Thesis for the Degree of Bachelor in Computer Science | 15.0 hp

MEASURING TEST COVERAGE IN EMBEDDED SOFTWARE DEVELOPMENT BRANCHES

Zaid Abed Jaser
Zar16001@student.mdh.se

Osamah Haitham Sabhan Al-Braichi
Oai16001@student.mdh.se

Examiner: Adnan Causevic
Mälardalen University, Västerås, Sweden

Supervisor: Daniel Sundmark
Mälardalen University, Västerås, Sweden

Company Supervisor: Per Erik Strandberg
Westermo Network Technologies AB, Västerås, Sweden

September 4, 2019
Abstract

Most of the hardware products today, especially those that people interact with, are controlled by software. There are many devices with inbuilt software which many people do not bother to notice. Software may be critical in terms of strict quality requirements due to failures which enable risks of endangering the production and more importantly the lives of people. The testing team of Westermo Network Technologies AB faced a challenge with not being able to identify the coverage of tested cases in software. Identifying the coverage of total executed test cases enables the possibility of knowing the potential future quality of a software. By reaching such a stage a software will likely not suffer from failures due to higher quality, therefore the lives of people and the production at stake will not get harmed. The essential problem is that test cases are both tested and skipped, which makes it more difficult for the company to identify what has been actually tested and skipped. The purpose of this thesis is to identify and calculate the coverage of test cases, the process was mainly to understand the essence in the identification of executed test cases. The ethical aspect of putting people's lives at stake is what inspired us to investigate software failures. Software is typically tested before a release, therefore our investigation was to research testing process of software. When investigating test result over many days and test systems we developed a coverage calculator system which helps Westermo to decide and determine the release of tested software, either the software test result is acceptable for a release or not.
# Table of Contents

1. **Introduction** .................................................................................................................. 3

2. **Background** ................................................................................................................. 4
   2.1 Testing ......................................................................................................................... 4
   2.2 Testing of Embedded systems ...................................................................................... 4
   2.3 Testing Execution Analysis ......................................................................................... 4-5
      2.3.1 Allocation Time ................................................................................................. 5
      2.3.2 Company’s Testing System and Branches ......................................................... 5-6
   2.4 Test Execution Coverage ......................................................................................... 6

3. **Related Work** .............................................................................................................. 7-8

4. **Problem Formulation** ............................................................................................... 9-10

5. **Method** ..................................................................................................................... 11

6. **Ethical and Societal Considerations** ....................................................................... 12

7. **Results** ...................................................................................................................... 13-15

8. **Implementation Process** .......................................................................................... 16
   8.1 Database Analysis ...................................................................................................... 16
   8.2 Initial Prototype ......................................................................................................... 16
   8.3 Front-end Implementation ......................................................................................... 17
   8.4 Back-end Implementation ......................................................................................... 17-18
   8.5 Coverage Calculation System .................................................................................. 18

9. **Discussion** ................................................................................................................ 19
   9.1 Future Work .............................................................................................................. 19

10. **Conclusion** ............................................................................................................... 20

11. **References** .............................................................................................................. 21-22
1. Introduction

Technology has developed exceedingly throughout the past years. It is rare to find a device not being controlled by software, it is more likely to come across embedded software devices. These embedded systems play a critical role in management systems in several areas, such as industry and transport [1]. The quality requirements in these areas are very strict because system errors can lead to very dangerous incidents which are irrevocable such as loss of production or even loss of life [2]. Thus the software in these domains needs to be reliable in order to avoid failures. To address these losses the software needs to be properly tested before it is delivered and ready for use.

Valuable testing is defined by the outcome of test cases, the test cases can deliver results which can help to understand the existence of problems in a software system [3]. The results might be easily understood if the test cases were performed manually because each test is done by a person who notes the testing result, but it is harder to perceive the outcomes of an automated test system. The reason could be one of many, such as the occurrence of a non-systematic error, for example, damaged wire connections or hardware issues. But also the misunderstanding of coverage, which is important to master as a result of measured test cases performed in a system. To achieve proper software testing with acceptable results, one will need to face challenges. For instance lack of time, understanding of failed outcomes and environment availability for testing. The results assist in the understanding of what test cases have actually been tested, skipped and the determination of different decisions which are the decisions of software quality level. These kinds of challenges have to be taken into account so as not to endanger the lives of people and nor to slow down the production.

Westermo Network Technology AB faced the same challenges with software testing. The company develops robust network equipment and solutions for industrial applications and systems. Most of Westermo products run the WeOS operating system - a Linux-based networked operating system, tailored for embedded systems [1]. This software is utilized in critical areas, for that reason, software needs to be highly tested, ensuring high quality and robustness. The company’s way of testing this software is during nights for highest efficiency because the software is not being managed by employees during nights. They use branches to facilitate the detection of errors by dividing the tasks into smaller groups of developers. Branches are developed by employees, and each branch has many different test cases for each software.

Our role in this process is to ensure that nightly testing is used wisely by helping Westermo in figuring out how to allocate proper and efficient time between different development branches. Our expected outcome in this thesis is to understand the testing challenges which are faced by Westermo and to deliver a coverage measurement of executed test cases. Since testing is a very important factor done before the software is released, we want to make our thesis worthy with solutions for those companies which have similar challenges.
2. Background

This chapter provides the reader with the terminology needed to understand the report work and contribution. Therefore, a brief explanation will be included in different sections.

2.1 Testing

Testing is one of the essential parts for every software and hardware because it helps to provide trustworthy products which deliver the desired requirements [4]. An important requirement that must be taken into account is the high quality of software, which lessen the risk of endangering human lives. Regarding costs, a well-tested software before a release has less risk to get damaged which avoid unnecessary costs of manufacturing. For instance, flight Boeing 737 max 8 was damaged severely due to software problems [5]. In this case, had this software been properly tested, neither unnecessary costs might have been wasted nor endangering human lives.

2.2 Testing of Embedded Systems

Embedded systems consist of both hardware and software. These embedded systems can be used in critical areas such as aerospace, military, trains, and airplanes [3]. Because of the strict quality requirements, the hardware system must deliver a reliable, stable and good portable outcome, therefore the software system has to be tested [2]. Embedded systems build process are designed to connect a software system to a hardware device [6]. The testing challenges in embedded systems are the limited hardware available for testing. The interaction between these results in different outcomes. These outcomes might result in failure due to lack of software and hardware verification. Therefore an embedded system must validate both their software system and hardware devices.

Westermo develops robust network equipment and solutions for industrial applications and systems. Many Westermo equipment run the WeOS operating system - a Linux-based networked operating system, tailored for embedded systems [1]. WeOS is developed in-house at Westermo, and development utilizes concurrent feature branches, as well as a master branch, and a release branch. Each night, many automated test cases are run for each branch, on a set of hardware test systems. Since there is insufficient time to execute each test case on every branch for all test systems, not all test cases, and test systems will be executed.

2.3 Testing Execution Analysis

Westermo stores its data of test cases in a database, the data delivers different kind of information which is helpful for the company. For example, all test cases and test systems have an identifier, which makes it easier to identify what test cases in a test system has been executed. The purpose of Westermo’s database is to gather the system test results to assemble a conclusion, where employees receive a better understanding of the outcomes. These outcomes are determined by numbers between 0 to 5 which are stored in the database. The numbers stand for a different type of results depending on the process of the tested system. Essentially numbers 0, 1 and 2 are tested, 3 and 4 can not be executed and 5 are skipped due to lack of time. See Table 1 for a detailed explanation.
### Test Cases Result

<table>
<thead>
<tr>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/Pass</td>
<td>The test verified that the software seems to work.</td>
</tr>
<tr>
<td>1/Fail</td>
<td>The test discovered that the software seems to be incorrect.</td>
</tr>
<tr>
<td>2/Invalid</td>
<td>There was probably a problem in the test framework, or the test environment or the software under test had serious problems.</td>
</tr>
<tr>
<td>3/Unmappable</td>
<td>The resources needed to run the test (hardware in the test system) are not present – we cannot start the test.</td>
</tr>
<tr>
<td>4/Unloadable</td>
<td>The test does not exist. Sometimes when one move tests around, the test framework cannot locate a test.</td>
</tr>
<tr>
<td>5/Skipped</td>
<td>There was no time left to run this test, but it could have been executed if there was more time, or if this test case had a higher priority</td>
</tr>
</tbody>
</table>

Table 1: Database outcomes.

#### 2.3.1 Allocation Time

Allocation time is important for nightly testing in Westermo. The allocated time is determined by ongoing projects and highly prioritized test systems. When this thesis was written there were more than 20 test systems and 20 branches in use at Westermo. The branches can be tested in different test systems, therefore each night a test system has a certain amount of allocation time assigned manually for each branch. For instance, test system X can test several branches at the same night, therefore 70% of the nightly time is allocated for branch B1 in system X, B2 has 20% and B3 has 10% of the nightly allocation time because of lower priority. The sum of the allocated time should be 100% in order to receive the best possible efficiency as shown in Table 2.

<table>
<thead>
<tr>
<th>Branches B</th>
<th>Test System X</th>
<th>Test System Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>70%</td>
<td>20%</td>
</tr>
<tr>
<td>B2</td>
<td>20%</td>
<td>20%</td>
</tr>
<tr>
<td>B3</td>
<td>10%</td>
<td>60%</td>
</tr>
<tr>
<td>Total Allocation</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 2: Allocation time of different development branches.

#### 2.3.2 Company’s Testing System and Branches

Westermo builds advanced test systems as shown in Figure 2. These test systems are based on cables which are connected between a computer with Linux environment and the testing hardware. These computers are used to manage the test framework. The test framework helps WeOS to receive the right branch which loads the devices before the tests execute. The test systems are used to test software in branches. The branches can execute multiple test cases which are managed by employees in Westermo. New branches may consist of new software in WeOS, which enables the possibility to create new test cases but the old test cases remain at the same branch.
2.4 Test Execution Coverage

In general, testing is executed to identify errors in a system which are later managed by developers [7]. There are functional and coverage testing, functional testing provides an acceptable system by assuring that the system satisfies the required specification. Coverage testing is different from functional because it is used to give a measurement of completeness in a system. The measurements are used as a checklist of errors detected which guides the tester to improve the software. The measurements are calculated by dividing the executed test cases by the ones which could have been executed and multiply the result by 100 to receive a result of percentage, see Formula 1.

\[
\text{Coverage} = \frac{\text{Number of executed test cases}}{\text{Total number of test cases}} \times 100
\]

Formula 1: Coverage Formula.

In section 7 we describe the coverage metric defined in this thesis.

\[\text{\textsuperscript{1} Photo by Westermo. Used with permission.}\]
3. Related Work

Westermo tests their software with automated test systems. The software does get affected by bugs, which decreases performance. In this case, the identification of what cases has actually been tested and skipped is not simple due to the lack of an overview of tested branches. According to Shahin et al. continuous integration seems to be an efficient method to test new changes in a software [8]. The method proposed by Elbaum et al. is based on signals which are implemented in all test cases, where the signals return the results of the executed tests. This method makes it possible to test the new changes in software precisely without any long delays [9]. Regarding the measurement of coverage, it is far more efficient to receive accurate coverage measurement with a continuous integration process at a unit level testing. Because the information obtained after every software change could also be the knowledge of coverage. According to Elbaum et al. this method is cost-effective at a system level testing because a continuous integration process at a unit level testing requires more test developers and more RAM hardware. However, Westermo testing process is based on continuous integration method but on system level testing, where their software changes are assembled and tested during nights. The challenge with system-level testing is not receiving an accurate measurement of coverage as fast as unit level testing, because a unit-testing is the smallest testable part of a software whereas system-level testing is the larger and more complete part.

The information we received from our research concludes that there is no solution on how to calculate the measurement of coverage in a testing process similar to Westermo’s. However, during our research, we discovered a formula which we based our solution on. According to Zhu et al. the mathematical calculation shown in section 2.4 calculates the total coverage, the challenge we faced was to receive the appropriate numerator. Regarding Formula 1 it requires a test case which is executed at least once in the numerator to fully calculate the formula [10]. Another challenge we faced is that Zhu et al. mentioned a common database outcome where it returns a Boolean which informs if the test case has been executed or not, either by returning a false or true. However, referring to these two challenges, when we investigated Westermo’s database we detected several duplicate test cases, which displays the same test cases with multiple outcomes. The outcomes of executed test cases have a different result in the database as explained in section 2.3 compared to the common database outcomes whom Zhu et al. mentioned.

A paper by Mordinyi et al mentioned that requirements coverage is a fundamental need through a software life cycle. Requirement coverage is based on several requirements which are equal to the total requirements in a test system [11]. Through traceable mapping between the requirements and the test cases a result could be generated, the results are either pass or fail. According to Mordinyi et al. the outcome is considered to pass only if all the linked test cases have passed. Referring to our research question, receiving a measurement of coverage over several test systems will be false following this method since the outcome of a requirement coverage is considered to pass only if all test cases within are acceptable. Because the failed requirement coverage will not be calculated as a measurement of coverage even if some test cases within have passed. In this case, the testers need to calculate the result of each requirement on each test system in order to get the right coverage measurement of all requirements. The solution to this problem could also be our developed system which calculates the measurement of coverage with many test systems over time. The system takes all test cases into consideration, both the executed and skipped test cases, therefore an accurate measurement of coverage is possible.

In general requirements coverage is a benefit for unchangeable software. Because for instance, Westermo has frequent changes in their software that are being tested which will also require changes in the requirements coverage. This is inefficient because changing both in the software and the requirements also means increasing the number of testers. Incorrect requirements changes may lead to
miscommunication between the testers and the requirement engineers. Such problems may lead to decreased product quality, wasted effort and delays [12].
4. Problem Formulation

Westermo company faced a challenge to understand the coverage measurement of test cases in their branches. Branches are developed by employees and each branch has many different test cases. Westermo uses branches to facilitate the detection of errors in the software by dividing the tasks into smaller groups of developers. Test cases have a task to detect the situation in a certain area of software, according to Westermo’s database each test case delivers five different outcomes which determine the status of it. Westermo developed an operating system called WeOS which is used in most of their products. WeOS is developed in-house at Westermo, and development utilizes concurrent feature branches such as feature X, Y and Z, as well as a master branch, and a release branch as shown in Figure 3. The branches are connected with Westermo’s test systems in order to receive the results of the test cases.

![Diagram of different branches of Westermo's system](image)

Each night, an automated test suite is run for each branch, on a finite set of hardware test systems [1]. Since there is insufficient time to investigate each test case on each branch for all test systems, only a subset of test cases and test system combinations will be tested for each branch. For instance, as shown in Figure 4, the test suites with the same test cases differ from day to day, some days they are approximately 80% tested and other days they are 20% tested. The problem that occurs is the measurement of coverage tested for more than one night. Because according to Westermo’s database those executed test cases can have a different outcome in the nights after the first one and further on. For instance, the outcomes displayed false results, where tested test cases were executed several times which is not only inefficient but also leads to false measurement of coverage. To solve this challenge the company needs to know exactly how many test cases have been tested during a certain period of time. Understanding this issue helps Westermo to allocate the right nightly time for the prioritized test system.
Regarding the research question, measurement of coverage seems to improve the nightly testing due to better chosen time allocation in each branch system. The measurement should be able to help identify which test system needs more time allocation for a better coverage result.

**Research question:** How to calculate executed test case coverage in network embedded systems and branches with many test systems over time?
5. Method

To identify the main challenge faced by Westermo we had to visit the company in order to fully understand the topic in discussion. Every week we visited the company to discuss the problem. We got brief explanations about the purpose of the company and how important software testing is by our supervisors. To understand how to calculate the measurement of coverage we had to do our research. In this phase, we decided to follow a scientific method to answer our research question. Through the case study method we were able to get one step closer to obtaining the goal of the thesis [14]. We followed the method and the first step was to specify and formulate the research question in order to perceive the desired goal. The second step was to gather all the necessary information we needed to answer our research question. The information we needed was Westermo’s database, therefore a case study was a relevant scientific method to follow in our work because no experimental controls or manipulations were involved. We did not have to use another database as a source to our solution, thus we were able to solve the problem by only using Westermo’s information from their database. Proceeding to the third step we searched for specific keywords to investigate the research area and not to receive and document on already solved problem. For instance, we used coverage measurement, coverage calculation, manual testing, automated testing and Coverage Embedded Systems as keywords. A subset of keywords was used in reliable databases such as IEEE Xplore and ACM.

After the research phase, we managed to find a formula which helped us to calculate the measurement of coverage. The solution was mainly based on the coverage formula but we had to do more work before receiving the appropriate data to calculate an accurate measurement of coverage. At this stage, we put a lot of time into testing different SQL queries. The querying of the database was done by a program called SQLite which manipulates with databases. The program limited our solution because the company desired an interactive system. Therefore we had to solve the problem by using other tools. We developed a user-friendly experience prototype which we followed to obtain the desired interface design. The database was thereafter uploaded locally to phpMyadmin in order to manipulate the database with other programming languages. The manipulation of the database was done by JavaScript, PHP, HTML, and CSS. At the last steps of our work, we discovered another useful feature used as an error handling to determine if the results of coverage are accurate. The feature was based on Formula 2 and 3 shown in section 7. To determine if the final solution was acceptable we presented our work to our supervisors as validation and verification.
6. Ethical and Societal Considerations

There are no limitations in terms of considering the ethical aspects of this work. The one thing that was obligated to complete was to sign a confidentiality agreement to receive the database which was given by Westermo company. The agreement was meant to keep the database secure.

Unlike ethics, there is societal consideration in our work. By answering our research question it will benefit the production of industry and transport and hopefully the lives of people at stake. The calculation of coverage improves the allocation time done manually in a system due to prioritization. By understanding what actually has been tested enhances software development, which increases the quality of a system. Poor quality might endanger the lives of people and lessen production.
7. Results

According to Westermo, the outcomes might help to determine several decisions which are based on the results of how many test cases have been tested and skipped. If a test case has been tested or skipped it informs the company of the software quality. The information confirms if the software should be either released or it needs to be re-tested. Software is released if the measurement of coverage is high enough to be reliable according to Westermo requirements. The software is not released if the measurement of coverage is unacceptable, due to lack of reliability.

There are two types of decisions which Westermo wants to make:

- A: Could we make a release of the software we are testing?
- B: How should we put test resources so that we can make decision A?

Regarding the research question, calculation of coverage supports the company to make the decisions of A and B. The measurement of the calculation system which we have developed, calculates the outcome of the test cases. The outcomes differ depending on the options available. The available options are shown in Table 3 and they are the following:

- Option 1. One branch, One test system, One night.
- Option 2. One branch, One test system, N nights.
- Option 3. One branch, M test systems, One night / N nights.

Option 1 displays for the company the measurement of coverage during one night in one branch on one test system. The common result has more skipped test cases than tested due to lack of time. A test system rarely finishes testing during one night because of many branches being tested. Option 2 is the same as option 1 but the outcome of coverage is for a longer period. Regarding option 3, one branch is tested in different test systems in either one or several nights. Our calculator system helps the company to prioritize the nightly allocated time for each test system depending on the coverage result. If the tested results satisfy the company's requirements then decision A and B is clear to make, otherwise, the branch should be retested until it meets the required measurement of coverage. To clarify these outcomes here are some examples from our developed system shown in Table 3.

Table 3 shows the outcomes of each option available. These outcomes differ depending on several situations, for example during weekends a test system runs for a longer period due to lack of employees. With longer periods a test system has a higher coverage measurement because of more time allocated. For instance, option 1 and 2 has the same overall count of test cases, which is 372 test cases. Option 1 has an overall coverage of approximately 25% but option 2 has much higher results of coverage, approximately 96%. The reason is the amount of time allocated, option 1 is only for one night but option 2 has one week allocated. Option 3 has a different kind of calculation, where it calculates several test systems, therefore the duplicate count is a major factor to receive the right amount of coverage because some test cases have been either skipped or tested several times across all test systems in a branch. The calculated measurement of coverage is appropriate since there are no duplicated test cases.

We developed another feature, where it displays the test cases which has been overall skipped, that means they have never been tested. This calculation presents the overall coverage measurement, but in order to assure that the calculation is correct we calculated the overall skipped by generating the duplicate test cases, which has been both skipped and tested as shown in Formula 2 and 3. This was done as an error handling just to ensure that the total sum of both overall coverage and overall skipped is equal.
to 100%. If the sum is not equal to 100% then there is a problem in the calculation. The coverage calculation system displays the needed results for Westermo company to easily make the decision of their next steps. As a final result, we have successfully answered our research question.

\[ S = \text{Skipped Count} - \text{Duplicate Count} \]

Formula 2: S stands for skipped test cases, which have never been tested.

\[ \text{Overall skipped} = \frac{S}{\text{All count}} \times 100 \]

Formula 3: Overall skipped calculation, where S variable is the number of skipped test cases, which is calculated by subtracting skipped count with duplicate count as shown in Formula 2.

<table>
<thead>
<tr>
<th>Coverage Calculation</th>
<th>Option 1</th>
<th>Option 2</th>
<th>Option 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Count</td>
<td>372</td>
<td>372</td>
<td>418</td>
</tr>
<tr>
<td>Tested</td>
<td>92</td>
<td>357</td>
<td>409</td>
</tr>
<tr>
<td>Skipped</td>
<td>298</td>
<td>358</td>
<td>413</td>
</tr>
<tr>
<td>Duplicate</td>
<td>18</td>
<td>343</td>
<td>404</td>
</tr>
<tr>
<td>Overall Coverage</td>
<td>( \frac{298}{372} \times 100 = 24.7% )</td>
<td>( \frac{357}{372} \times 100 = 95.9% )</td>
<td>( \frac{413}{418} \times 100 = 97.8% )</td>
</tr>
<tr>
<td>Overall Skipped</td>
<td>( \frac{298 - 18}{372} = 75.3% )</td>
<td>( \frac{358 - 343}{372} = 4.03% )</td>
<td>( \frac{413 - 404}{418} = 2.15% )</td>
</tr>
</tbody>
</table>

Table 3: Outcome of the coverage calculation system

We came across another option before we used our main one. The option 4 is based on looping through each test system in order to receive information about each test case. The test case has either been tested or skipped. The measurement of coverage is received by dividing the executed test cases by the possible combinations of all test cases in all test systems as shown in Table 4. The benefit with this solution, according to the companies preferences, might be the possibility of knowing which test cases have been executed in each test system. If the company desires such an outcome, the benefit will improve the nightly time allocation due to not allocating time in unused test systems. The consequence with this solution is the overall coverage which includes duplicate test cases in different test systems. At this moment, our main solution does not provide such information because it is out of scope for this thesis, therefore we managed to deliver another more efficient solution, thus we could also have both solutions but this was left for future work.
<table>
<thead>
<tr>
<th>Coverage measurement</th>
<th>System 1</th>
<th>System 2</th>
<th>System 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test case 1</td>
<td>Tested</td>
<td>Tested</td>
<td>Skipped</td>
</tr>
<tr>
<td>Test case 2</td>
<td>Skipped</td>
<td>Skipped</td>
<td>Tested</td>
</tr>
<tr>
<td>Test case 3</td>
<td>Tested</td>
<td>Skipped</td>
<td>Skipped</td>
</tr>
<tr>
<td>Each System Coverage</td>
<td>$\frac{2}{3} \times 100 = 66%$</td>
<td>$\frac{1}{3} \times 100 = 33%$</td>
<td>$\frac{1}{3} \times 100 = 33%$</td>
</tr>
<tr>
<td>Overall coverage:</td>
<td>$\frac{3}{3} \times 100 = 44%$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Option 4, alternative solution of coverage measurement.
8. Implementation Process

In this chapter, detailed explanations of how we developed the coverage calculator system will be presented in different sections, primarily to explain our work.

8.1 Database Analysis

At this phase, we began to analyze the database to receive more information about potential solutions. We used SQLite database browser to research the tables and columns to experiment on different SQL commands. The SQL commands displayed information which helped us to understand the outcomes and some were used as solutions. The solutions which were obtained by the database were unfortunately not enough, the company desired an interface system which employees can interact with easily. Therefore we decided to locally upload the database to phpMyAdmin which is a free software tool used to interact with databases. By using this tool it increased our possibilities of potential solutions, the interaction with several languages helped us to deliver an interactive interface system.

We came across some issues when locally uploading the database to phpMyAdmin. One of the problems was to change the database file format to .sql. Another problem occurred during the upload was some syntax error, where phpMyAdmin did not understand some of the code in the actual database. The syntax Begin Transaction and quotations marks were not recognized. There was one more problem which occurred due to the size of the database. The settings of phpMyAdmin had to be changed to satisfy the required file size. The problems were easily solved but hard to detect, thereafter the database was successfully uploaded and ready to use.

8.2 Initial Prototype

Once we understood the problem fully, we decided to design an interface which will be interactive. The goal here is to deliver a user-friendly design which is a benefit if users will interact with it frequently. The prototype consisted of several select option and buttons which generated an outcome depending on the button pressed. The initial prototype is shown in Figure 5.

Figure 5: Prototype.

2 https://www.phpmyadmin.net/
8.3 Front-end Implementation

This project is being worked on by two researchers, therefore we split our work to make it as efficient as possible. The implementation was divided in both front-end and back-end, it was decided by our experience but we invested more time in back-end due to more complexity. In the front-end phase, there were several programming languages used to achieve a proper and user-friendly design. For instance, HTML and CSS were used to reach a well-structured design and Javascript was primarily used for the interaction between the front-end and back-end.\(^3\)

By following the prototype shown in Figure 5, the design process was possible to achieve. The select options shown in Figure 6, consist of a start date, end date, system names and a branch name. These select options enable the possibility to display a coverage measurement from a set of servers and branch name in a specific range. The measurement of coverage is possible to receive once every select option is chosen. The coverage result is thereafter generated by the show coverage button and the system could also generate the id of test cases, which has been either skipped, tested or duplicated by the other button shown in Figure 6. Further detailed explanation on how these buttons and select options are generating the outcomes is explained in section 7.5.

![Figure 6: select options in the initial interface design determines the results of systems and branches the user choose in a specific range to display. The outcomes are thereafter calculated and generated as desired by the blue buttons.](image)

8.4 Back-end implementation

To generate the desired outcomes we had to implement the functional backend requirement. Since our main goal is to reach a measurement of coverage, we had to decide what kind of algorithm is able to reach such a result. The algorithm was determined by our research and also the analysis of Westermo’s database. Considering the importance of back-end functionality, we had to connect the database to the back-end in order to reach the desired goal of this project. The connection process was done by PHP, which is a programming language that can interact with databases.\(^4\) The buttons and select option explained in section 7.4 which generates different outcomes was possible due to PHP and some SQL

3 \[https://www.edureka.co/blog/what-is-javascript/\]
commands because the results are initially generated from the database in use. SQL is used in the programming and management of database systems [15]. With the help of both front-end and back-end we managed to develop a coverage calculation system.

8.5 Coverage Calculation System

The coverage calculation system was reachable due to our research and Westermo’s database. We managed to develop the desired system based on the general coverage formula shown in Formula 1. Therefore, we had to first generate the total of tests and skipped cases. The issue of receiving such an outcome was difficult because the results showed thousands of duplicate test cases. These test cases were either skipped or tested, therefore it was a challenge to determine what has been actually tested and skipped. After further research, we managed to test Westermo’s database with different SQL commands to achieve the total of tested and skipped cases. Some of these SQL keywords which were used in order to receive a proper result were, for instance, DISTINCT, which generates non-duplicate test cases. JOIN keyword was used to combine more than one table according to a specific condition. The result of these SQL queries delivered a number of total tested and skipped cases which were used in the coverage calculation phase of this project. The mathematical calculations were based on Formula 1 and 2.
9. Discussion

The coverage calculation system generated a lot of different outcomes depending on the number of nights and systems. More nights result in a higher measurement of coverage but lesser nights end up in much lower coverage due to lack of time. The higher quantity of systems the more test cases the system displays, thus the test cases might run several times in both the actual and other test systems, which result in duplicates. Our thoughts about these results are that the test systems should have a better performance quality in terms of identifying the already tested test cases to avoid putting the effort of testing them again. By reaching this avoidance, the results of the coverage measurement will be higher due to fewer duplicates and more tested test cases. The process of the work could have been less effortless if there were more related works which could have made our solutions easier. But the research was limited and we could only find the general coverage formula which we based the solution on. The solution was purely done by personal effort.

Our aim in this thesis is to only calculate and display the coverage measurement of chosen data. We do not consider the decision making of Westermo company which is based on the result of our calculation system. However, we make it easier and possible for them to make the right and appropriate decisions. Other than that, our developed system reached the desired goal of this thesis, for example displaying the needed data of coverage in combining the results of both several systems and many nights. The outcome of our developed system calculates test execution coverage in network embedded systems, which answers the research question in discussion. Similar testing companies which encounter the same challenges could use our work as a base to calculate a convenient coverage result. Companies with manual testing could also benefit from our work, where they could develop automated tests and still keep the coverage of tested test cases known.

9.1 Future work

Westermo company preferred a solution with a specific programming language called GO. Unfortunately, our coverage calculation system was solved by other programming languages than the one they desired due to lack of time. We did not have time to master another language, therefore we worked with languages which we were comfortable with. Our coverage calculator system can add the sum of several test systems, the limited amount of test systems is up to six different systems. If Westermo prefers more they could develop our current system to make it compatible for more than six testing systems based on our solution. Overall the system does benefit the company since we reached the goal they asked for, however, there are some questions which can be investigated for future research which could also benefit the company.

- How to avoid testing of duplicated test cases?
- How to identify the essence of errors and what are they originating from?
- What is the difference in the measurement of coverage between manual and automatic testing?
10. Conclusion

Measuring test coverage in embedded systems when using many test systems and code branches is the main goal of this thesis, we have researched and worked effortly to achieve the goal. We want to make our thesis worthy with solutions for companies which encounters such challenges. The challenge of not knowing what the actual measurement of coverage is. Measurement of coverage is the percentage of total executed test cases which provides the possibility of easier decision-making of tested software. Regarding this thesis, these decisions are how to allocate nightly time more wisely for each test system. The allocated time is determined by the priority of the test systems, which could be identified by the calculation of coverage. The challenge Westermo company faced was the knowledge of coverage, which is not knowing the actual sum of tested test cases. This issue causes decision-making challenging, because of the lack of knowledge in the measurement of coverage. There are many benefits in receiving accurate coverage measurement, such as knowing the quality level of tested software. Because it helps the company to determine if software should be released or retested. We reached the goal and we delivered a calculation coverage system which enables the possibilities of better decision-making.
11. References


