9%** of the total test coverage of the customer's usage, but FAHP still outperforms AAGS and random order prioritization. Where FAHP needs less than half the time as AAGS and random order prioritization to reach **99.9%** coverage.

5.2.2 Focus group meeting

The purpose of this meeting was to involve the testing department as soon as possible in the design of the tool and process. We also wanted to gather feedback on the proposed solution and how the results from the tool could be used to increase the efficiency of the testing at AAGS.

Participants

In total six participants were invited for the meeting, where all the participants had a part in the testing at AAGS. All participants were present at the meeting.

Discussion

The use of a tool for the prioritization of test cases was seen as positive, as it could reduce the amount of time needed for manual evaluation of individual test cases. Using code coverage information as one of the criteria was also considered positive as it could increase the confidence in the amount of testing done before sending out a new release candidate for beta testing. In addition to increasing the confidence in the quality of the code, the participants were pleased with possibly reducing the amount of testing for each release. Having a tool that could find out how each test case contributes to code coverage was well received.

However, even though the tool was deemed as beneficial and useful, there were some doubts about trusting the tool completely, and a process for evaluating the results of the tool was discussed. The test cases that according to the tool do not cover any additional code when running in the specific order, should be manually reviewed such that the testing department knows if they have to run it at all. From the discussions, a process of handling the maintenance of the test suite emerged, see Figure 5.7. Step one in the process consists of collecting coverage data from the testing efforts and information regarding the customer usage of the system. Step two is the identification phase, where the testing department of AAGS uses the tool and looks at the output information regarding the amount of additional coverage each test

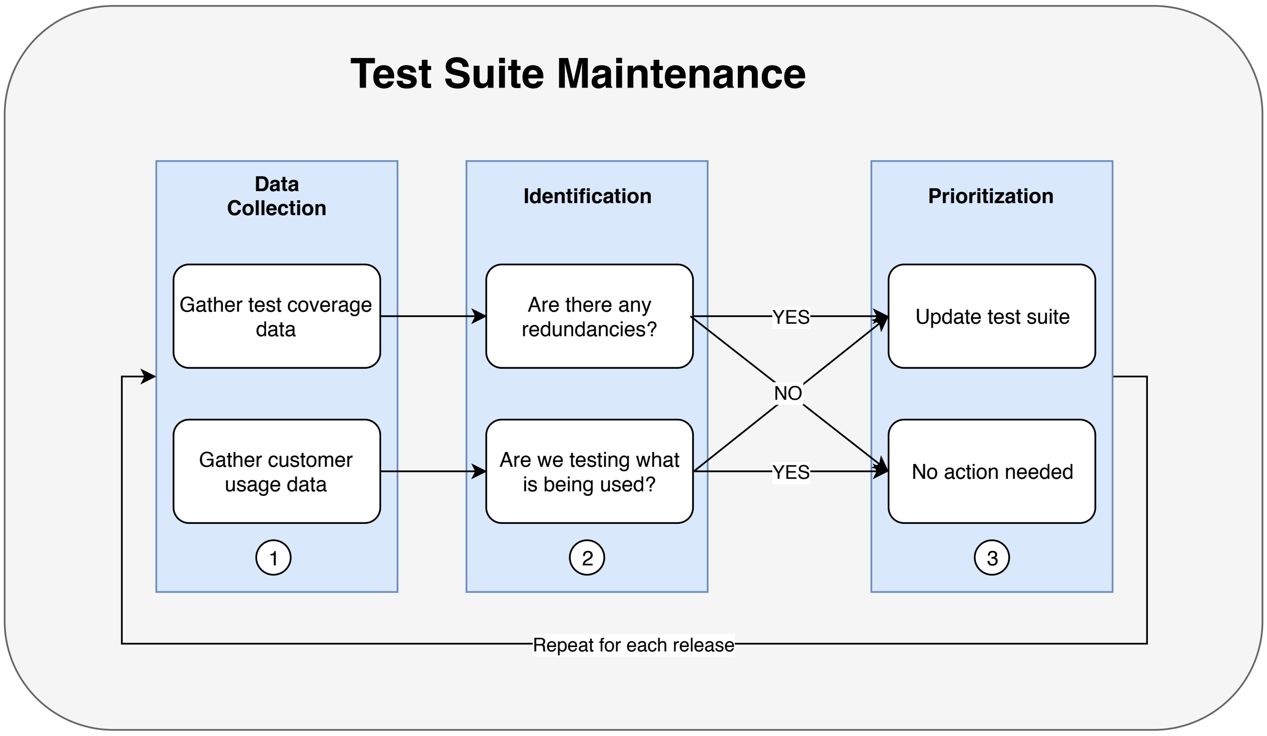

Figure 5.7: Test suite maintenance process

case adds and what is not covered by the testing efforts. From this information, they can find the test cases that are deemed redundant as they are not adding any new coverage when executed in the specified order. In the same stage, they can also find out what parts of the system that are covered, how much of the customer usage that is covered and what is not covered at all by the testing efforts. Depending on the results from stage two, a decision needs to be made. In stage three, this decision results in either an update of the test suite or no action is needed.

It was concluded that the identification stage of the test suite maintenance process needs its own process as it is an important stage to ensure the quality of the test suite, see Figure 5.8.

Where the possibly redundant test cases are manually reviewed such that AAGS does not miss important test cases that are testing nuances or any critical functionality. Nuances of test cases could yield the same results when it comes to the regions of code covered. However, there could be some differences depending on the specifics regarding e.g. the different types of cards that are used to unlock the locks, that might come in question later on that could yield different results. If the test case is still deemed redundant after the manual revision, it was suggested that the test case could be placed in storage. Such that it is not a part of the active test suite but still kept in the case of changes that affect that region of source code that could make it useful again.

When dealing with missing coverage, they decided that it would be useful to check the old test cases in the storage if any of them covers the missing code coverage. If not any older test case covers the missing coverage, AAGS must analyze if this is something that needs to be covered or if it is deemed as less important. For this, they proposed that they would have workshops with the developers to find out it is needed to cover this, and if it is, how they

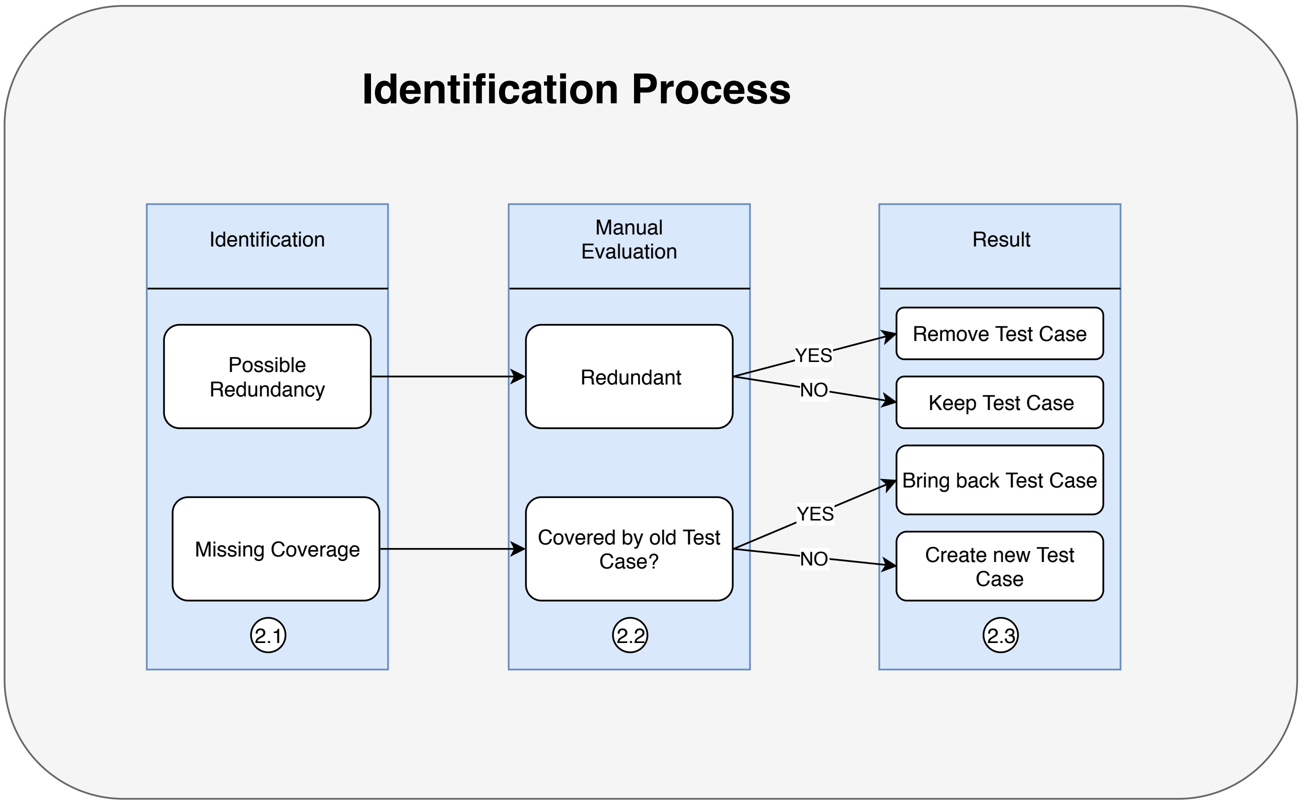

Figure 5.8: The process of identifying and handling possible issues with the test suite

should go about covering it.

5.2.3 Conclusions

We applied AHP in combination with Fuzzy values (FAHP) as the DSS for the evaluation of test cases based on the identified criteria. The proposed method performs better than both AAGS' currently employed strategy as well as the independent method of random order prioritization with regards to the amount of code covered. This holds true for both system testing and customer usage testing but is most apparent in the latter case. This is most likely due to discrepancies between the test strategy and the customer usage of the system. The cause of these discrepancies might be because of the age of the system, where a lot of functionality has been added along the way.

As can be seen in the above graphs, there exist some purely horizontal parts of the lines, which indicates that there is some degree of redundancy in the testing efforts of AAGS. This could be due to various reasons, either there exists redundancy to some degree or there could be test cases that test nuances of functionality. For such cases, the code covered remains the same, but the assertions of the test cases might be different.

The use of a tool for the prioritization of test cases was well received and the preliminary results of the tool were met with both positive and skeptic response. Where the results were seen as too good and there is a risk that we might miss important test cases if we blindly trust the results from the tool. These discussions resulted in the design of a new process of maintaining the manual test suite, and how to handle possibly redundant test cases as well as missing coverage. It was also concluded that there is a need to further investigate and discuss how the manual revision of the possibly redundant test cases and possibly missing coverage should be handled.

Chapter 6

Ex-post: Results from static validation

In this chapter we present the results from the static validation of the tool developed during this thesis and the proposed process for the maintenance of the test suite.

6.1 Process evaluation meeting

This meeting was held in order to gather valuable feedback from the engineers that will employ the proposed solution in their daily work. We started the evaluation meeting by presenting the proposed process developed from the results from the focus group meeting, see Figure 5.7. After summarizing the findings from the previous meeting, we presented the main points to be discussed for this meeting. These include the evaluation of the tool and process developed in the design science stage, and the identification stage of the maintenance process developed in the focus group meeting, see Figure 5.8.

6.1.1 Participants

In total six participants were elicited for the evaluation meeting: the test manager, two manual testers, a test developer, a test tools development lead and one test leader. Due to various reasons, only three of the elicited participants were able to participate in the meeting. The missing participants were instead told to write down their ideas, possible problems and questions regarding the topics for the meeting. In order to incorporate their ideas, we chose to read them out loud during the meeting, such that the participants could discuss their ideas and draw possible conclusions.

6.1.2 Results

The proposed tool and process were well received. The use of the tool to prioritize and evaluate the performance of the test cases was perceived as simple and straightforward. The process of employing the results from the tool for the maintenance and improvement of the test suite was seen as a step in the right direction, and a way to reduce the amount of time needed to evaluate the test cases.

By using the tool for the evaluation of the test cases, more time could be spent on manually reviewing other aspects of the test cases, to find out if and why it is important to incorporate the specific test case in the test suite. By reviewing the customer usage that was not covered by the test efforts, new test cases could be created to cover this, which would raise the overall quality of the testing.

When discussing how the information from the tool could be used, there were a lot of ideas pertaining to how AAGS could increase not only the efficiency of their testing efforts, but also the overall quality of the testing. The output of the tool could be used in the creation of new test cases that could cover the customer usage that is not covered by the currently employed set of test cases. Possibly redundant test cases could be identified by reviewing the test cases that yielded no additional coverage when run in the specified order.

The need for several workshops to analyze the results of the tool emerged through the discussions. In those workshops, the test cases that were deemed as redundant should be manually reviewed by the testing department of AAGS and then if needed by the developers as well. Furthermore it was also concluded that the possibly missing customer usage, should be reviewed by the developers to find out if it is something that should be covered. If it is deemed as important, find out how this could be covered by creating use cases for future test cases.

During the meeting we decided that the ones not present would get a chance to comment and add their understanding or ideas about the discussions and results of the meeting. For this, we compiled the results of the meeting and sent it to all the elicited participants such that the engineers not present could review the results and provide feedback. But also allow for the participants to reflect upon the findings of the meeting and add any possibly missing idea/thought.

6.2 Survey

Finally we conducted a survey as a validation of the results from the meeting and a final validation of the proposed processes and use of the tool developed during this thesis. The questions in the questionnaire were a mix of open and directed questions. The questionnaire can be found in Appendix B.

6.2.1 Participants

We sent the questionnaire to all the elicited participants, and during the course of two weeks, in total five out of six responses were received. As all respondents but one sent in their answers, we consider all relevant perspectives covered.

6.2.2 Survey Results

All the participants answered that the proposed process and tools would benefit their work to some degree. Where some feel that it would have a direct effect on their work, while others feel it would indirectly affect their work. One of the main benefits seen by the participants regarding the tool, was that it could speed up the process of regression testing without endangering the level of quality and it could highlight where they should focus their efforts. Regarding the process of maintaining the test suite there were several possible benefits if more customer usage could be collected too increase the confidence in the findings. The usage of customer usage information was seen as a key component, as it could reveal the real usage of the system, what should have a higher priority when testing and if there is a need to create additional test cases.

The test developers found that the information from the tool could benefit their work by elucidating what is already covered by the automated efforts and what remains to be automated. As the proposed solution could decrease the amount of time needed for performing regression testing, the engineers executing the manual test cases could possibly focus more on other types of testing, e.g. exploratory testing. From the perspective of the test manager, the results of the tool are beneficial as the information can serve as an additional basis for decisions regarding the testing practices of AAGS. There were some concerns regarding using the tool without caution, several of the respondents wrote that information found from the tool should be used but the limitations of the tools should be reviewed and considered when making decisions. More customer usage information must be collected, and the setup must be reviewed to increase the confidence in the findings.

We have summarized the possible benefits perceived by the participants of this survey below:

- Find possibly duplicated testing
- Find possibly untested customer usage
- Speed up the regression testing
- Increase the quality of the testing
- Possibly identify what could be automated

The possible challenges/problems can be summarized by the points below:

- Need to review the setup and what parts of the system that are considered
- Need to map out the limitations of the tool
- Need to collect usage information from a larger group customers

Chapter 7
Discussion

7.1 DSS for regression test prioritization at AAGS

The employment of a DSS for the regression test prioritization yielded good results compared to the currently employed method and random order prioritization. We first intended to compare the proposed approach to history based testing in addition to random order prioritization. History based testing relies on increasing the priority for failed test executions, and the system under test was a mature and stable system. Thus, there were hardly any failing tests between releases in the regression testing suite, which resulted our decision to remove history based testing as it would not yield much different results compared to the currently employed strategy.

If we judge the performance of our solution solely on the amount of code covered in the least amount of time, then the proposed solution would save AAGS a considerable amount of time and effort when regression testing. But as code coverage is naive, and does not consider variations of test cases that could cover the same regions of code but still could be testing different things, the results must be met with some skepticism. The results need to be manually reviewed as discussed in the evaluation.

The difference in performance is most apparent when regarding the customer usage of the system, which has not been used by AAGS before. When testing the whole system, there is still quite a difference in performance between the methods, with FAHP reaching the highest level och coverage with the least effort. But when closing in on the total system test coverage, the difference in performance between the methods are less and less extreme.

The initial work to be able to employ this method is rather time consuming, as the testers must evaluate all test cases based on the performance when considering each criterion. In addition to the collection of judgments of the testers, to gather the code coverage information from the testing efforts of AAGS the testers needed to rerun all test cases with the Bullseye compiled version. We needed to both install the Bullseye compiled version at the customer's site and then collect the results. These tasks consume a large portion of time and resources, which means that the investment for using our solution could be considered rather large, even though the potential yield could make up for it in the longer run. The ideal situation would be to collect information from the customers after each release, and rerunning all the test cases. However, as long as information is collected during the release testing most of the test suite will be kept up to date.

In addition to the initial work to apply this tool and process, there is some work that needs to be done continuously for the evaluation not to become old and stale. The test cases chosen for the regression testing needs to be executed with Bullseye enabled to update the coverage information, but the test cases that are not involved could become relevant later on. These test cases must thus be updated within some time interval. AAGS also needs to collect information from customers to be able to trust the results, and this must be done for each release of the system if the information should be kept up to date. This could be up for discussion, how often or for which releases the information must be updated, but it must be done to prevent that the information becomes outdated.

Using the tool and process for the prioritization and maintenance of the test suite can help the testers to increase the quality of the testing and reduce the time needed for regression testing. As mentioned before, the tool does not only calculate the order in which test cases should be run but also stores information regarding the individual performance of each test case and the total coverage for the given scope. From this information, possibly redundant test cases could be found, and customer usage of the system that is not tested could be identified. This could be used when performing maintenance of the test suite, to possibly reduce the time need to perform regression testing and possibly increase the quality of the testing.

7.2 Threats to validity

In this section, we list the possible threats to the validity and possible limitations of the work presented in this thesis.

- **Test case evaluations:** When gathering evaluations of the test cases for the FAHP, we were only able to collect answers from one tester regarding the whole test suite. This means that there is a risk of the evaluations being skewed by the judgment of a single source as other testers might judge differently.

- **Code Coverage Measurements:** When measuring the coverage of each test case, we only had time to run the whole manual test suite once due to a large number of test cases. This means that we were not able to filter out any possible noise in the system that should not be regarded as the standard coverage of a specific test case. This runs the risk of skewing the evaluation of test cases and possibly increasing the priority of a test case that actually has less coverage than another. This could be mitigated by rerunning all test cases multiple times and remove the oddities that do not show up in the majority of test runs.

- **Customer base:** Due to time constraints, we were not able to gather user data from a large number of customers, but instead have to rely on a single customer. This runs the risk of missing important functionalities that are not used by the customer that we collected data from. This will be a continuous effort for AAGS, and the proposed method will yield more trustworthy results as more user data is added. Until then AAGS will not remove any test case that is not a hundred percent covered by other testing efforts, such that they do not miss any important test case.

- **Code Coverage as a criterion:** The efficacy of using code coverage as a way of ensuring the effectiveness of testing is considered questionable as code coverage does not guarantee test quality [13]. We chose to use this criterion anyhow since we are able to match the code coverage of AAGS testing efforts against the usage by their customers. Thus increasing the reliability in that they at least cover what is used by most customers.

7.3 Future Work

In this chapter, we propose different topics for further research on this subject as well as possible features and development for the tool that could yield better results.

- **Collect a larger set of customers:** When regarding the customer data we were only able to collect data from a few customers, which might not reflect the needs of the entire customer base. It would increase the level of confidence in the prioritization and possible minimization of the test suite if we had data from a wider range of users.
- **Create customer type profiles:** When releasing new increments of an existing software system, it could be that the changes made to the code might not affect all customers. Thus it would be beneficial to create profiles based on the different types of customers, such that one can re-prioritize the testing efforts based on the type of customers that will be affected by the changes.
- **Extended duration of data collection:** We collected the data from the execution of instrumented code during a relatively short amount of time, but long enough to reflect the normal usage of the system. However, it does not take different seasonal behavior into consideration, where different functionality may be used due to seasonal differences in activity by the end users of AAGS customer's hotels. It would be beneficial to base the prioritization on data collected over a longer period of time.
- **Combine the information regarding coverage with specific releases:** When deploying different releases it could vary a lot which customers that will be affected by the update. To further increase the efficiency when regression testing one release could match the knowledge of what configurations are used by the different customers. Selecting a subset of test cases that cover the regions of source code that are affected by the changes or additions made could be possible.

Chapter 8
Conclusion

In this thesis we developed a tool for the prioritization of test cases based on multiple criteria. We present a process for how the tool could be used for the maintenance of a test suite. For this we applied FAHP, as the DSS for the evaluation of test cases based on the criteria. The main criterion employed in the evaluation of test cases was code coverage, where we combined information about the customer usage of the system with the code coverage of the test cases. Based on this, we identified which test cases cover the most of the customer usage of the system, which is not already covered by other testing.

Our tool performed considerably better than both the currently employed method and random order prioritization when considering the number of test cases or execution time needed to reach a high level of coverage. But as code coverage is naive, and might not consider variations of test cases that could cover the same regions of the source code but still test different things, the results must be considered with caution. To handle the results of the tool, we held a focus group meeting with the testing department to discuss how they should handle test cases that are deemed as redundant by the tool, and how to handle the customer usage that is not covered by the testing efforts of AAGS. It was concluded that this should be manually reviewed, where the testing department and the developers would have workshops where they shall decide if the test cases are indeed redundant and if/how to cover the missing coverage.

By employing the proposed solution, AAGS could possibly reduce the time taken for regression testing without reducing the level of quality when testing. This is due to the fact that the tool elucidates the information pertaining to what is used be the customers and what is tested by AAGS.

So far, we could only collect information from one customer, which means that AAGS needs to collect information from more customers before deciding which test cases that should hold a lower priority when testing against what is actually used by customers. In the meantime, the results could be used to guide reviews of test cases that do not cover any additional code when testing the whole system, i.e., the potentially redundant test cases identified by our tool.

Appendices

Appendix A
Interview guide

1. What is your role at AAGS?
2. What is your background?
3. What is your view on regression testing?
4. Do you see any problems or challenges related to performing regression testing?
5. How would you describe your capability to perform regression testing today?
6. How do you verify that you have tested enough?
7. How would you describe the current approach to regression testing at AAGS?
8. How would you describe the different levels of prioritization that is used today (1-4)?
9. Anything else that would provide further insight into the current regression testing practices at AAGS?

Appendix B
Questionnaire

- What are your thoughts on using the tool for the prioritization of test cases?
  - Do you see any possible challenges/problems?
- What are your thoughts on the usage of customer usage data when prioritizing test cases?
  - Do you see any possible challenges/problems?
  - Any benefits?
- Would the tool and process have an positive impact on your work?
- What are your thoughts on the proposed maintenance process for the test suite?
  - Do you see any possible challenges/problems?
  - Could it be improved?
- How would the tool and process affect your work?
  - Any benefits?
  - Could it consume more time than it saves?
- Is there anything you would like to add?
  - Problems?
  - Challenges?
  - Benefits?

Bibliography

[1] Agile 101. https://www.agilealliance.org/agile101/, 2018. [online, Last accessed 16 February 2019].

[2] A. Ahlam, A. Khan, A. Khan, and K. Mukdam. Optimized regression test using test case prioritization. *Procedia Computer Science*, 79:152–160, 2016.

[3] ASSAABLOY. About assa abloy global solutions. https://www.assaabloyglobalsolutions.com/en/aags/com/about-us/. [online, Last accessed 16 February 2019].

[4] K. Beck, M. Beedle, and van A. Bennekum. The agile development manifesto. https://agilemanifesto.org/, 2001. [online, Last accessed 16 February 2019].

[5] R. Beena and S. Sarala. Code coverage based test case selection and prioritization. *International Journal of Software Engineering & Applications (IJSEA)*, 4, 2013.

[6] A. Beszedes, T. Gergely, L. Schrettner, J. Jasz, L. Lango, and T. Gyimothy. Code coverage-based regression test selection and prioritization in webkit. *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, page 46, 2012.

[7] E. Bjarnason, M. Unterkalmsteiner, M. Borg, and E. Engström. A multi-case study of agile requirements engineering and the use of test cases as requirements. *Information and Software Technology*, 77:61–79, 2016.

[8] E. Costa, L.A. Soares, and P.J. Sousa. Situating case studies within the design science research paradigm: An instantiation for collaborative networks. In *Collaboration in a Hyperconnected World - 17th IFIP WG 5.5 Working Conference on Virtual Enterprises, PRO-VE 2016, Porto, Portugal, October 3-5, 2016, Proceedings*, pages 531–544, 2016.

[9] B. DiCicco-Bloom and B. F. Crabtree. The qualitative research interview. *Medical Education*, 40(4):314 – 321, 2006.

[10] Umm e Habiba and S. Asghar. A survey on multi-criteria decision making approaches. *2009 International Conference on Emerging Technologies*, page 321, 2009.

[11] S. Elbaum, G. Rothermel, and A.G. Malichevsky. Incorporating varying test costs and fault severities into test case prioritization. *IEEE Transactions on Software Engineering*, 23(10):929–948, 2001.

[12] E. Engstrom, P. Runeson, and M. Skoglund. A systematic review on regression test selection techniques. *Information and Software Technology*, 55:14–30, 2010.

[13] G. Gay, M. Staats, M. Whalen, and M.P.E. Heimdahl. The risks of coverage-directed test case generation. *IEEE Transactions on Software Engineering*, 41(8):803, 2015.

[14] T. Gorschek, C. Wohlin, P. Carre, and S. Larsson. A model for technology transfer in practice. *IEEE Software, Software, IEEE, IEEE Softw*, 23(6):88, 2006.

[15] M. Harman. Making the case for morto: Multi objective regression test optimization. *2011 IEEE Fourth International Conference on Software Testing*, page 111, 2011.

[16] Hadi Hemmati. How effective are code coverage criteria? In *Proceedings of the 2015 IEEE International Conference on Software Quality, Reliability and Security*, QRS '15, pages 151–156, Washington, DC, USA, 2015. IEEE Computer Society.

[17] A.R. Hevner, S.T. March, J. Park, and S. Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75 – 105, 2004.

[18] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig. Usage, costs, and benefits of continuous integration in open-source projects. *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, page 426, 2016.

[19] Z. Hong, P.A.V. Hall, and J.H.R. May. Software unit test coverage and adequacy. *ACM Computing Surveys*, 29(4):366 – 427, 1997.

[20] F. Horváth, T. Gergely, Á. Beszédes, D. Tengeri, G. Balogh, and T. Gyimóthy. Code coverage differences of java bytecode and source code instrumentation tools. *Software Quality Journal*, 27(1):79, 2019.

[21] IEEE. Iso/iec/ieee international standard - systems and software engineering– vocabulary. *ISO/IEC/IEEE 24765:2017(E)*, page 1, 2017.

[22] G. Kabir and A. A. Hasin, M. Comparative analysis of ahp and fuzzy ahp models for multicriteria inventory classification. *International Journal of Fuzzy Logic Systems*, 1:87–96, 2011.

[23] S. Kadry. A new proposed technique to improve software regression testing cost. *International Journal of Security and Its Application*, 5(3), 2011.

[24] J.C. Knight. Safety critical systems: challenges and directions. *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, page 547, 2002.

[25] S. Kukolj, V. Marinkovic, M. Popovic, and S Bognar. Selection and prioritization of test cases by combining white-box and black-box testing methods. *2013 3rd Eastern European Regional Conference on the Engineering of Computer Based Systems*, page 153, 2013.

[26] S. Lauesen. *Software requirements : styles and techniques.* Addison-Wesley, 2002.

[27] A. Lawanna. An effective test case selection for software testing improvement. *2015 International Computer Science and Engineering Conference (ICSEC)*, page 1, 2015.

[28] Z. Li, M. Harman, and R.M. Hierons. Search algorithms for regression test case prioritization. *IEEE Transactions on Software Engineering*, 33(4):225, 2007.

[29] N.M. Minhas, K. Petersen, N.B. Ali, and K. Wnuk. Regression testing goals - view of practitioners and researchers. *2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW)*, page 25, 2017.

[30] J. Myers, G and C. Sandler. *The Art of Software Testing.* John Wiley & Sons, Inc., USA, 2004.

[31] M.V. Mäntylä, B. Adams, F. Khomh, E Engström, and K. Petersen. On rapid releases and software testing: a case study and a semi-systematic literature review. *Empirical Software Engineering*, 20(5):1384–1425, 2015.

[32] L. Nan, M. Xin, J. Offutt, and L. Deng. Is bytecode instrumentation as good as source code instrumentation: An empirical study with industrial tools (experience report). *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, page 380, 2013.

[33] G. Rothermel and M. J. Harold. Analyzing regression test selection techniques. *IEEE Transactions on Software Engineering*, 22:329–338, 1996.

[34] G. Rothermel and M. J. Harold. Empirical studies of a safe regression test selection technique. *IEEE Transactions on Software Engineering*, 26(6):401–419, 1998.

[35] G. Rothermel, A.G. Malichevsky, and S. Elbaum. Test case prioritization: A family of empirical studies. *IEEE Transactions on Software Engineering*, 28(2):159–182, 2002.

[36] G. Rothermel, R.H. Untch, C. Chu, and M.J. Harrold. Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering*, 27(10):929–948, 2001.

[37] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131, 2009.

[38] T.L. Saaty. How to make a decision: The analytic hierarchy process. *Interfaces*, 24(6):19 – 43, 1994.

[39] S. Tahvili, M. Saadatmand, M. Bohlin, W. Afzal, S. Larsson, and D. Sundmark. Dynamic integration test selection based on test case dependencies. *2016 IEEE Ninth International Conference on Software Testing*, page 277, 2016.

[40] Sahar Tahvili, Mehrdad Saadatmand, and Markus Bohlin. Multi-criteria test case prioritization using fuzzy analytic hierarchy process. In *The Tenth International Conference on Software Engineering Advances*, November 2015.

[41] D. Talby, A. Keren, O Hazzan, and Y. Dubinsky. Agile software testing in a large-scale project. *IEEE Software*, 23(4):30, 2006.

[42] Bullseye Testing Technology. Bullseye coverage measurement technique. `https://www.bullseye.com/measurementTechnique.html`. [online, Last accessed 16 February 2019].

[43] K.R. Walcott, M. Soffa, G.M. Kapfhammer, and R.S. Roos. Timeaware test suite prioritization. In *Proceedings of the 2006 International Symposium on Software Testing and Analysis*, pages 1–12, New York, NY, USA, 2006. ACM.

[44] K. Wang, T. Wang, and X. Su. Test case selection using multi-criteria optimization for effective fault localization. *Computing*, 100(8):787 – 808, 2018.

[45] W. Wong, S. London J. Horgan, and H. Agrawal. A study of effective regression testing in practice. *Proceedings The Eighth International Symposium on Software Reliability Engineering, Software Reliability Engineering, 1997. Proceedings., The Eighth International Symposium on*, page 264, 1997.

[46] L. Zhang, S. Hou, C. Guo, T. Xie, and H. Mei. Time-aware test-case prioritization using integer linear programming. In *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis*, pages 213–224, New York, NY, USA, 2009. ACM.

EXAMENSARBETE Test Case Prioritization Using the Analytical Hierarchy Process and Customer Usage Profiles for Regression Testing
STUDENT Mattias Karlsson
HANDLEDARE Markus Borg (LTH), Per Nordbeck (ASSAABLOY), Sahar Tahvili (Ericsson AB)
EXAMINATOR Per Runeson (LTH)

Prioritering av testfall baserat på kunders användning av systemet

POPULÄRVETENSKAPLIG SAMMANFATTNING **Mattias Karlsson**

Regressionstestning är en viktig process för att kunna säkerhetställa att tidigare testad funktionalitet fortfarande fungerar utan att fördröja en release av mjukvara för mycket. I detta arbete nyttjar vi multikriteria-bedömning av testfall, varav en av de viktigaste kriterierna är korrelationen mellan hur mycket av koden som täcks av tester och hur mycket som täcks av kunders användning av systemet.

I dagens snabbt förändrande marknader är det viktigt att kunna anpassa sig och kunna få ut ny mjukvara så snabbt och med så hög kvalitet som möjligt för att inte missa en möjlighet eller tappa marknadsandelar till konkurrenter. För att kunna säkra att tidigare testad funktionalitet fortfarande fungerar och inga nya fel har introducerats, utförs regressionstestning som ett sätt att säkerhetställa kvalitén på produkten.

Regressionstestning har en tendens att bli kostsam och ta mycket tid, vilket kan fördröja en release av ny mjukvara. För detta har ett stort antal olika metoder för testoptimering tagits fram, varav många är baserade på ett kriterium för bedömning av testfall. På senare tid har fler och fler metoder baserat på flera kriterier tagits fram. Det beror på att det kan vara svårt att bedöma prestandan på ett testfall baserat på ett kriterium när det kommer till större och mer komplexa system.

I detta examensarbete har vi utvecklat ett verktyg baserat på multikriteriaoptimering för regressionstestning, där ett beslutstödssystem kallat AHP nyttjas för att lösa multikriteriaoptimerings problemet för regressionstestning. För att kunna snabbt säkerhetställa att det kunder nyttjar är tillfredsställande testat så är en av de viktigaste kriterierna i vår modell är korrelation mellan testtäckning och kundanvändning av systemet. Vi jämför först hur mycket kod varje enskilt testfall täcker med det som redan är täckt av automatiserade tester och jämför sedan detta med kundernas användning av systemet. Detta gör vi för att först ta reda på hur mycket varje enskilt testfall täcker som inte redan är täckt av annan testning. Sedan hur mycket som det täcker utav det som används av kunderna, och ökar sedan prioriteten på de som täcker mest.

Förutom att reducera tiden det tar att utföra regressionstestning, kan även detta verktyg användas för att öka kvalitén på testningen, genom att använda informationen gällande vad som inte är täckt för att skapa nya testfall. Med hjälp av denna informationen skapade vi en process för underhåll av testfallen.

Resultatet visar att nyttjandet av AHP och kunders kodtäckning kombinerat med testtäckning kan väsentligt reducera antalet testfall och tiden som det tar att uppnå en hög nivå av kodtäckning vid testning.