Test case selection based on code changes and risk of regression

ALEXANDER OLOFSSON & CHRISTOFFER LARSSON

BACHELOR´S THESIS

DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Test case selection based on code changes and risk of regression

Alexander Olofsson
`al3586ol-s@student.lu.se`

Christoffer Larsson
`ch0815la-s@student.lu.se`

Department of Electrical and Information Technology
Lund University

Supervisors:
Christin Lindholm, `christin.lindholm@cs.lth.se`
Alf Stenbrunn, `alf.stenbrunn@axis.com`
Lars Nylander, `lars.nylander@axis.com`

Examiner:
Christian Nyberg, `christian.nyberg@eit.lth.se`

Bachelor's thesis work carried out at Axis Communications AB.

June 20, 2022

© 2022
Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Test case selection is an important part of the quality assurance process in a software project. Since time and resources are limited, test cases have to be prioritized in order to maintain efficiency and meet deadlines. This thesis researches the possibility of improving the development process at Axis Communications by partly automating the test case selections. In this thesis, an algorithm called the 'Test Selection Algorithm' is developed. The algorithm contains two methods for selecting test cases based on recent code changes and their risk of introduced errors or regression. The first method draws a connection between test cases that have previously found errors and reported them and the commits that solves the errors. The second method makes use of text analysis and keyword extraction and matches test cases to commits by analysing the commit messages. The resulting algorithm combines the two methods and produces suggestions on which test cases that should be included in upcoming test runs. A graphical interface containing the suggested test cases was created for the user of the algorithm to easily look at the results from the algorithm.

The result of this thesis is an algorithm that help partly automate the quality assurance process by suggesting test cases that the test leader should include.

Keywords: *Quality Assurance, Test Case Selection, Keyword Extraction, Text-analysis, Application Programming Interface.*

Sammanfattning

Urval av testfall är en viktig del av processen för kvalitetssäkring i ett projekt. Eftersom tid och resurser är begränsade behöver testfallen prioriteras för att upprätthålla effektivitet och för att möta deadlines. Detta examensarbete utforskar möjligheten att förbättra utvecklingsprocessen på Axis Communications genom att delvis automatisera utväljandet av testfall. I detta examensarbete har en algoritm som kallas 'Test Selection Algorithm' utvecklats. Algoritmen innehåller två metoder för att välja ut testfall baserat på nyligen gjorda kodändringar och deras risk för nya fel eller regression. Den första metoden drar en direkt koppling mellan testfall som tidigare hittat fel och rapporterat dessa, till kodändringar som löser felen. Den andra metoden använder textanalys och nyckelord för att matcha testfall till kodändringar genom att analysera meddelandena i kodändringarna. Den resulterande algoritmen kombinerar de två metoderna och producerar förslag på vilka testfall som borde inkluderas i kommande testrundor. Ett grafiskt användargränssnitt som innehåller de föreslagna testfallen skapades för att användaren av algoritmen enkelt skall kunna få en överblick över resultaten från algoritmen.

Resultatet av detta examensarbete är en algoritm som delvis automatiserar processen för kvalitetssäkring genom att föreslå testfall som testledare borde inkludera i testrundor.

Nyckelord: *Kvalitetssäkring, Testfallsutväljning, Nyckelords-uthämtning, textanalys, API.*

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

When changes in code are made there is always a risk of error and regression. Manual system testing is done before a new launch but since there is a limited amount of time the test cases must be prioritized. The Quality Assurance team, furthermore described as QA team, at Axis Communications is using an algorithm that analyzes code changes and evaluates the risk of errors in the change. This process is done every time Axis plans on releasing a new version of the software. The algorithm produces a list and a graphical overview of all commits* and ranks them based on risk for errors. This algorithm will further be called the "risk prediction algorithm". Commits are ranked based on risk of new errors in the code. The list of commits is then sent to the test leader who manually selects the test cases that need to be run based on the risky commits. Test cases are already defined, and the test leader selects the test cases that best match the commits with highest risk. Machine learning is used in the risk prediction algorithm to train and improve the predictions it produces.

A lot of time and resources are spent when a test leader has to manually analyze commits and select test cases based on the resulting list from the risk prediction algorithm. Therefor the team at Axis Communications want to develop a new algorithm that automatically selects the test cases that match the high risk commits. The new algorithm shall analyze commits, commit messages** and the resulting list produced by the risk prediction algorithm. The result of the new algorithm will help save time in the testing process by automatically selecting test cases that shall be completed before every new launch of software. The new algorithm will further be called the "test selection algorithm".

Commits and tests can be categorized to help the development of the test selection algorithm. Examples of categories could be size of changes or which area of code the changes are made.

Axis Communications AB is a Swedish company based in Lund that was founded in 1984. Axis Communications is the industry leader in network-based video surveillance since the launch of the worlds first network camera in 1996. Today, Axis Communications offers a wide variety of solutions of network cameras, network audio and access control for the physical security and video surveillance industry. Rather than targeting individual users, Axis main target customers are larger companies and institutions across varies segments, such as retail, transportation, government, healthcare and education etc.

*Commit - The latest change made in the code.

**Commit message - A description message from the developer who made the change.

1.2 Purpose

The purpose of this thesis is to research if the development process at Axis Communications can be improved by partly automating the testing process which will then save time, money, and resources.

1.3 Objectives

The thesis will research if the selection of test cases can be automated with the help of an algorithm. A prototype of the "test selection algorithm" shall be developed and evaluated. The expected result is that the test leaders will no longer have to manually select test cases.

1.4 Research questions

In this thesis the following questions shall be answered.

1. How are the test leaders at Axis currently selecting the test cases?
2. How does the risk prediction algorithm work?
3. How are code changes going to be analyzed and matched to test cases in the test selection algorithm?
4. Will text analysis be used in the test selection algorithm?
5. Can commits and tests be categorized to help the development of the test selection algorithm?
6. Can machine learning be used in the test selection algorithm?
7. How will the prototype of the test selection algorithm be evaluated?

1.5 Thesis motivation

This thesis was chosen because it was an interesting problem and the task description matched our education. We have also been in contact with the company earlier at different student work related events.

Axis Communications wants this work to be done since the result will hopefully improve the development process by saving time, money, and resources.

The thesis will help improve society since it can help the development of new tech and could also be applied to other areas of development. The test choosing algorithm which will be developed in this thesis shall be made for Axis Access Control products which are used to make places safer.

1.6 Boundaries

The test selection algorithm prototype shall be developed for both Windows and Linux environments but not for Mac.

Chapter 2

Technical background

2.1 Previous work

Test case selection is a method which aims to save time by only running the necessary test cases ahead of each release. By selecting the right test cases redundant test data can be avoided. The process of selecting which test cases to run has always been a time consuming process and has been automated in many different workplaces before. The difference between this thesis and work done before is that this is a continuation of a previous thesis written at Axis Communications called "Unsupervised predictions of software faults using change metrics" by Oskar Holmqvist and Elias Tedenvall which in this thesis is mentioned as the risk prediction algorithm. [1] The risk prediction algorithm uses machine learning to analyze which changes in code are high risk and which changes are safe. But selecting the test cases based on the results are still made manually by the test leaders.

To help the development of the "test selection algorithm" which will automatically select relevant test cases based on recent code changes, a literature review was made to search for different solutions and approaches to the problem. The following works are relevant to discover and analyze different ideas that could be implemented.

Filip Normann, a former student at Uppsala University, did a similar bachelor thesis with the same problem definition. In this thesis, titled "Test Case Selection Based on Code Changes"[2], Normann produced an algorithm for selecting test cases by leveraging code dependencies.

Beszédes et al.(2012)[3] used a code-coverage based technique for test case selection and prioritization with the goal of reducing the number of necessary tests. Alves et al.(2013)[4] had the same goal but chose to go for a refactoring based approach which mainly detects refactoring errors but also uses prioritization. The refactoring based approach was chosen since according to their case study this is the most common issue when regression testing.

Wang et al.(2013)[5] used a feature model for automatically selecting test cases for new products. Where test engineers had to select the relevant features and the existing test cases would be selected based on the features selected. This is mainly used to save time when testing new products.

In a systematic literature review Rongqi et al.(2021)[6] analyzed different machine learning models for selecting and prioritizing test cases which could be relevant if machine learning is used. Another approach that could be used to match testcases and commits is by analysing the information and extracting different keywords which is a technique shown in "Automatic Text Summarization and Keyword Extraction using Natural Language Processing"[7] by Payak et al.(2020). Where they have developed a tool to gather information from different sources and automatically sort the relevant information. This technique could be used to analyze and help match test cases in the "test selection algorithm".

The works referenced in this chapter will lay the ground work for developing the "test selection algorithm" and later when evaluating the solution.

2.2 Tools

This section describes the tools used for developing the "test selection algorithm" and to what purpose they were used.

2.2.1 Risk prediction algorithm

The risk prediction algorithm is used to get a list of all commits made in a chosen time period. The commits that are considered fault-prone or risky are used to determine which test cases that should be performed. [1]

2.2.2 TestTracker

TestTracker* is a platform developed by Axis Communications to manage and visualize test-runs, test cases and the results from the test-runs. In this thesis TestTracker is used to fetch test cases and test-run results. In the "test selection algorithm" the test cases and test-run results are used to create a connection to recent commits.

*TestTracker is a made up name, because of confidentiality, the real name will not be presented in this thesis.

2.2.3 Jira

Jira is a project and issue tracking tool that Axis uses to document detected issues and tasks within the software. The issues and tasks are posted as a "ticket" and is the basis for the code changes. In this thesis, Jira is used for selecting test cases that may catch Tickets reported to Jira. The tickets in Jira are categorized in to one of the following; Story - A requirement written from the perspective of an end user. Epic - A large task that can be broken down into several smaller tasks. Task - Work that shall be done. Issue - a bug or problem that needs to be fixed. The

'Test Selection Algorithm' makes use of all the different categories when matching test cases to commits.[8]

2.2.4 Gitlab

Gitlab is a git repository management platform. Gitlabs built-in API is used to fetch commits and relevant data which is used in the "test selection algorithm". Furthermore, Gitlab is used to track the development of the "test selection algorithm" itself.[9]

2.2.5 Gerrit

Gerrit is a web-based code collaboration tool used for code reviews and git management. Like Gitlab, Gerrit has a built-in API which is used for fetching commits and relevant data.[10]

2.2.6 Postman

Postman is an API platform which is used for building and using different APIs. In this thesis Postman is used for testing and visualising the information gathered from every API needed for the "test selection algorithm".[11]

2.2.7 RAKE

RAKE stands for Rapid Automatic Keyword Extraction and is an algorithm for extracting keywords from individual documents and text. The algorithm removes certain words, known as stop-words (e.g. 'and' 'of' the' etc.), and produces keywords with a given score. The higher score means a higher occurrence and co-occurrence* of the words. [12]

RAKE is used by the 'Test Selection Algorithm' to produce keywords which are used to select relevant test cases.

*co-occurrence - The frequency of which a word occurs together with other keywords.

2.2.8 SequenceMatcher

SequenceMatcher is a library available in Python. SequenceMatcher computes similarities between two given string sequences. In the 'Test Selection Algorithm' SequenceMatcher is used to determine similarities between the commit message and the titles of the test cases. A higher similarity means a higher relevance between the test case and the commit. [13]

2.2.9 YAKE

YAKE, similarly to RAKE is an automatic keyword extraction tool that identifies the most relevant keywords from text and documents. After comparing the resulting keywords from YAKE and RAKE it was decided that RAKE would be used in the algorithm since the keywords extracted from RAKE were more relevant. [14]

2.2.10 DASH

DASH is a library in python and an open source framework for building data visualization interfaces. DASH is used by the 'Test Selection Algorithm' to visualize the suggested test cases in a user-friendly manner instead of having all the data in separate text- and csv-files. [15]

Chapter 3

Method

The work process was split into four different phases as shown in Figure 3.1. The initial phase was gathering information. This was done by conducting a literature study on previous work related to the topic. The next method used to gather information was by having interviews with employees at Axis, including an interview with a test leader. The final method of information gathering was by manually testing one of the products. The second phase was spent on developing, testing and evaluating a prototype to the first solution, the TestTracker-Jira connection. The third phase was spent on developing, testing and evaluating a prototype to the second solution, the text analysis and keyword extraction. Finally the fourth phase was spent on merging the two prototypes developed into a finished product, also testing and evaluating it. The testing sub-phases consisted of running the tools developed and making sure the results are reasonable and consistent. Evaluation was conducted after every development sub-phase. The different phases and sub-phases are described in further detail throughout this chapter.

3.1 Research and information gathering

This section describes the methods used to research and gather information.

3.1.1 Literature studies

In the beginning of the thesis, information had to be gathered in order to come up with a solution. By conducting literature studies of similar and previous works, an approach on how to solve the problems of the thesis could be created.

The tools used to search for relevant literature were LUBsearch and Google Scholar using search terms "test case selection", "test case selection based on code changes and risk of regression" and "automatic test case selection". Other relevant literature gathered was sent from the supervisors at Axis Communication.

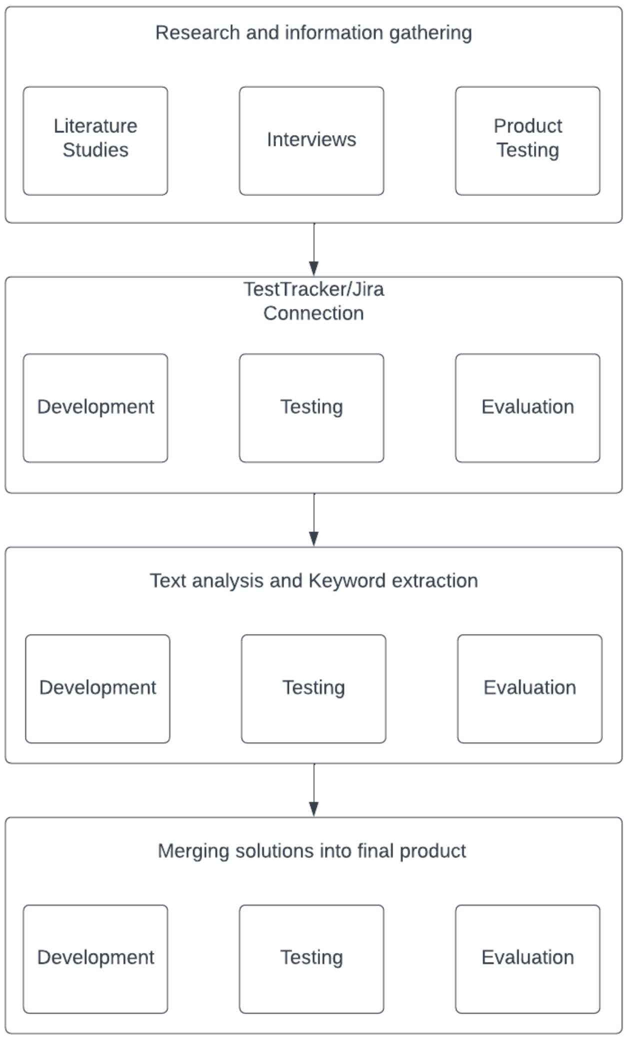

Research and information gathering
Literature Studies
Interviews
Product Testing
TestTracker/Jira Connection
Development
Testing
Evaluation
Text analysis and Keyword extraction
Development
Testing
Evaluation
Merging solutions into final product
Development
Testing
Evaluation

Figure 3.1: Visual representation of the phases.

3.1.2 Interviews

After completing the literature studies, information about the testing-process and how the QA team is currently working had to be gathered. Two interviews were planned and conducted. The first one was with the author of the "risk prediction algorithm" where the focus was on how the algorithm works and how it is used. The second interview was with one of the test leaders at Axis where focus was on how the test-leader uses the algorithm to select test-cases. The interviews provided valuable insight to the work process at Axis and helped lay the groundwork for developing the "test selection algorithm".

The interviews were conducted in a semi-constructed way with open conversations and a few prepared questions. [16] Both interviews were held over Microsoft Teams.

The most important questions asked from the interview with the author of the "risk prediction algorithm" were:

1. What information does the list of commits produced by the "risk prediction algorithm" contain?
2. Are the commits categorized based on which area of the code they concern?
3. How does the test leader use the "risk prediction algorithm"?
4. Is the data gathered from Jira used to evaluate the results?

The most important questions asked from the interview with one of the test leaders were:

1. How do you use the "risk prediction algorithm"?
2. What makes a test leader select a specific test case based on the results of the "risk prediction algorithm"?
3. Are pre-existing tags in test cases used?
4. How are test cases prioritized?
5. How would you like to view the results of our "test selection algorithm"?
6. How do you construct a test run before the next release?

The interviews gave satisfying answers to the questions and also gave more useful information such as a thorough presentation of the "risk prediction algorithm" in use.

3.1.3 Product testing

In order to better understand the product the 'Test Selection Algorithm' would be applied to, testing of the product was conducted. The testing was done by setting up a test environment for the product and manually running tests from the pool of test cases. The testing gave insight into the process which the algorithm would automate. The testing yielded a better understanding about the different test cases that the algorithm would select.

3.2 Development

After having completed the literature studies and interviews an idea for a solution was formed and a prototype was developed. The prototype solution contains two methods for selecting test-cases. The first method draws a direct connection between test-runs and commits that references the same issue-tickets. The other method is a text-analysis based approach that extracts keywords from the commit-messages. The reasoning for developing these two methods are described in section 3.2.1 and 3.2.2 respectively.

3.2.1 TestTracker-Jira connection

The interview, as described in 3.1.2, gave insight to the workflow of the QA team and which tools are used for the testing process. At a closer inspection of the tools, it was revealed that the test-run results from TestTracker (section 2.2.2) and many commits contained a link or reference to the same ticket in Jira (section 2.2.3). Therefore, a connection could be made between the test-case that reported the Ticket and the commit that solved the Ticket. This was the first method for selecting test-cases but since not all commits and test-runs contains a ticket reference, a second method had to be developed.

The TestTracker-Jira connection solution fetches all relevant information from TestTracker. This includes test-runs, test-run results, test-cases performed in the test-runs and comments from the test-cases. This is done using the built-in TestTracker API. All comments containing a link to a Jira-ticket is stored in a list. This list is then matched with all commits from GitLab and Gerrit containing Jira-Tickets. Since a commit can contain multiple Jira-Tickets, a commit can be matched to multiple test cases as well.

3.2.2 Text-analysis and keyword extraction

After conducting the literature studies of the previous works mentioned in chapter 2.1 it was decided to use text-analysis and keyword extraction for the solution. The decision was made partly because one of the research questions for this thesis is "Will text analysis be used in the test selection algorithm". The decision was also made because the first method for selecting test-cases using the connection between TestTracker and Jira, only covers a portion of the commits.

This method uses "RAKE" to generate keywords from commit messages which are later used by "SequenceMatcher" to compare keywords to test cases. "RAKE" also has a metric for how relevant every keyword is on a scale from 0-10. All keywords with a rating below 4.0 were removed since those keywords are often irrelevant. For every commit all test cases are ranked by a relevancy metric from "SequenceMatcher" where test case ranking for each commit can be viewed. Since a commit could cover multiple test cases a feature was added where the user can select how many of the top test cases for each commit that will be added to the final selection. This feature is important for evaluation where statistics can be measured for different selections.

3.2.3 Final development

After finishing the prototypes for both solutions and evaluating them the final part of development was started. This phase consisted of merging the two solutions together into a finished product where results and statistics are easily viewed in a browser. A configuration file was created where the user can manually choose which repositories the commits will be gathered from and the product that will be tested. The reason for this is for the 'Test Selection Algorithm' to work for every product at the company and not just for the one tested while developing.

3.3 Testing

For the TestTracker-Jira connection tests were made to verify all the issue tickets are getting added by the code and the correct matches were made. To test this all commits and test cases gathered were checked manually by the authors of this thesis to make sure the code collected every issue from the commit messages and that no potential matches were missed. To test the keyword extraction, the keywords generated were compared to the text they were extracted from to make sure they were relevant and that the prioritization looked reasonable. The test case selection from both methods is evaluated more thoroughly in the evaluation process.

3.4 Evaluation

Since the manual test selection today is based mostly on the experience and opinions of the test leaders it's difficult to determine if the "test selection algorithm" is making the correct selections. Therefor the evaluation has to be made in cooperation with test leaders. To evaluate the selections an overview of the commits along with all relevant information and selected test cases is made. The overview will be used to compare the "test selection algorithm's" selections to the selections of the test leaders. This method can also be used to gather statistics and measure accuracy.

Another way to evaluate the "test selection algorithm" is by comparing results to previous test runs. By selecting a test run and running the "test selection algorithm" for all risky commits* that happened in the two weeks prior to the test run a comparison between the test cases selected and the test cases in the test run can be made. This evaluation method was tried with selecting the top 10, top 5, top 3, and top 1 ranked test cases for each commit to see which one was the most accurate compared to the actual test runs chosen by the test leader. These evaluation runs were tried with both methods separately and together.

*Risky Commit - A code change that have a risk of introduced errors or a risk of regression.

3.5 Communication and work process

Since the majority of work is conducted in Axis office most of the communication is in person but Microsoft Teams is also used for different meetings, interviews, and to quickly send information back and forth. The work process is planned each week

3.6 Information evaluation

The referenced sources are trusted published scientific literature, technical tools and theses published at universities.

"Unsupervised predictions of software faults using change metrics"[1] and "Test Case Selection Based on Code Changes"[2] are both theses published at universities in the last year. Both are written by students and are meant for students with the same level of knowledge.

"Code coverage-based Regression test selection and prioritization in WebKit"[3], "A refactoring-based approach for test case selection and prioritization"[4], "Automated Test Case Selection Using Feature Model: An Industrial Case Study"[5] and "Automatic Text Summarization and Keyword Extraction using Natural Language Processing"[7] are all published at different international conferences. They are all written by specialists and experts on the topics and the works are meant for academics and professionals.

"Test Case Selection and Prioritization Using Machine Learning: A Systematic Literature Review"[6] is a literature review published at University of Ottawa. It's written by students and is meant for students with similar level of knowledge.

Chapter 4

Analysis

4.1 Evaluation results

The first test run compared(Table 4.1) consisted of 54 test cases where the "test selection algorithm" was tried in a few different ways. Starting with selecting only the highest ranked test case for each commit, then selecting the 3 highest ranked moving on to 5 and later 10. No TestTracker-Jira connections were found for this comparison.

Table 4.1: Table of first test run comparison.

First test run comparison			
Tests/commit	Test cases found	Correct test cases	Test cases in run
1	21	13	54
3	41	23	54
5	58	34	54
10	78	54	54

Since the most accurate method found too many test cases for a single run another method was tested. Adding together the relevancy score for every single test case and using the 54 highest since the run consisted of 54. This method was more successful having 40 out of the 54 selected test cases correct. Usually a single test run consist of 30-40 test cases so more test runs had to be compared.

The second test run comparison(Table 4.2) didn't find any TestTracker-Jira connections either. Using the method of adding all scores together and keeping the 43 highest ranked test cases for this run found 23 correct out of the 43 chosen.

The third test run comparison(Table 4.3) didn't find any TestTracker-Jira connections either. Using the method of adding all scores together and keeping the 40 highest ranked test cases for this run found 16 correct out of the 40 chosen.

Table 4.2: Table of the second test run comparison.

Second test run comparison			
Tests/commit	Test cases found	Correct test cases	Test cases in run
1	8	4	43
3	23	12	43
5	34	17	43
10	53	29	43

Table 4.3: Table of the third test run comparison.

Third test run comparison			
Tests/commit	Test cases found	Correct test cases	Test cases in run
1	12	6	40
3	32	16	40
5	45	19	40
10	64	27	40

4.2 Choices

In this section the different choices made for developing the 'Test Selection Algorithm' are discussed.

4.2.1 Two solutions

As stated in section 3.2.1, after a closer inspection of the different tools it was revealed that some comments from the test-run results in TestTracker and some commit messages from Gitlab or Gerrit referenced the same Jira-Issue. Logically, if a test case finds an issue and a commit claims to solve that issue, it would make sense to run the same test case again. This was the idea behind the TestTracker-Jira part of the 'Test Selection Algorithm'. However, while this method is accurate in selecting test cases, it demands that developers always reference Jira-Issues when committing.

Since the TestTracker-Jira solution only covers commits with a Jira-Issue reference, it was decided to implement a second method for selecting test cases. This method makes use of text-analysis and keyword extraction, as described in section 3.2.2.

4.2.2 Tools and languages

The 'Test Selection Algorithm' was developed in Python. [17] The decision to write the algorithm in Python was made because Python is one of the preferred languages at Axis and the 'Risk Prediction Algorithm' was written in Python as well. When developing the Keyword Extraction part of the 'Test Selection Algorithm' two different libraries were tested, RAKE (see section 2.2.7) and YAKE (section 2.2.9). It was decided to continue developing the algorithm with RAKE since the keywords yielded from the library were more relevant than those of YAKE. When the functionality of the algorithm had been implemented it was decided to visualize the result using DASH (see section 2.2.10). DASH was chosen because it has all the features needed to display the results and the 'Risk Prediction Algorithm' uses DASH as well.

4.2.3 Filters

When manually inspecting the commits gathered from the 'Risk Prediction Algorithm' it was discovered that some commits were only made for testing purposes and did not add or remove any functionality. A discussion with the test leader led to a decision being made that those type of commits could be filtered out and not be included in the 'Test Selection Algorithm'.

The resulting keywords from the RAKE library are given a score between 1 and 10. A higher score means that the keyword is of higher relevance. It was decided to remove every keyword that had a score lower than 4.0 since those keywords were irrelevant.

4.3 Problems and solutions

The first problem encountered while developing the TestTracker-Jira connection was that not every commit-message contained a specified Jira-Ticket. However since the keyword solution covers these commits as well the "test selection algorithm" can still find relevant test cases for these commits. This issue can also be solved by developers writing more detailed commit-messages and always including the specified Jira-Ticket the commit is solving.

Another issue encountered during development was commit-messages containing multiple Jira-Tickets. This was an oversight from the thesis workers part but was easily solved by changing the "test selection algorithm" to work with multiple Jira-Tickets for every commit. This also helped improve the TestTracker-Jira connection since the algorithm now found more matching Jira-Tickets.

While testing and evaluating the TestTracker-Jira part of the algorithm, there were not enough matches to be a proper solution. By looking closer at the Tickets in Jira, it was discovered that the Tickets had a field for 'Related Tickets'. The related Tickets are Tickets that have reported the same or similar bugs, problems or tasks. By taking the related Tickets into account in the algorithm, more matches could be made. However, the amount of matches were still not enough. To solve this, it would be required for developers to include a reference to Jira Tickets.

Chapter 5

Result

5.1 TestTracker-Jira connection

The TestTracker-Jira connection solution, as described in section 3.2.1, fetches all relevant information from Axis own platform TestTracker. This information includes test-runs, results from the test-runs and the different test-cases performed in the test-run. See table 5.1 for the information gathered from each test run. The results from the test-runs includes whether or not the test-cases passed or failed and a comment. The comments that contains a reference to a ticket in Jira are used to match test-cases to commits that solves the tickets. The information gathered from Gerrit and GitLab seen in table 5.2 are used to match the commits with the test cases. See figure 5.1 for a full representation of how the method works.

Table 5.1: Table showing an example of data gathered from a TestTracker test-run.

TestTracker Test-run Example		
Test Case	Status	Comment
Test 1	Passed	No comment
Test 2	Failed	Jira Issue A
Test 3	Failed	Jira Issue B
Test 4	Passed	No comment

If a connection between a test-case and commit can be made, the algorithm presents the title of the test-case together with in which test-run it was performed. The algorithm also presents the commit that solves the ticket and which ticket that was reported and solved. See table 5.3 for TestTracker/Jira results.

As evident in the evaluation results (see section 4.1) the TestTracker-Jira connection only works if developers references a Jira ticket in the commit message which is not the case for the most part. The functionality behind the method works and the suggested test cases from the solution are accurate. This is backed up by the

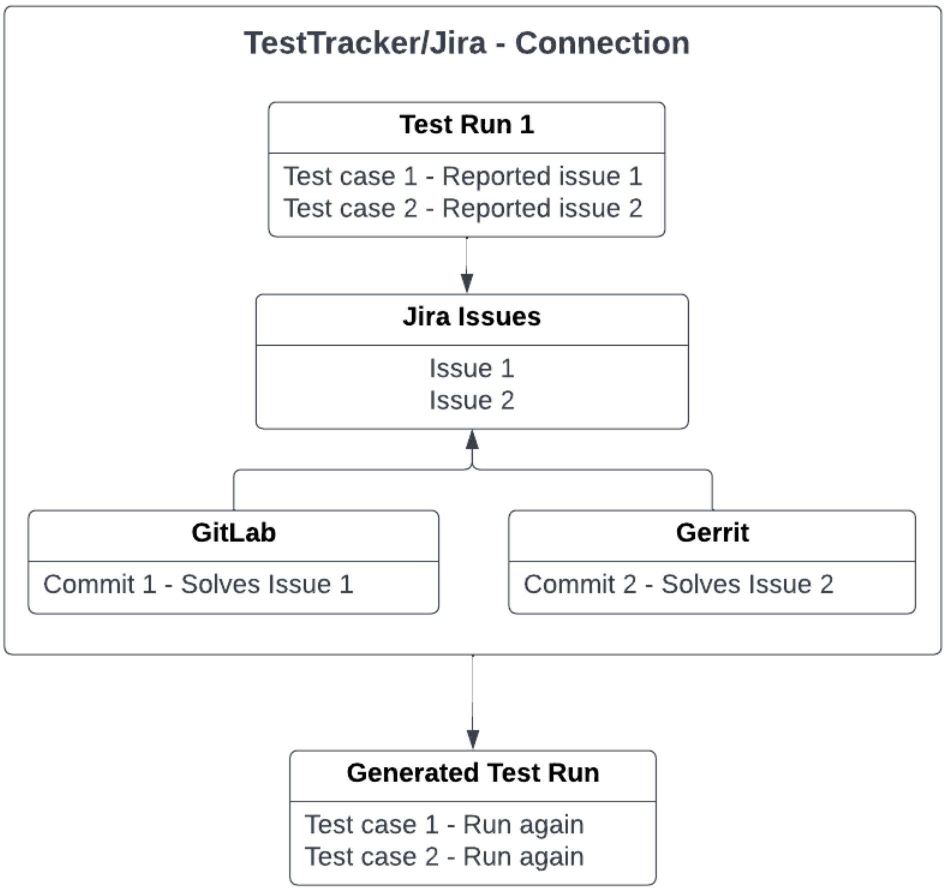

TestTracker/Jira - Connection
Test Run 1
Test case 1 - Reported issue 1
Test case 2 - Reported issue 2
Jira Issues
Issue 1
Issue 2
GitLab
Commit 1 - Solves Issue 1
Gerrit
Commit 2 - Solves Issue 2
Generated Test Run
Test case 1 - Run again
Test case 2 - Run again

Figure 5.1: Visual presentation of the TestTracker/Jira connection.

Table 5.2: Table showing an example of data gathered from Gerrit and GitLab.

Gerrit and GitLab Example		
Commit-ID	Jira-Issue	Message
Commit 1	Jira A, Jira B	Message
Commit 2	Jira C	Message
Commit 3	No Jira	Message
Commit 4	No Jira	Message

Table 5.3: Table showing an example of the resulting list from the TestTracker/Jira Connection.

TestTracker/Jira Connection Result		
Commit-ID	Jira-Issue	Test Case
Commit 1	Jira A, Jira B	Test 2, Test 3
Commit 2	Jira C	No Connection
Commit 3	No Jira	No Connection
Commit 4	No Jira	No Connection

testing of the method (see section 3.3). In order for the TestTracker-Jira connection to work, it would require developers to reference Jira tickets in the commit messages and for the test team to report Jira tickets during test-runs.

5.2 Text-analysis and Keyword extraction

The solution using text analysis and keyword extraction (see section 3.2.2) fetches the commit messages for all commits used in the algorithm. The commits are gathered from the 'Risk Prediction Algorithm'. The commit messages are then run through RAKE (see section 2.2.7) and keywords from the message are extracted. The keywords are then used to match the commit to the test-cases via SequenceMatcher (see section 2.2.8). After running the keywords through SequenceMatcher, a value is given to every test-case. The higher the value, the more relevant the test-case is to the commit. See figure 5.2 for a representation of the method.

The 'Test Selection Algorithm' presents the commits paired together with the values given for each test-case as seen in table 5.4. It also presents the total ranking for all selected test cases as seen in table 5.5. The user of the algorithm can determine how many test-cases per commit that shall be presented via the configuration file.

Table 5.4: Table showing an example of the results for a specific commit.

Single Commit Results		
Ranking	Test Case	Value
1	Test 8	7.0
2	Test 5	5.5
3	Test 12	3.2
4	Test 23	1.8

As evident in the evaluation results (section 4.1), around 10 test cases per commit yields the most correct suggestions. However, it also suggests more test cases than what is usually included in a test-run.

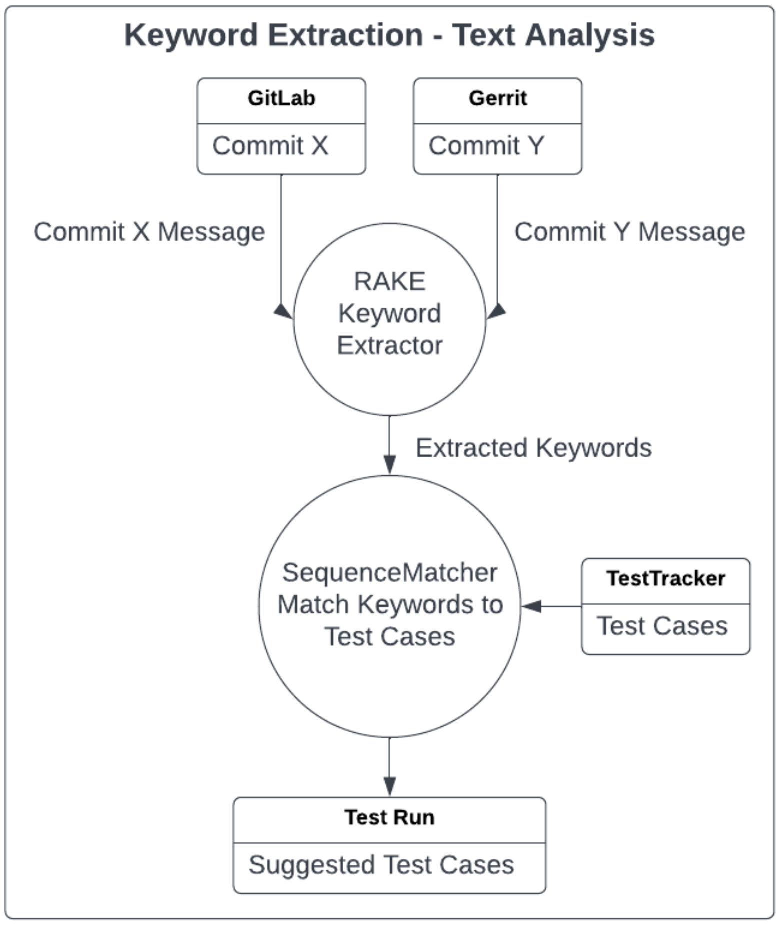


Figure 5.2: Visual representation of the keyword extraction and text analysis.

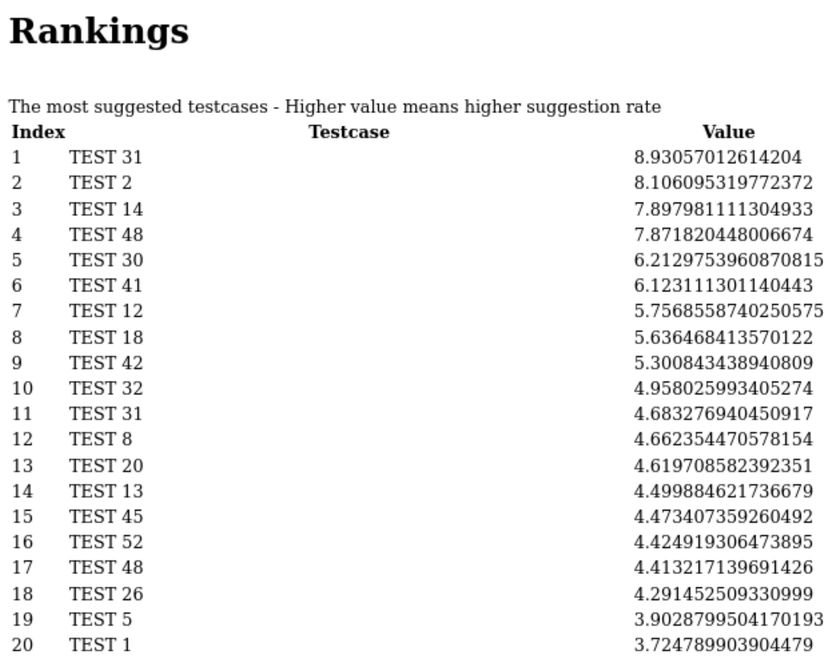
Table 5.5: Table showing an example of the total ranking for all test cases.

Total Ranking Results		
Ranking	Test Case	Value
1	Test 5	35.0
2	Test 2	25.5
3	Test 42	23.2
4	Test 19	21.8

5.3 DASH - Data visualisation

After the 'Test Selection Algorithm' has been run, a web browser is opened in which the results from the algorithm is presented. The user can view a ranking of the most suggested test cases (see figure 5.3) as well as view suggestions for each individual commit. When viewing suggestions for an individual commit, the commit-id and a link to the commit are displayed. A table consisting of the suggested test cases for the commit is also displayed, see figure 5.4.

If any TestTracker-Jira matches were made, these can be viewed as well (see figure 5.5). Figure 5.6 shows a visual representation of how the final tool works together with DASH to gather, match and display all results.

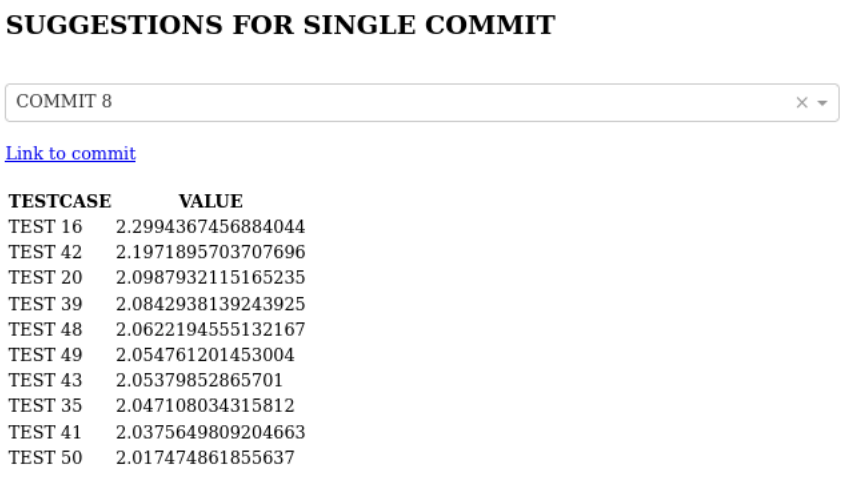


Rankings

The most suggested testcases - Higher value means higher suggestion rate

Index	Testcase	Value
1	TEST 31	8.93057012614204
2	TEST 2	8.106095319772372
3	TEST 14	7.897981111304933
4	TEST 48	7.871820448006674
5	TEST 30	6.2129753960870815
6	TEST 41	6.123111301140443
7	TEST 12	5.7568558740250575
8	TEST 18	5.636468413570122
9	TEST 42	5.300843438940809
10	TEST 32	4.958025993405274
11	TEST 31	4.683276940450917
12	TEST 8	4.662354470578154
13	TEST 20	4.619708582392351
14	TEST 13	4.499884621736679
15	TEST 45	4.473407359260492
16	TEST 52	4.424919306473895
17	TEST 48	4.413217139691426
18	TEST 26	4.291452509330999
19	TEST 5	3.9028799504170193
20	TEST 1	3.724789903904479

Figure 5.3: Page for viewing total ranking.



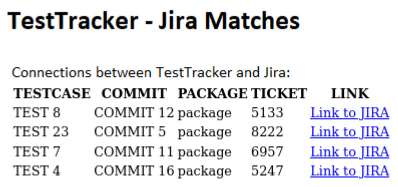
SUGGESTIONS FOR SINGLE COMMIT

COMMIT 8

Link to commit

TESTCASE	VALUE
TEST 16	2.2994367456884044
TEST 42	2.1971895703707696
TEST 20	2.0987932115165235
TEST 39	2.0842938139243925
TEST 48	2.0622194555132167
TEST 49	2.054761201453004
TEST 43	2.05379852865701
TEST 35	2.047108034315812
TEST 41	2.0375649809204663
TEST 50	2.017474861855637

Figure 5.4: Page for viewing a single commit.



TestTracker - Jira Matches

Connections between TestTracker and Jira:

TESTCASE	COMMIT	PACKAGE	TICKET	LINK
TEST 8	COMMIT 12	package	5133	Link to JIRA
TEST 23	COMMIT 5	package	8222	Link to JIRA
TEST 7	COMMIT 11	package	6957	Link to JIRA
TEST 4	COMMIT 16	package	5247	Link to JIRA

Figure 5.5: Page for viewing TestTracker - Jira matches.

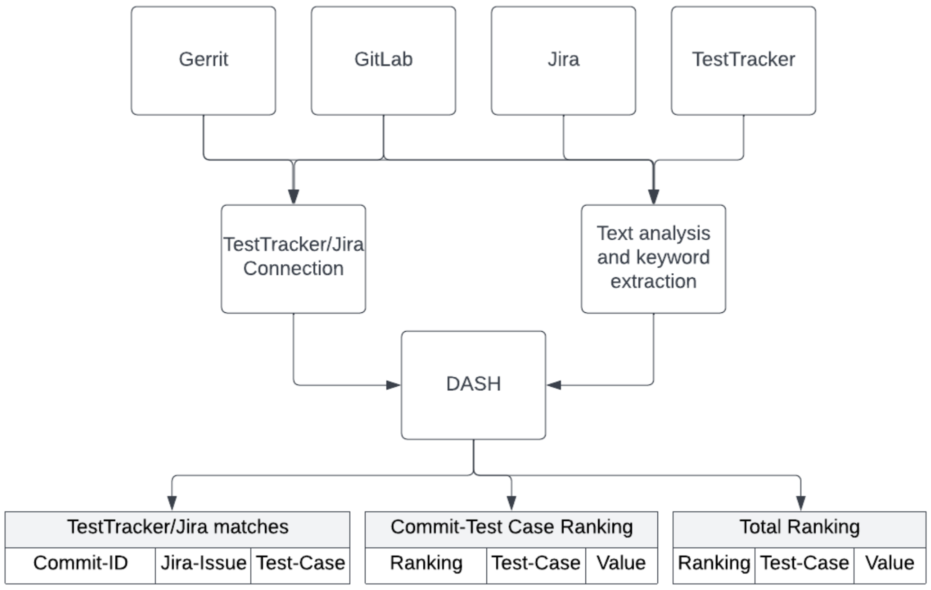


Figure 5.6: Visual representation of the final "test selection algorithm".

Chapter 6

Conclusion

6.1 Summary

The result of this thesis is an algorithm that automatically suggests test cases based on recent code changes and the risk of introduced errors or regression. The algorithm contains two methods for suggesting test cases. The first method, the TestTracker-Jira connection, draws a direct connection between the test-case that reported an issue or bug and the commit that solved it. This method yields the most accurate suggestions but demands that developers and testers always references Jira tickets.

The second method makes use of text analysis and keyword extraction and suggests test cases based on the keywords extracted from commit messages. This method yields less accurate suggestions compared to the first method, but it does not require developers to always include references to Jira tickets. In order for this method to work correctly and accurately, the commit messages have to be descriptive enough to extract accurate keywords.

The algorithm opens a graphical user interface in a web browser where the QA test leader can view rankings of the test cases as well as suggestions for individual commits.

Since the test cases that are run are mostly based on the experience and knowledge of the QA test leader, it is hard to determine how accurate the suggestions from the 'Test Selection Algorithm' are. The algorithm suggests test cases based on direct connections between test cases and commits or the commit messages but in the end it is the QA test leader that selects which test cases that shall be run. Therefor, the algorithm solely gives suggestions and the test leader can then choose which of the suggestions that should be included.

The purpose of this thesis (see section 1.2) was to research if the development process at Axis Communications can be improved by partly automating the testing process. The 'Test Selection Algorithm' fulfills this purpose by helping the QA test leader select test cases that shall be included in test-runs.

6.1.1 Research Questions

This section answers the research questions stated in section 1.4.

- How are the test leaders at Axis currently selecting the test cases?

  The test leaders at Axis are currently selecting test cases mostly based on experience and prior knowledge as well as 'gut feeling'.

- How does the risk prediction algorithm work?

  The 'Risk Prediction Algorithm' is a machine-learning algorithm that depicts whether or not commits are considered prone to introduce errors or have a risk of regression. The algorithm has a number of features to determine if a commit are risky, examples of these features are lines of code added or deleted and number of commits per file.

- How are code changes going to be analyzed and matched to test cases in the test selection algorithm?

  The 'Test Selection Algorithm' matches test cases to code changes via two methods. The first with a direct connection between test cases that reported an issue and commits that solved the issue. The second method uses text analysis and extracts keywords from the commit messages and matches them to test cases.

- Will text analysis be used in the test selection algorithm?

  Text analysis was used in developing the second method for matching test cases to commits.

- Can commits and tests be categorized to help the development of the test selection algorithm?

  Although categorization was not used in the 'Test Selection Algorithm', it is listed as a potential further development, see section 6.3.5.

- Can machine learning be used in the test selection algorithm?

  Machine-learning was not used in the 'Test Selection Algorithm' but is listed as a potential further development, see section 6.3.2.

- How will the prototype of the test selection algorithm be evaluated?

  The 'Test Selection Algorithm' was evaluated by comparing the resulting suggestions to prior test-runs. The algorithm was also evaluated by discussing the results with the QA test leader in order to determine if the suggestions were reasonable enough to use.

6.2 Ethical reflection

This section discusses ethical aspects that this thesis may apply to.

6.2.1 Confidential Information

The 'Test Selection Algorithm' is a new initiative for the New Business QA department at Axis Communications. Since the algorithm is applied to products not yet out on the market, it is of great importance that confidential information is not leaked. For this reason, it is important that any code snippets, screenshots or images used in this thesis, whether it is for the report, poster or presentation, do not contain information that is deemed confidential.

6.2.2 Societal benefit

Axis Communications offers network solutions in the domain of physical security. In order for the security to be effective, it has to be reliable. Quality Assurance helps ensure that the products that Axis offer are reliable and up to quality. The process of Quality Assurance includes running tests on the products. However, time and resources are limited and the test-cases needs to be prioritized. The 'Test Selection Algorithm' improves the Quality Assurance process by suggesting test-cases that shall be run and overall speeding up the process. This leads to an improved Quality Assurance which in turn leads to more reliable products. As stated, reliability is of great importance when offering solutions within physical security.

6.3 Future development opportunities

6.3.1 Area of use

While the 'Test Selection Algorithm' was developed for a specific product of Axis Communications, it is adaptable. The algorithm contains a configuration file where the user can determine which product or project the algorithm should apply to and from which source-code repositories the commits are gathered from. While this makes the algorithm adaptable, it is only available for the projects and products at Axis Communications since the platform for fetching the test cases is an in-house platform. The algorithm could possibly be developed further to be applicable to a more general project and select test cases from a variety of sources.

6.3.2 Machine Learning

One of the research questions in section 1.4 states: "Can machine learning be used in the Test Selection Algorithm?". While machine learning was not used in the finished algorithm, it is worth bringing up as a potential further development and perhaps even an improvement.

6.3.3 Heat map

The QA team at Axis Communications is using a heat map over which area the recent code changes may affect. The 'Test Selection Algorithm' could possibly incorporate this heat map to further suggest accurate test cases.

6.3.4 Code packages

Filip Normanns thesis about 'Test Case Selection based on Code Changes' describes a similar algorithm to the 'Test Selection Algorithm'. Normanns algorithm levaraged code dependencies for selecting test cases. [2]

Although the 'Test Selection Algorithm' has a different approach, including the packages in which the commit is made and making use of code dependencies, could potentially be a further development opportunity.

6.3.5 Categorization

Categorization of the commits could potentially help with selecting relevant test cases. An example of categorization could be that commits are categorized by the area of code the commit may affect. By doing this, it could be combined with the use of the heatmap (section 6.3.3).

Chapter 7

Terminology

This chapter lists and explains terms and shortenings used in this thesis.

7.1 Terms and shortenings

- Commit - The latest change of source code in a repository.
- Commit-id - A hash-encrypted identification number for a commit.
- Commit message - A descriptive message from the developer that made the code change.
- Heat map - Heat map is a data visualization technique where the data is represented as colors. The color variation can depict intensity or affected areas.
- Regression - Regression or software regression is a software bug where new changes makes previously working code stop working and you have to revert back.
- Repository - In Software version control systems, a repository is a storage location for software packages and source code.
- Quality Assurance (QA) - Quality Assurance is a way of preventing mistakes and defects when delivering products to customers.

References

[1] Oskar Holmqvist, and Elias Tedenvall, *Unsupervised predictions of software faults using change metrics*, `https://www.lunduniversity.lu.se/lup/publication/9066702`

[2] Filip Normann, *Test Case Selection Based on Code Changes*, `https://uu.diva-portal.org/smash/get/diva2:1371200/FULLTEXT01.pdf`

[3] Árpád Beszédes, Tamás Gergely, Lajos Schrettner, Judit Jász, László Langó, Tibor Gyimóthy, *Code coverage-based Regression test selection and prioritization in WebKit*, `https://ieeexplore-ieee-org.ludwig.lub.lu.se/stamp/stamp.jsp?tp=&arnumber=6405252&isnumber=6404866`

[4] Everton L.G. Alves, Patricia D.L. Machado, Tiago Massoni, Samuel T.C. Santos, *A refactoring-based approach for test case selection and prioritization*, `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6595798&isnumber=6595779`

[5] Shuai Wang, Arnaud Gotlieb, Shaukat Ali, Marius Liaaen, *Automated Test Case Selection Using Feature Model: An Industrial Case Study*, `https://doi.org/10.1007/978-3-642-41533-3_15`

[6] Rongqi Pan, Mojtaba Bagherzadeh, Taher A. Ghaleb, Lionel Briand, *Test Case Selection and Prioritization Using Machine Learning: A Systematic Literature Review*, `https://arxiv.org/abs/2106.13891`

[7] Avinash Payak, Saurabh Rai, Kanishka Shrivastava, Reshma Gulwani, *Automatic Text Summarization and Keyword Extraction using Natural Language Processing* `https://ieeexplore-ieee-org.ludwig.lub.lu.se/stamp/stamp.jsp?tp=&arnumber=9155852&isnumber=9155547`

[8] Atlassian, *Jira Software development tool* `https://www.atlassian.com/software/jira/features`

[9] Gitlab, *Gitlab DevOps Platform* `https://about.gitlab.com/`

[10] Gerrit, *Gerrit Code Review* `https://www.gerritcodereview.com/`

[11] Postman, *Postman API Platform* `https://www.postman.com/product/what-is-postman/`

[12] RAKE, *Rapid Automatic Keyword Extraction* `https://pypi.org/project/rake-nltk/`

[13] Python SequenceMatcher, *SequenceMatcher* `https://docs.python.org/3/library/difflib.html`

[14] Ricardo Campos, Vítor Mangaravite, Arian Pasquali, Alípio M. Jorge, Célia Nunes, Adam Jatowt, *YAKE! - Yet Another Keyword Extractor* `https://pypi.org/project/yake/`

[15] Plotly, *DASH* `https://plotly.com/dash/`

[16] Annika Lantz, *intervjumetodik*, 2013

[17] Python Software Foundation, *Python*, `https://www.python.org/about/`

Printed by Tryckeriet i E-huset, Lund 2022

Series of Bachelor´s theses
Department of Electrical and Information Technology
LU/LTH-EIT 2022-879
http://www.eit.lth.se