Mittuniversitetet
MID SWEDEN UNIVERSITY

An examination of automated testing and Xray as a test management tool

Simon Bertlin

Type of document — Computer Engineering BA (C), Final Project
Main field of study: Computer Engineering
Credits: 15 hp
Semester, year: Spring, 2020
Supervisor: Dr. Ulf Jennehag, ulf.jennehag@miun.se
Examiner: Jan-Erik Jonsson, jan-erik.jonsson@miun.se
Degree programme: Computer Engineering, 180 credits

Abstract

Automated testing is a fast-growing requirement for many IT companies. The idea of testing is to create a better product for both the company and the customer. The goal of this study is to examine different aspects of automated testing and Xray as a test management tool. The literature study considers information from several different scientific reports. It shows that the benefits of automated testing include increased productivity and reliable software but also pitfalls like a high initial cost and a maintenance cost. Research suggests that automated testing is complementary to manual testing. Manual testing is more suited for exploratory testing, while automated testing is better for regression testing. Using historical data manual tests can be placed into prioritised clusters. The coverage of each test within a cluster determines its priority, where a test with high coverage has a high priority. A near-optimal solution for prioritising automated tests is the combination of two well-known strategies, the additional coverage strategy and the total coverage strategy. Tests are prioritised based on how much of the code they uniquely cover. Code coverage is measured using statements, methods or complexity. Furthermore, this thesis demonstrates a proof of concept for how the unified algorithm can prioritise Xray tests. Xray is evaluated according to the ISO/IEC 25010:2011 standard together with a survey done on Xray practitioners. The evaluation for Xray shows that Xray provides the necessary tools and functions to manage a testing suite successfully. However, no official encryption exist for the Jira server, and Xray lacks integrated documentation.

Keywords: Xray, testing, automated testing, manual testing, Jira, REST, API, test prioritisation

Table of Contents

Terminology / Notation

REST	Representational State Transfer
API	Application Programming Interface
CI	Continuous integration
HTTP	Hypertext Transfer Protocol
HTML	HyperText Markup Language
XML	Extensible Markup Language
TCP	Test Case Prioritisation
APFD	Average Percentage of Faults Detected
JRJC	Jira REST Java Client
ADC	Additional Coverage Strategy
TCS	Total Coverage Strategy

1 Introduction

The demand from consumers on applications and software products are higher than ever. Famous IT companies, such as Facebook push code daily into production[1]. A core part of this fast development is automated testing. Automated testing allows code to be tested before being pushed into production. Testing ensures correct operation, and that new or modified code does not break any other code that is currently in production. Thus testing is a critical component of serious IT companies.

The idea of automated tests is to create more value for both the company and the customer. The customer gets a reliable product, and the company can more easily maintain their product since the automated tests notify the developers if changes to the codebase have broken some part of the software.

This project is done with the company Knowit in Sundsvall, Sweden. Knowit is in the process of incorporating automated testing and wants to know what current research says about automated testing and how the tool Xray can help to manage tests, both automated and manual.

1.1 Background and problem motivation

Tests give the customer some guarantee that the software is working correctly and is mostly bug-free. Tests can either be automated or manual. The manual test is where a team or an individual manually operate the software to make sure it's operating correctly. In contrast, automated tests run automatically to ensure correct operations.

There are several different types of tests a few examples are, unit tests, system integration tests, and acceptance tests. Having all these tests to keep track of creates a need for a tool to manage tests. The choice of instrument for this task is an important one since unmanaged testing can be costly. Furthermore, a too complicated tool might frustrate the developers, and a too simple tool might not live up to the requirements of the project.

Companies that perform testing on their products might have to adhere to a testing budget. This budget consist of all the tests that a company can afford to run. This budget creates a need for tests to be prioritised, such that the most critical tests have a high priority.

1.2 Overall aim

This project aims to examine testing both in research and in practice. The project examines the challenges with automated testing and what support testing have in current research as well as how tests can automatically be given a priority. Furthermore, the project investigates when manual testing is needed and when it can be automated. Finally, the project analyses if Xray is a suitable tool for test management.

1.3 Scope

This project examines current research on testing when you can move manual testing to automated testing and how to prioritise tests. There exist several strategies for prioritising tests. For test case prioritisation the focus is on coverage-based and risk-based strategies. Another focal point of the project is the technical challenges of automated testing, but a few administrative challenges relevant to test management are also considered. Furthermore, the project examines current research to explore the benefits and possible drawbacks of automated testing. As for tools, the project is limited to the Jira server platform and examine Xray integration with Jira server and how Xray can help to manage different types of tests, both manual and automated. The analysis of Xray is restricted to the Maven integration and the REST API services of Xray

1.4 Concrete and verifiable goals

This project looks to answer the following questions.

- What role do automated tests have in the current software development?
- Should a company abandon manual testing and only use automated tests?
- Does existing research support that a company can increase productivity and efficiency by moving to automated testing?
- What does the research say about test prioritisation?
- Can Xray issues be prioritised automatically?
- Is Xray a suitable tool for handling automated testing?

1.5 Outline

Chapter 2 introduces the theory needed to understand the result. This chapter contains more detailed information about the concepts and ideas that are presented in the result. Chapter 3 talks about the methodology used to achieve the result and complete this project. Chapter 4 presents the construction of the software that was later tested with Cucumber tests in Xray. Chapter 5 shows the result of the litterateur study and the evaluation of Xray. Chapter 6 analyses the result of the external examination of Xray and the result of the survey. Chapter 7 discusses the research study along with future work and the goals of the project.

2 Theory

This chapter presents the theory that is relevant for the understanding of the result and the conclusion.

2.1 Different types of tests

Manual and automated testing [2] are the two main categories for testing. Manual testing means that a real person performs some tests on the software. An example of a manual test is pressing a button and making sure the button functions as intended. There is usually some visual indication that the button operates successfully. Navigation might happen, a pop-up might appear, or text might be printed to the terminal. This way, the tester that issued the button press gets verification that the test was successful. Automated tests are performed by the machine running the tests. The machine follows a set of precise steps that instruct the machine how to complete the test. Tests are then split into more specialised forms. Below a few of the more common types of tests are presented.

2.1.1 Unit testing

Unit tests [2] run at a low level, and they are designed to test a small piece of code. A unit test usually asserts that function returns a specific value. A class that has several functions can then have several unit tests.

2.1.2 Integration testing

Integration tests [2], like unit tests, run at a low level. The purpose of integration testing is to see how well different components or modules of the software work together. For example, the interaction with a database through a REST API.

2.1.3 Exploratory testing

Exploratory testing [3] is a way to review a product from a user respective. The tester manually explores the software looking for bugs or defects. It gives the tester rapid feedback if something is missing or faulty.

2.1.4 Black-box testing

Black-box testing [4] ignores the internal mechanism of software and instead focuses on verifying that the output is correct. Black-box testing is

generally used to test the whole system instead of some specific part. Or how the system integrates with another system.

2.1.5 White-box testing

In contrast to black-box testing, white-box testing [4] takes into consideration the internal working of the software. White-box testing usually measures some coverage, such as the tests coverage of statements or branches.

2.1.6 Regression testing

Regression testing [5] means to test if modifications to the software or product have caused some unintended defects. Regression testing exists to ensure correct operation after modification.

2.2 Test Case Prioritisation

Test case prioritisation (TCP) [6] is a tool that can be used to reduce the cost of testing. This is done by only selecting specific tests to run instead of running every test. To rank TCP algorithms, a metric called Average Percentage of Faults Detected (APFD) is used. The value from APFD range from 0 to 100. A higher number means better fault detection.

2.3 Jira

Jira [7] is a software platform which supports the addition of plugins. A plugin can be connected to Jira and change the way Jira functions. The idea of Jira is to help manage teams and projects. Some common uses of Jira are planning, issue tracking, reports and work management, Jira itself it does not support tests. However, Jira defines a Representational State Transfer application programming interface (REST API) to allow applications created by independent developers to interact with the Jira platform.

REST [8] is an architecture style that sets up rules for how web services can safely communicate. API [9] is the endpoints that software can use to get data or post data. APIs allows for software specialised in other things such as testing to display and report the result of the tests to the Jira platform[10] .

2.4 HTTP Requests

One of the usage areas for the Hypertext Transfer Protocol (HTTP) [11] is to interact with a REST API. Some of the more relevant ones for this

project is this GET and POST request. When a GET request is executed, it requests a transfer of data from some destination. In contrast, when a POST request is executed, it requests that the destination processes its payload.

2.5 Xray

Xray [12] was created to manage tests and integrate with Jira. Xray supports both manual and automated testing. Xray defines several REST API endpoints which can be used to export tests created in Jira or import test results to Jira [13]. An example of this is shown in figure 1.

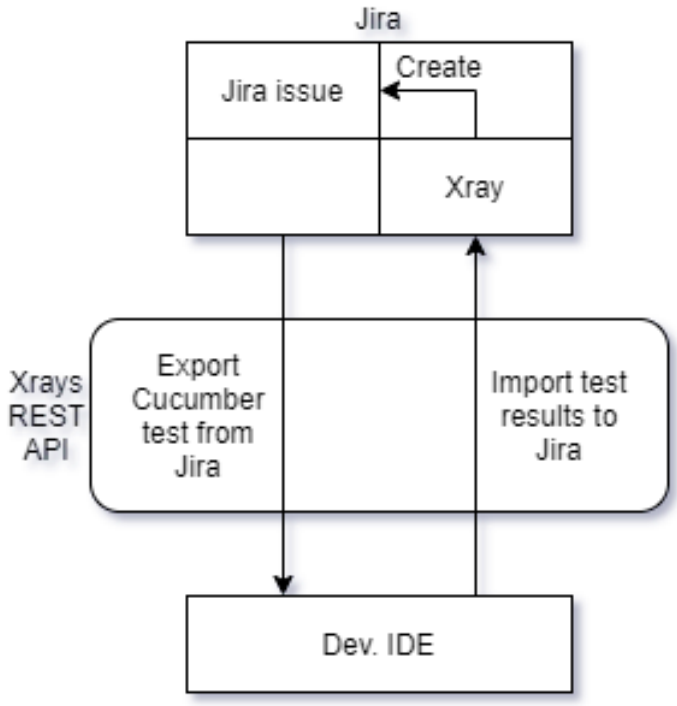

Figure 1: Overview of interaction between Jira and Xray.

2.5.1 How Xray interacts with Jira

Xray complements Jira by adding new Jira issue types [12]. Jira uses issues to represent things that need to be done. Such as a task that needs to be completed or a bug a that needs to be fixed [14]. A goal of Xray is to allow it's users to manage the entire test process in Jira. Xray adds five new issue types available in Jira [15]. These issues are meant to help the user follow the test life cycle shown in figure 2.

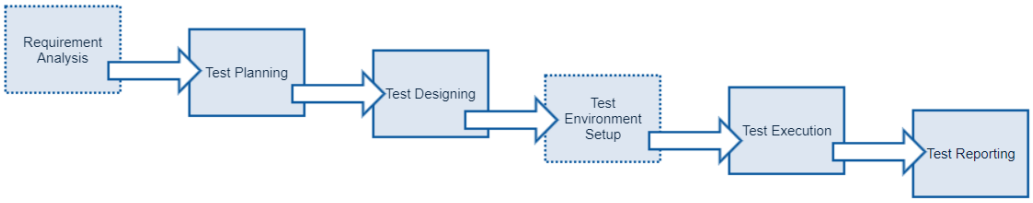

Figure 2: Test life cycle. [15]

When setting up a test suite, the following issues are defined as follows:

1. **Pre-condition**
   A pre-condition [16] is something that needs to be true before a test can be executed. For example, the user might need to be authenticated before using some service.

2. **Test issue**
   Xray defines a test issue [17] that support both automated and manual tests. For automated tests, there exist two types Cucumber and generic. Cucumber is described in 2.6. Manual tests require a guideline for how the test should be performed. Similar to Cucumber, the test issuer defines a set of steps for the tester to go through. The Test issue in Jira can be used for both automated and manual tests.

3. **Test Set**
   A test set [18] is a Jira issue that handles the organising of tests. It is a way to group two or more tests into a logical collection. For example, a test set called "security" might contain all tests that deal with security.

4. **Planning tests**
   The idea of the test plan [19] issue is to help the user manage their test suite. A test plan can contain several test executions, tests and test sets.

5. **Test Execution**
   A test execution [20] issue provides a context for tests that have been or should be executed. When a test belongs to a test execution, that test becomes a test run. A test run is then an instance of a test within the context of a test execution. The definition of a test can be changed without changing the test run. By keeping the test run and the test definition separate the test runs become consistent with the test execution. An example of this is shown in figure 3

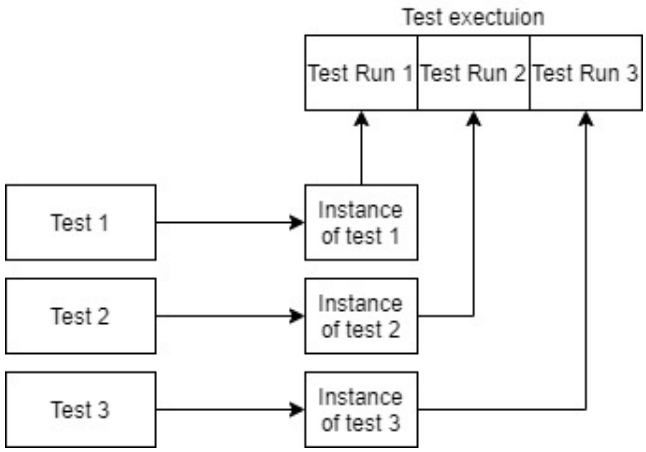

Figure 3: Test execution.

2.6 Cucumber

Cucumber [21] allows a user to write tests in plain text. Cucumber then reads each step of the test and verify that it executes correctly . An example of a Cucumber test is shown below.

```
1 Feature: Add two integers
2     Scenario: User wants to calculate a sum
3       Given the application is running
4       When the user input integer A
5       And the user input integer B
6       Then calculate the sum A+B
```

The *feature* [22] keyword gives a high-level overview of some system feature. In this example, the feature is adding two integers. One feature can contain several scenarios. A *scenario* is an example of software usage. The example above shows an example of a user wanting to calculate a sum.

A scenario [23] can contain several steps. Steps are keywords that describe some event, context or outcome. The example above contains four steps. *Given* is a step that initiates the system. *When* defines some action or event that occurs. For example, a user interacting with the system. *Then* is a step that should describe the expected outcome of the previous steps. *And* is a helper step that makes the scenario read more fluidly.

These steps are part of Gherkin[21]. Gherkin is a language that defines the grammar rules that the scenario must follow such that Cucumber can understand the scenario.

2.7 JUnit

JUnit [24] is a test framework for writing tests. JUnit consists of three main pillars the first is *Assertion*. Assertions enable testers to assert that some function or statement produces an excepted result. Next is the *test fixtures*, these are used to share test data. The final pillar is *test runners*. The runner executes all tests in the same package as the runner itself.

2.8 Code Coverage

Code coverage [25] is some the % of code that is covered by automated tests. Code coverage is obtained by analysing the running program and combine that information with the test suite to create a report on the code coverage.

2.9 OpenClover

OpenClover [26] is a tool that measures code coverage in Java. The code coverage report supports different formats such as XML or HTML[27]. OpenClover combines three types of coverage [28] statement, branch and, method coverage:

- Statement - Checks if a statement in the code is executed or not.
- Branch - Measures if the flow control branches. For example, if statements. This type then checks if the tests cover all paths.
- Method - Analysis if a method was executed or not during execution.

2.10 Postman

Postman [29] is a platform that allows users to analyse REST APIs. The idea is to streamline the API development process so that developers do not have to write any code to test an API. Instead, developers can directly test the API with for example GET or POST requests in Postman.

2.11 Jira REST Java Client Library

The Jira REST Java Client (JRJC) [30] is an external java library developed to abstract the REST API and HTTPS needed for communication between the Jira server and the REST API.

2.12 The ISO standard

ISO standards [31] are agreed upon best practices developed by experts in different fields to create a unified approach to solve common problems. For example how to measure the quality of software or how to secure sensitive data.

2.13 ISO/IEC 25010:2011

The ISO/IEC 25010:2011 standard [32] defines a product quality model which categorises a system into eight characteristics. For each characteristic, there exists one or more sub-characteristics. These are distinct components that aim to create a measure for software quality. The standard notes that it is not practical to try and measure every characteristic and its related sub-characteristic. Instead, the model should be tailored and based around the requirements of the user. The model can then be used as a checklist to see how many relevant sub-characteristics the system or product upholds.

2.14 Confluence

Confluence [33] is a tool that exists within the Jira platform to share knowledge and documentation. Documentation for Jira plugins such as Xray is available in the Confluence workspace [12].

2.15 Google Lighthouse

Google Lighthouse [34] is an open-source tool that can perform audits on web pages. The Lighthouse gives the web page a score based on five categories that can be turned on or off. The scoring system aims to evaluate how well well a web-page lives up to certain characteristics such as contrast ratio or screen reader friendliness. The score is ranked 0 to 100 where 100 is the best possible score for each category.

2.16 OAuth

OAuth [35] is an open-source protocol that uses public-key cryptography for user authentication. The idea is to use OAuth to transfer user privileges to some client, such as a REST API client. That client is then allowed to make requests on behalf of the user.

2.17 Maven

Maven [36] is a tool based on the Project Object Model(POM). POM [37] is an XML file that contains project information and configurations that are needed to build the Maven project. The idea of Maven is to simplify and uniform the build process. Maven also helps developers share and publish there project information and configuration [38].

2.17.1 Active profiles

A profile [39] is a set of configurations parameters for plugins that execute at build time. An active profile is a profile that is configured inside the *activeProfiles* section in the *settings.xml* file. This profile is activated by default.

2.17.2 Maven Surefire Plugin

Surefire [40] is a plugin for Maven that can run the unit tests of an application. After test execution, the plugin then constructs a report showing the results of the tests. When the plugin finishes the report outputs an XML and a plaintext file containing the result.

2.18 Continuous integration

Continuous Integration (CI) [41] means to merge several developers local branches into a single software stored in a system that implements version control. The merging of branches makes sure that each developer has an up to date branch and that their work remains updated. Before merging CI asserts that the software is functioning correctly by the use of automatic tools. These tools are usually automated tests but can also be code reviews or syntax style checks.

2.19 Continuous Delivery

Continuous delivery [42] streamlines the release process. The aim is to deliver changes to the end-user as quickly as possible. The process becomes streamlined by automating every step of the process. Code changes are delivered to a staging environment where automated tests make sure the changes don't negatively affect other code. The changes are then pushed to the release environment and available to the customer.

3 Methodology

This chapter describes the workflow of the project. The workflow includes how the project divides into phases, how the software was analysed, evaluated and finally, the tools used to complete the project.

3.1 Literature study

For the literature study to be source-critical, a source criticism checklist was used [43]. The checklist defines a few points that the reader should follow when reading information online. The points are as follows:

- Who is the author of the source?
- Why was the source created?
- Is the source authentic?
- Is the information provided by the source relevant or outdated?
- Is the source independent, or does it rely on other sources?
- Is the information provided by the source biased? Do other sources contradict the information provided?

3.2 Procedures

The project follows the funnel model described in [44], which defines a funnel where the top of the funnel is the most general relevant data, and as the project moves through the funnel, it gets more specific.

The project was divided into four phases. A pilot study, research, construction and analysis of the software.

3.2.1 Pilot study

The pilot study aims to build a basic understanding and vocabulary that is needed to understand research and discuss ideas at a higher level with the company.

3.2.2 Research

In this phase, current academic research is studied with a systematic literature examination to gain knowledge about manual and automated testing. This project follows the information retrieval process suggested in

[44] so that the process can be effective and correct. The process defines six steps that are summarised in figure 4.

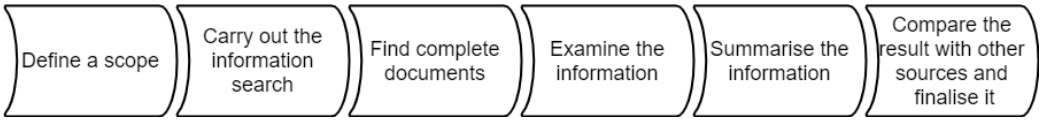

Figure 4: Research process.

The research was conducted by scanning scientific databases such as Google Scholar, IEEE, and, Elsevier and various internet sources.

3.2.3 Analysing Xray

To evaluate the quality of the Jira plugin Xray the ISO/IEC 25010:2011 standard is used. For this evaluation, five out of the eight characteristics defined in the standard are selected. One or more questions or examinations have been created that are relevant to the definition of the sub-characteristics. A sub-characteristics is relevant if it can be measured by an external analysis and is of interest to Knowit. These examinations and questions are later used in the report to evaluate Xray. The definitions and related examinations are as follows:

1. **Functional suitability**
   Functional suitability aims to examine how appropriate, correct and complete a product or system is for the users of said product or system [32]. The relevant sub-characteristic is:
   - *Functional completeness:* How complete is the set of functions? That is, to what degree can a user perform test management in Xray?
2. **Usability**
   The usability characteristic examines how accessible a product is. A product has high usability if a user can achieve specified goals with reasonable effectiveness, efficiency and satisfaction. Usability has several sub-characteristics [32]. The relevant ones for this analysis are:
   - *Learnability:* How can a user learn the product? Does product documentation exist? Are there user guides with examples?
   - *Operability:* How is the product controlled? Keyboard shortcut, clicking, drag and drop?
   - *User error protection:* Can a user delete important data by accident?

- *Accessibility:* Is the default options accessible for people with disabilities?

3. **Reliability**
   Reliability is a measure of how a product behaves over time and how it handles and recovers from faults [32]. The relevant sub-characteristics are:
   - *Recoverability:* How does the product deal with faults? Can lost data be recovered?
4. **Security**
   To have security means that a users information and data is protected and a user is only allowed to read/write data that is appropriate to the user's privilege level [32]. The following sub-characteristics are relevant:
   - *Confidentiality:* Is the data in the Jira platform encrypted? What encryption does Jira use?
   - *Integrity:* How does Jira prevent unauthorised access? What types of authentications are there?
5. **Portability**
   Portability examines how well environments can interact with each other and how data can move from one environment to another[32].
   - *Adaptability:* Can the data in Xray be transferred and used in other environments? How can Xray interact with other systems?
   - *Installability:* Is the installation process for Xray straight forward or are there unnecessary steps?

Some characteristics were left out because it was not feasible to examine them from the perspective of the end-user. An external analysis of the product examined the characteristics that were chosen.

3.3 Test prioritisation algorithm testing

The operational correctness of the test prioritisation algorithm was ensured by constructing four cucumber tests. Each Cucumber test contained several JUnit tests which assert that methods were returning correct values and that XML data was correctly extracted.

3.4 Survey

Reliable survey questions were designed by following three primary principals defined in [45]. Adequate wording, consistent meaning and well-defined terms. The idea is that the questions posed in the survey should mean the same thing to all participants. Also, the words should be chosen such that they are relevant and understandable by the participants.

To create an electronic survey and get data from participants, the online service Qualtrics was used. The survey has seven open questions regarding Xray that would take participants with some experience with Xray about 5-10 minutes to answer. This survey was sent as an electronic mail and distributed to different companies who are in the process of moving from manual to automated testing.

3.5 Agile work

At the beginning of the project, a Gant chart was made to create structure and break the project up into tasks. A timeline was used to keep track of the progress made and what needs to be done. The chart was created with the online application TeamGantt. Jira was used to create a Scrum board and divide the tasks in the Gant chart into smaller, more manageable pieces.

3.6 Tools

The tools used to achieve the result are presented below.

3.6.1 Version control

The online service Github was used to store the source code for the project. This was done so the project could be worked on independent of the workstation.

3.6.2 Developing environment

To produce the work presented in chapter 4 the integrated development environment IntelliJ was used together with the operating system Windows 10. The software was written in the programming language Java. For API calls to the REST API, the project uses Jira REST client for Java. Test design was achieved in the Jira platform using Xray issues with Cucumber and later implemented in Java with the JUnit framework. To compile the Java code Maven was used.

3.6.3 Working with the REST API

In order to test and learn the REST API provided both by Jira and Xray, the Postman application for Windows was used. Postman was used for sending GET, and POST to the REST API. These requests were then analysed and studied to learn the REST API.

4 Construction

This chapter demonstrates Xray integration with Maven and the REST API. Furthermore, this chapter presents an implementation of the algorithm found for prioritising automated test cases in chapter 5.1.3. Finally, this chapter shows how prioritised test cases can be uploaded to Jira with the Xray API.

4.1 Xray with Maven integration

Maven integration was achieved with the Xray-Maven-Plugin. An active Maven profile was created to avoid having to type log credentials. Before running tests with the Maven plugin, an Xray Test Project have to be created in Jira. The test project creates a *project key* which is a requirement for the Xray-Maven-Plugin. A project named MavenIntegration with the key MAV was thus created. A Maven projects pom.xml was edited to enable the Xray-Maven-Plugin along with JUnit and Surefire. See appendix A.1 for details on how the pom.xml was configured. Standing in the Maven project directory, a user can run the following command to upload test results to the MavenIntegration project:

```
1 mvn com.xpandit.xray:xray-maven-plugin:xray
```

In order to run all the tests and then upload the results, the following command can be used:

```
1 mvn clean package surefire:test com.xpandit.xray:xray-maven-plugin:xray
```

Tests result can be uploaded to a specific test execution and / or test plan by specifying the Xray properties *xray.projectKey* and *xray.testPlanKey* in the pom.xml file

To create a new issue with custom values in Jira, a user can use the multipart endpoint can be used. Before running the command, a JSON file needs to created. This JSON file contains the issue field values that are needed by Jira. See appendix A.1 for details on an example configuration. Furthermore, the following lines need to be added to the pom.xml file

```
1 <xray.testExec-fields.path>${basedir}/info.json</xray.testExec-field.path>
```

The edit specifies the path to the JSON file. Now the user can run the following command to create a new Jira Issue with predefined values for the issue fields:

```
1 mvn clean package surefire:test
  ↪ com.xpandit.xray:xray-maven-plugin:xray_multipart
```

4.2 Extracting coverage data from OpenClover

OpenClover outputs several HTML files containing the test coverage information. This information was collected and stored in an XML file. The XML file could then be parsed into the Java application and used as input for the test case prioritisation algorithm described in 4.3

For the implementation of the prioritisation algorithm, a unit represents a class method. A custom class was created to store the test cases. The primary function of this class is to store the methods it covers in a hash map. The hash map key is the unit name, the occurrence of that unit together with the statement coverage, is stored as a Pair and mapped to the previously mentioned key. The test prioritisation algorithm can use the function *get* from the hash map to fetch the number of occurrence for a given unit in O(1) time.

4.3 Implementing the unified test prioritisation algorithm

This sub-chapter shows the implementation of the test case prioritisation algorithm presented in chapter 5.1.3.

The algorithm consists of four major steps. The first step is the initialisation step. Variables and data containers are set to their initial state. In the second step, the algorithm finds any not previously selected test case t_k. After t_k is found, the algorithm checks the number of occurrences of each unit covered by t_k. If a unit m is covered by t_k then the value f_p is normalised in the range $0.6 <= f_p <= 0.95$ and used to calculate the probability that t_k contains any undetected faults. In the third step, the algorithm looks through the remaining test cases and searches for a test case t_l that has a higher probability of containing undetected faults than t_k. If it finds such a test case, t_l then t_l is set to have the highest priority. Then the probability that a unit covered by t_l contains an undetected fault is updated. The algorithm then restarts at step two and finds the test case with the second-highest priority. The process continues until all test cases have been evaluated. The flow chart for the algorithm is shown in figure 5 and for full implementation details see appendix B.2

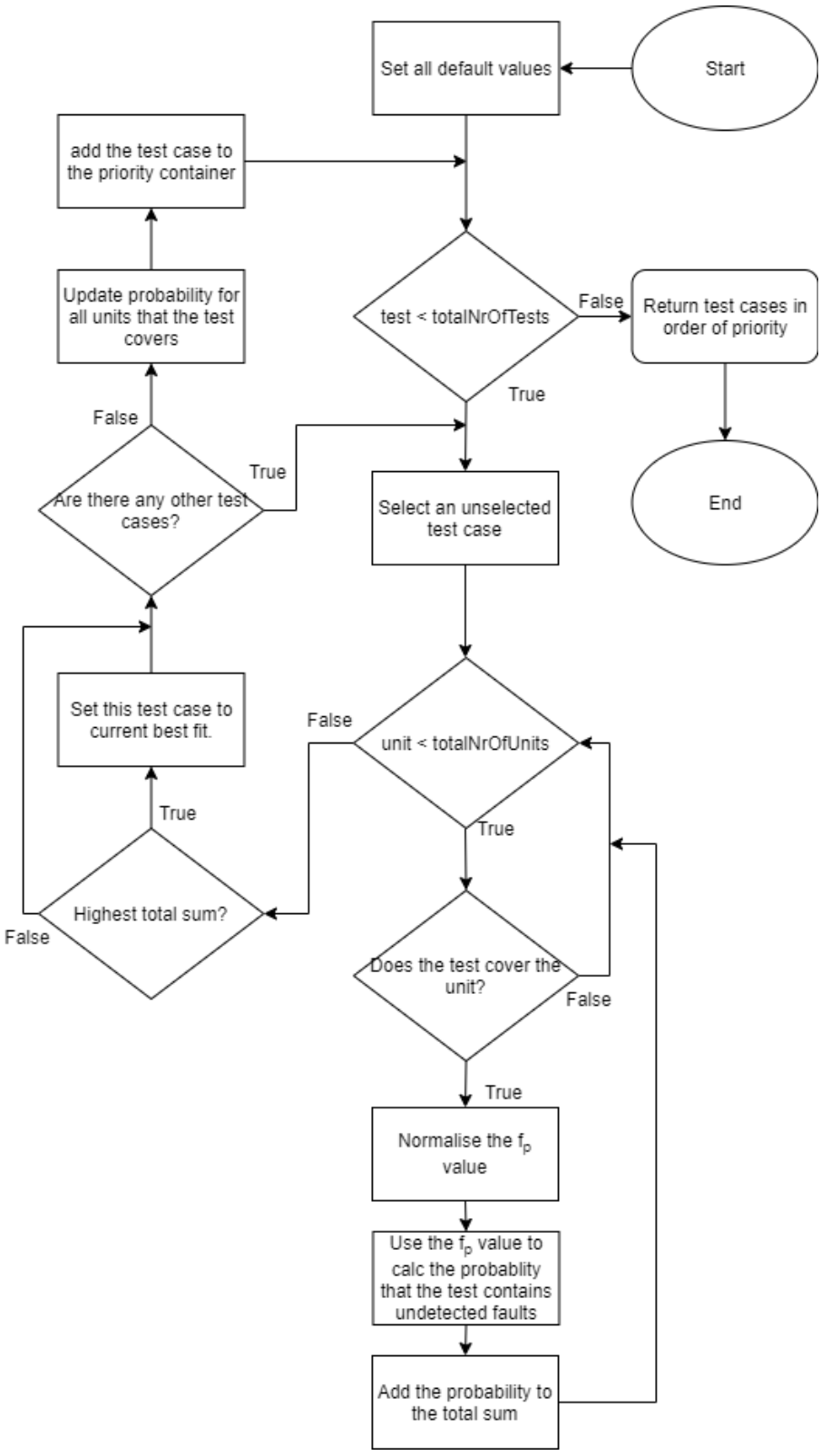

Figure 5: Test prioritisation algorithm.

4.4 Updating test priority with the Xray API

To connect to the Xrays REST API, the implementation uses the officially supported JRJC Library. An instance of the REST client was obtained with the *Asynchronous Jira Rest Client Factory* method call. The method call creates a factory for the JRJC. The factory then authenticates the user with basic HTTP authentication. The JRJC have a method for getting the *Issue Client* this method can update the priority of an Xray issue given the issue key and input to be sent to the REST API.

For implementation details see appendix B.1.

4.5 Automated tests in Jira

A project was created in Jira to enable the Xray issue types. The purpose of the project was to test the three major components of the software. The major components are the XML parser, the prioritisation algorithm and the Jira REST client.

4.5.1 Setting up Xray tests with Cucumber in Jira

To test the XML parser in chapter 4.2 one Test issues, and a Test Execution issue of the type Cucumber was created as well as a mock file containing data on which the XML parser can be tested. The Cucumber scenario created in Jira looks as follows:

```
1 Feature: XML parser
2     Scenario: Testing the XML parser.
3     Scenario Outline: As a user I want to be able to extract tests from
  ↪   an XML file
4     Given That a user wants to parse a XML file
5     When Test cases have been extracted
6     Then A test case <nr> should have a "<key>"
7     And A test case <nr> should have "<units>" in them
```

A requirement for the *Scenario Outline* is too have a data table filled with test data.

Table 1: Test data for the XML parser test.

nr	key	units
0	AP-1	XMLParser.getProjectStatements
1	AP-2	TestCase.TestCase

In table 1 the "nr" column will replace the <nr> variable in the test scenario. Same goes for "key" and "units". This test case parses an XML file, extract test cases and then check if those test cases are valid.

For the prioritisation algorithm, an additional test issue is created and added to the test execution created previously.

```
Feature: Prioritisation algorithm
    Scenario: Test the prioritisation algorithm
        Scenario Outline: As a user, I want to be able to priorities my
↪ tests.
        Given That the XML has been loaded test cases correctly.
        Then A test case "<key>" should have a <priorityWeight> after
↪ being prioritised
```

Table 2: Test data for the prioritisation algorithm.

key	priorityWeight
AP-1	2.52
AP-2	1.92

The prioritisation algorithm needs input data, so a mock XML was created in order to supply it with input. The XML is then parsed and test cases extracted. The test cases are then prioritised with the prioritisation algorithm. If successful, the test case with the mentioned keys in table 2 should have a corresponding priority weight.

To test the JRJC a pre-condition was constructed. This pre-condition gets valid user credentials from a file. Then a test for the JRJC was created and added to the test execution previously created.

```
Feature: Jira REST Client.
    Background:
        Given That there exists a file for user credentials
        Then fetch the credentials.

        Scenario: Test that the Jira REST Client can change the priority
↪ of an XRAY issue
        Scenario Outline: As a user, I want to be able to update my Xray
↪ issues.
            When A user wants to give a test a new priority.
            Then A <priorityWeight> should be mapped to a <XrayPriority>
            And A test case "<test key>" should be given a <XrayPriority>
```

Table 3: Test data for the Jira REST client

test key	priority weight	Xray priority
DEMO-1	1	5
DEMO-2	3	4
DEMO-2	5	3
DEMO-1	7	2
DEMO-1	9	1

In table 3 in the column "Xray priority" a 5 represent the lowest Xray priority and a 1 represents the highest priority.

Testing the JRJC was done by creating a, a mock user and having the mock users credentials stored in a file. Those credentials were fetched in the precondition and authenticated to the Jira server. The two tests, DEMO-1 and DEMO-2, were updated with new priorities with data from the data table.

Finally, a test was created for the test case container class. This test required no mock data since all the data from table 4 could be used in order to test if the class functions correctly. Implementation of the class can be found in appendix B.3.

```
1 Feature: TestCase container
2     Scenario: Tests the TestCase container class
3     Scenario Outline: As a developer, I want to be able to store test
  ↪ cases in a container class
4         Given that a developer wants to create a test case with a
  ↪ "<key>", "<units>", <coveredStatements> and a <priorityWeight>
5         Then A developer should be able to retrieve the test case
  ↪ "<key>".
6         And A developer should be able to see how much <priorityWeight> a
  ↪ test case have.
```

Table 4: Test data for the test case container

Test key	Units	Covered statements	priority weight
DEMO-1	method1	5	2.4
DEMO-2	method2	6	3.5

All tests were then exported from Jira in there corresponding .feature files and implemented.

5 Results

This chapter shows the result of the literature study and the examination of Xray. The aim of the sub-chapter 5.1 is to present the findings regarding what the role of automated testing is. Sub-chapter 5.2 shows the result from testing implementation of the prioritisation algorithm. Sub-chapter 5.3 gives the result of the examination of Xray and a survey done on practitioners that are moving from manual to automated testing and using Xray as a test management tool.

5.1 Automated tests

A study on CI [46] shows that automated testing has a central role in succeeding with CI. The goal of CI is to provide both productivity and quality. Productivity is improved since more code can be merged into production and automatically tested to be compliant with the current codebase. Quality is an attribute that comes from running tests on the codebase and reducing human error and providing some guarantee that the software is bug-free.

Automated tests are a core concept in the test-first methodology[47]. The idea of this methodology is first to write an automated test then write code that passes the test. There are a few software development processes that have adopted this methodology. One of these is Test-driven development. Another is Behaviour-driven development[48].

5.1.1 Benefits and limitations of automated testing

A quantitative study [46] done in 2015 examined 246 GitHub projects and how CI has affected those. The focus was on how productivity and quality were affected by adopting CI and in turn, automated testing. The result achieved in this study points to CI helps both with increasing productivity and quality. The study noticed a significant increase in merging pull requests and attributes this to the usage of CI. Furthermore, the study noticed an increase in bug reports after adopting CI, which suggests that CI improves the quality of code by bringing more bugs to the developer's attention.

An analysis of automated testing in a Continuous Delivery (CD) approach done at a company with six years of experience of CD [42] shows that there exist several pitfalls when a company makes a move to automated testing.

Among the most crucial pitfalls are:

1. *Abandoning manual testing:*
   When moving to automated testing, companies should not abandon manual testing. Manual testing still has use cases where automatic testing falls short. Such as exploratory testing in complex systems.
2. *Failing to write testable code:* The software needs to be easily testable. Attempting to write automatic tests for code that is hard to test can cause the pipeline to be blocked and ruin profits gained by automatic testing.
3. *Managing the test environment:* A lack in configuration management can send bugs to the production pipeline that are very hard to reproduce in the test environment.

A study conducted in 2016 [49] analysed how manual testing compares to automated testing with a focus on cost, time and, software quality. It examines different versions of the three different software. The study finds that the initial cost of implementing and maintaining automated tests in the three software is high. Furthermore, the research shows that given time, testing automation improves both reliability and maintainability of the software, which result in higher software quality. The increase in software quality was attributed to the code running through a suite of tests ensuring correct operations and reducing the risk of human error.

A comparative study [50] between research and practice shows that automated testing can improve reusability, repeatability and reduce the effort required when executing tests. However, the study also found that the initial cost for implementing automatic testing is high due to staff requiring training, cost of test automation tool and the maintenance of automatic testing. The study then suggests that this cost balances out with time, and the company can expect some return on investment.

An empirical analysis[51] of discussion boards for support issues regarding implementation of test automation frameworks examining 8769 posts shows that setting up the framework and test system was the primary source of problems for test automation. The two most significant thread types presented in the study were: problem reports, they are defined as: "Execution fails for some unknown reason". Another thread type, help requests were defined as: "Where the user asks for help on how to perform a certain task". In the problem threads, the most common issues were: 38% had problems developing tests scripts and problems with tool usage. 21.7% had difficulties understanding how to configure and implement the test framework. 29.9% had problems with there IT environment. In

the help request threads, the two most common requests were: Requests wanting help with developing the test script or tool usage covered 91.5 % of all help requests, 3.3 % wanted information about configuration for the test framework, the remaining 5.2% where miscellaneous help requests.

Summary of benefits
Improved software quality: Automated tests allows developers to get notified if modified code have stopped the software from working.[46], [49] , [50], [52] *Increased productivity:* Since the tests run automatically developers have more time to develop the codebase.[46], [53] *Reliable software:* Automated tests can act as a proof that the software is free from a percentage of bugs.[49], [46], [50] *The software is easier to maintain:* Code becomes easier to maintain, tests runs automatically and developers can be certain that the code is working as intended.[49], [50], [52] *Reduction of cost:* As companies becomes more familiar with testing and developers gain more experience with the tools the overall cost goes down. Automated testing also leaves more time for developers to focus on developing. [53], [50], [52]

Summary of limitations
High inital cost: The codebase might be untestable, and in need of significant refactoring, automated tests have to be created. Developers also require training and experience with the tools that automated testing requires.[49], [50], [53] *Framework complexity:* Depending on the framework used for automatic testing, difficulties can arise from implementing automatic testing if the framework is complex. [51], [52] *Proper management and team commitment is required:* There needs to exist the knowledge of the risks and requirements at both the management and developer level. [42], [54], [50], [53] *Difficult to maintain automated tests:* Developers needs to keep up with the updates of the tools they are using as well as updates in the software they are developing to keep the tests working as intended. [50], [42], [52], [53]

5.1.2 Automatic testing and manual testing

Two studies [54], [50] examines companies moving from manual to automated testing and the lessons learned. Most companies have a defined budget for testing, the number of automated tests, manual test executions and their expenditure are all part of this budget. Their combination can not exceed the budget. The authors suggest that the cost of automated testing is the number of different tests. The cost of manual tests is the number of test executions. This difference suggests that to keep the cost down and gain the most benefit. Automated tests should be used for evaluating modified but previously working code because the test is already written and need only needs to be executed. Manual tests are more suited to explore new features and find defects or bugs but also helps the software follow specific requirements such as the look and feel of a user interface. Thus the study suggests that the maximum benefit comes from doing a combination of manual and automatic testing.

In an industrial case study [52], the company Saab AB wanted to make a move from a manual system test suite to automated testing. The automated tests covered the same code blocks as the manual tests. What the researches found was that automated testing found more bugs than the manual tests. Thus automated testing improves code quality. However, automated tests can not completely replace manual testing as manual testing allows exploratory testing and uncovering new faults or defects. The study also shows that its challenging to make automated testing work for large complex system because of the many variables involved.

5.1.3 Prioritising test cases

Studies show that manual test case prioritisation based on historical test case data can be an efficient method to prioritise manual tests.[55], [56] The empirical study conducted in [55] suggests that a risk-driven approach can outperform the more traditional prioritisation methods for manual testing in a rapid release environment. Manual tests are usually performed in a black-box environment. That is, the testers don't have access to the source code. The study examines four older releases of the web browser Mozilla Firefox. It suggests that the proposed risk-driven approach is 65 % more effective than the traditional alternatives presented in the study. The idea proposed is to cluster tests based on if they have detected errors or not in the past. A test that fails often is noted as risky and have a higher priority. Tests that failed in the previous version becomes part of the cluster with the highest priority. Then comes the tests that failed two releases ago but not the most recent version. If the tests does not exist in highest-priority clusters they are put in the cluster with

second-highest priority and so on. After all the failing tests have been assigned the successful tests from the previous release are append to the highest priority cluster and the successful tests from the release before to the second-highest priority cluster if they do not already exist in the highest priority cluster and so on. Several Test Case Prioritisation (TCP) techniques to prioritise test cases within a given cluster exist. A more recent one is diversity-based TCP which tries to obtain maximum test diversity by prioritising tests that uniquely covers as much code as possible. This technique was noted as the most efficient and reliable when prioritising manual test cases in the above study.

A common algorithm for white-box testing is the Additional Coverage Strategy (ADC) this is a greedy algorithm. When the algorithm selects a test case, it takes into consideration how much coverage previously selected test cases have already covered. The algorithm then selects the test case that covers the most units not previously covered. A unit can be a statement, method or complexity. That is, a unit should be something that can indicate coverage. The processes repeats until all units have been covered by at least one test case. Once all units have been covered, all coverage information is reset, and the algorithms process the remaining test cases. [57]

Several studies [58], [59], [60], [57] have shown that the additional strategy is close and sometimes even with an optimal coverage-based solution that is measured using the Average Percentage of Faults Detected (APFD) metric.

A study from 2013 [61] suggested that there exists a weakness in generic ADC. Suppose a test case t that covers a fault f. If t can not detect f then the there is a significant delay before ADC considers a test case t' that can detect f. To solve this, the author suggests combining additional with total coverage. The Total Coverage Strategy (TCS) does not share the previously stated weakness since for every test case t it considers all units. However, the total coverage strategy only counts the number of units each test covers and does not consider units covered by other test cases. The algorithm that the author suggests was revised and improved in 2014 in the paper [62].

The algorithm suggested in [62] unifies the ADC and TCS. This is done by defining a parameter f_p. The parameter is updated for every unit in order to instruct the algorithm how close it should run to pure ADC or pure TCS. $f_p = 1$ is the ADC and $f_p = 0$ is the TCS. The authors suggests that the f_p parameter should have a value in the range $0.6 <= f_p <= 0.95$ for optimal APFD. This was according to the empirical study carried out in the report.

Furthermore, the empirical study shows that by differentiating the f_p value, this approach can be more effective than the ADC or TCS with the same cost as the ADC. That is, for each unit a new f_p value is calculate in the range $0.6 <= f_p <= 0.95$ and used to determine the probability of the unit containing undetected faults.

5.2 Testing the implementation of the prioritisation algorithm

The result from running the tests described in 4.5 are shown in figure 6. The green checkmark indicates that the test successfully completed. The time to the right indicates the time it took to run the tests in milliseconds. In figure 6 some tests appear several times, this is because the same tests are executed but with the different values found in the data tables in chapter 4.5.

Figure 6: Result of automated testing

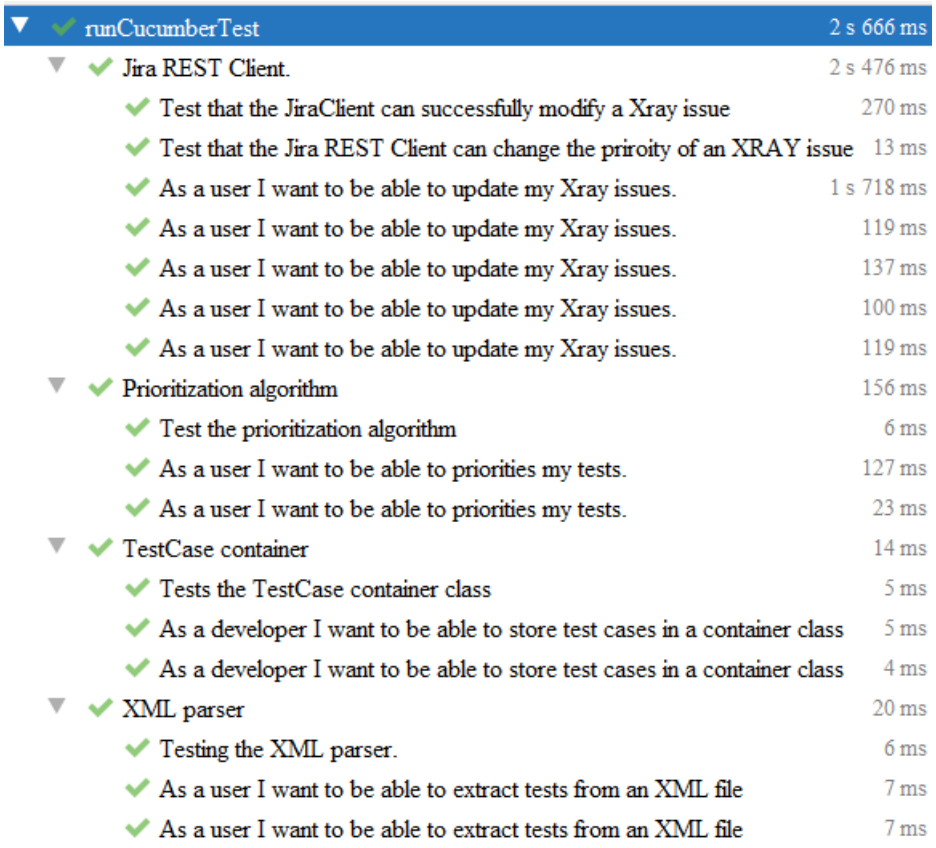

Table 5 shows how the prioritisation algorithm gives a test a priority weight and what Xray priority that priority weight is mapped too. Xray defines five priorities, where 1 is the highest, and 5 is the lowest. As mentioned in chapter 5.1.3, each unit has a value between 0 and 1 that number is the probability for that unit to contain an undetected fault. The priority weight is the sum of all units probabilities covered by a Xray test. Figure 7 shows which methods are covered by which tests.

Table 5: Tests and their priority weight mapped to an Xray priority

Test name	Priority weight	Xray priority
PrioritisationTest	5.4	1
JiraClientTest	2.96	3
TestCaseTest	2.73	3
XMLParserTest	0.44	5

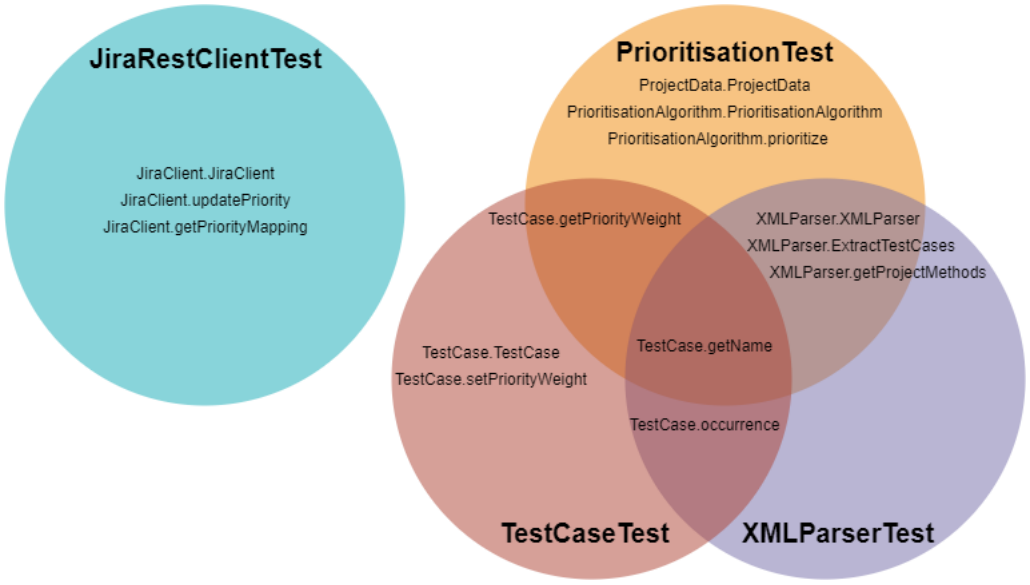

Figure 7: Tests and their covered methods.

5.3 Xray as a test management tool

This sub-chapter presents an external examination of the Xray plugin for the Jira server using the ISO/IEC 25010:2011 standard.

5.3.1 Functional suitability

Xray exists in the Jira platform, and Xray uses Jira issues for test management. This connection allows users to connect the Xray tests to other

tools within Jira. Xray also provides the users with an overview of the testing suite with the Xray Report, which lets the user analyse status and success rate of tests in the testing suite. Another Xray feature in Jira is the Test Plan Board. This feature aims to help the user create plans for there testing. The planning board does this by letting the user organise tests in directories. A summary report is presented to the user when a directory is clicked. The Xray test repository is similar to the test plan board instead of organising test plans a user can organise regular test cases.

Xray test issues can have three different types of manual, Cucumber and generic. These types enable both automated testing and manual testing.

Xray allows its user to set a test priority manually for the following issues: Test, Test Plan, Test Execution, and Test set. Since Xray provides a REST API, the algorithms presented in 5.1.3 can be used together with a POST request to the REST API to create a test and set a priority automatically.

5.3.2 Reliability

- **Recoverability**
  The responsibility of storing and backing up data falls on the user. Jira can enable a user to create a backup of the application data. This backup does not cover attachments, Jira configurations or Jira plugin installation files. The backup is by default, a manual process. However, the user can create a "Backup Service" which automates the process and creates backups at specified intervals. A stored backup can then be imported to the Jira server to recover from fault or loss of data.

5.3.3 Usability

- **Learnability**
  The main documentation for the Xray plugin exists in Confluence. A user can not view the documentation from the Jira platform but must instead navigate away from Jira to read the documentation. The Xray issue types do have a short description in the Jira platform, but it provides little information. For example a *test set* is described as follows "Test Set - Represents a Test Set"

  The documentation at Confluence contains step-by-step guides for installation and usage. Here a user can get a more rich explanation of the different issue types that the Xray plugin adds to the Jira platform.

- **Operability**
  From the perspective of the Jira platform, a user can navigate it and

manage Xray issues by either clicking or using the predefined keyboard shortcuts.

From a REST clients perspective, a user can manage their Xray issues by, for example integrating with Maven as shown in chapter 4.1. A set of console commands then manages Xray issues. Using the Xray REST API, a user can create there own tool for managing Xray issues. An example of usage is shown in chapter 4.4 where the REST API is being used to prioritise test cases automatically.

- **User error protection**
  When attempting to delete an issue from the Jira platform, a warning displays for the user. The warning informs the user that the action of deleting an issue is irreversible. However, a user requires the privilege to perform this action. The delete privilege is only set to administrators by default, and an administrator can configure the privileges for all users.

  Using the REST API, a user can delete issues without any warning. However, performing the delete requires the appropriate user permissions.

  If a user makes a mistake when creating an Xray issue, all fields for that issue can be edited in the Jira platform. The REST API support editing of most fields except the issue type.

- **Accessibility**
  From the Jira settings, a user can choose to underline links to help with visibility. The user can also turn on patterns for issue statuses. This can help the user distinguish issue statuses without the need to see the colour.

  The Lighthouse tool was executed on the welcome page, dashboard, project, and the Xray sub-pages. The audit score for the accessibility category are presented in the table below:

Page	Score
Dashboard	93
Projects -> Summary	93
Projects -> Issues	88
Projects -> Issues -> Test Execution Details	84
Projects -> Xray Reports	81
Projects -> Xray Test Repository	84
Projects -> Xray Test plan	88
Projects -> Add-ons -> Tests	81
Projects -> Add-ons -> Test Sets	81
Projects -> Add-ons -> Tests Executions	80
Projects -> Add-ons -> Tests Plans	81

From the table, the average score for pages related to Xray is 85.

5.3.4 Portability

- **Adaptability**
  Xray interacts with the Jira platform and adds new issues that the user can manage. Jira can generate reports and chart with data from the Xray issues. The user can also choose a plugin for Jira that interacts with Xray issues. All Xray issues can be exported from the Jira server to XML, Word and HTML. For the Test and Test execution issue, they can also be exported to CSV and Cucumber. A test set can be exported to Cucumber as well. Xray issues can also be exported using the REST API or the Maven integration. A Jira user can also choose to customise visual fields, such as the navigation bar, changing colour or contrast.

- **Installability**
  To install Xray on a Jira server, the user must first have a Jira account. The user can then download the installer for the server and launch it. The installer follows a basic installation process and prompts the user for basic configuration choices. Once the server has been installed, Jira offers to help set up the servers configuration. The setup involves creating an administrator account, language options and showing a sample project to help the user get started. Once that's done, a user can head to the Jira market place and download the Xray plugin. The installation process for Xray manages itself. The user is presented with a short tutorial for how to use it once the installation is done.

5.3.5 Security

Before connecting to the Jira server, a user has to tell the Jira server who they are. Once the user has authenticated themselves, they need the necessary authorisation to perform actions. To help with user integrity, a user can authenticate to the Jira server with OAuth, basic authentication or cookie-based authentication. The REST API can use any of these three. The maven integration and default login to the Jira server is restricted to basic authentication.

If a user fails to authenticate three times, a CAPTCHA is displayed. The CAPTCHA prevents the REST API and maven integration from working. The user needs to manually login to the server completing the CAPTCHA.

During the examination, there was no indication that the data on the Jira server was encrypted. A member of the Atlassian team said the following about encryption on Jira servers: *"Natively, Jira is not able to encrypt data such as issues/attachments. Jira does have the ability for you to use issue-level security in order to further restrict users from seeing specific issues and their attachments beyond the scope that the standard permission schemes encompass on a per-project basis. But this isn't truly encryption, but rather an additional layer of permissions."* [63]

While Jira does not offer encryption, plugins [64] for Jira do. For example, a user can install the "Encryption for Jira" which provides well-known encryption techniques.

5.3.6 Survey results

This sub-chapter presents the result of the survey. Participants were asked the following eight questions regarding Xray:

1. What is your role within the company?
2. How many years of experience do you have with testing?
3. Are you experienced with automated or manual testing or both?
4. Do you think that Xray provides a sufficient environment for test management? Please elaborate on your answer.
5. What pitfalls do you see when moving to automated testing? Does Xray help solve any of these pitfalls?
6. What benefits do you see when moving to automated testing? Does Xray improve upon or bring any of these benefits?
7. Have you noticed any major problems with Xray? If so, please elaborate

8. What changes are needed in Xray to make it a more applicable?

Q1-Q3: All participants had at least 5 years of experience with testing in a management or leadership role. One participant had previous experience with manual and automated testing where the others only had experience with manual testing.

Q4: All participants agree that Xray serves as a sufficient tool for test management. Many of the participants also made a note of how the interaction with Jira aids the test management process. This was attributed too that the same tool is used for several parts of their development projects.

Q5: Four out of five participants felt that they didn't have enough experience to answer this question. However, one participant said that management might have high expectation on automated tests and are not considering the negative aspects of moving to automated testing. The participant then goes on to say that Xray can help demonstrate both the negative and positive aspects and thus help managers gain a clearer picture of the testing situation.

Q6: Three out the five participants thinks that automated testing improves productivity, regression testing and reduces the time spent on testing. Two participants were not sure what benefits to expect from automated testing.

Q7: Three out of the five participants noted minor usage problems that they attributed to having little experience with Xray and expected it to get better with time. Two participants said that the reporting feature for tests could use some improvement, and it was not always easy to see the desired information.

Q8: Participants saw a few improvements that could be made to Xray such as
Reading the reports on the Jira dashboard instead of having to view the Xray reports separately.

Creating a link between Confluence and Xray such that an Xray issue can be created and its description is directly linked from Confluence instead of having to copy it.

Finally, some participants noted that better documentation of features and best practices for Xray usage is needed.

6 Analysis of Xray

This chapter analyses the result presented in chapter 5.3

One of the limitations of automated tests shown in 5.1.1 is that proper team management is required as well as the difficulty to main automated tests. Since the Xray issues are Jira issues, this can help test leaders and test managers manage their teams by assigning issues to developers when something requires maintenance. Priorities can also be set for these issues, and having test data and test status integrated with scrum boards can help facilitate agile development.

As mentioned in 5.1.2 when a company moves to automated testing, manual testing still plays an important role. Xray supports both so project teams can manage both their manual and automated testing in Xray.

While Jira did offer the option to backup data, it was not on by default. If this feature were to be a part of the introduction to Jira, user data becomes more securely stored.

In the survey presented in chapter 5.3.6, **Q7** participants attributed many problems to their lacking experience with Xray. One of the issues participants found in **Q8** were that documentation was lacking. If Xray were to work towards integrated documentation in Jira, users might have a better experience. Integrated documentation would enable a user to remain in the Jira platform and have relevant documentation for the specific feature they're trying to use instead of having to look the feature up at another web-page.

As shown in 5.3.3, the Xray pages in Jira got an average accessibility score of 85. The score got a bit lower because the navigation bar was present for all sub-pages and the audit tool noted on every test that the "create" button had poor contrast ratio. Another note from Lighthouse was that names and labels were missing from buttons, forms and links, making it harder for a screen reader to assist the user.

In chapter 5.3.4, the adaptability of Xray is shown. The result shows one of Xrays strengths. A user can call the REST API to manage almost every Xray feature that exists in the Jira platform. Xray and Jira are customisable to individual user need increasing their accessibility. Such as making test prioritisation automated as shown in 4.3.Since Xray supports several different types of exports, a user can make custom graphs or charts enhancing customizability.

The installation for Xray and Jira was straightforward. There was almost no user interaction, and the installation process managed most of the work, creating a plug & play experience.

Chapter 5.3.5 shows that there exist three types of authentication methods. The choice of authentication is up to the user, making it possible for the user to make a bad decision. The Xray documentation is written for basic authentication with only a brief mention of the OAuth. The documentation in its current state can cause some users to ignore the OAuth since there exist no tutorials or guides for it with Xray. To improve this Xray could provide documentation for different authentication methods and inform the user about the strengths and weaknesses of each.

Furthermore, in chapter 5.3.5, the Jira server only encrypts passwords and does not inform the user that the stored data is not encrypted. Unencrypted data brings to light some ethical aspects, such as when a customer trust a company with there information. The company then uses the Jira Server to create some product for the customer. Sensitive customer information might be inferred from names, summaries or description of issues. This leakage of information can violate the customer's trust in the company. To better the user experience, Jira should be more clear about that the user data is stored unencrypted. Thus it might be in the interest of companies and customers for Jira to officially provide encryption.

7 Discussion

This chapter discusses the goals, research study, survey, prioritisation algorithm implementation and future work.

7.1 Goals

The goals of the project have been successfully achieved. In chapter 1.4 six questions were asked. The project answers question one to four by conducting a research study. Question five and six were answered by examining the different parts of Xray and suggesting implementations of test prioritisation.

7.2 Research study

Chapter 5.1.1 presents several benefits and limitations to automated testing. A company can expect to see increased productivity, have reliable software and improved software quality. These advantages are also what most participants from the survey were expecting, according to **Q6**. However, there are also a few pitfalls that companies must watch out for such as developers failing to write testable code. Bad code can make a move to automated testing more difficult since most of the code would have to go through a major refactor. A combination of bad code and inexperienced developers can cause a high startup cost for automated testing. Several studies showed in 5.1.1 that many companies don't consider the startup cost of automated testing. From the survey responses in chapter 5.3.6, **Q5** shows that most participants who had test manager experience also don't consider the negative impact that automated testing might have. Neglection of the startup costs can be dangerous and exceed companies testing budget if not considered. Developers also need to be assigned to maintain these automated tests, or the tests might become obsolete and more of a burden than a benefit. The expectation of test managers shows that there is a need to educated both developers and managers about the pitfalls of automated testing.

Chapter 5.1.2 showed that automated and manual tests are complementary. Manual tests are more suited for exploratory testing, while automated tests are better for regression testing.

In chapter 5.1.3, two methods were presented for test case prioritisation. One for manual testing and one for automated. While both these methods show efficient ways of prioritising test cases. None of them considers the human element. While the algorithms perform as instructed, the tests

are written and in the case of manual testing, executed by humans and humans make errors and inefficient decisions. Consequently, the effectiveness of these algorithms might be less in a real-world application then presented by scientific research in a controlled environment.

7.3 Xray as a test management tool

All participants agreed in **Q4** that Xray would be a sufficient tool for test management. The analysis that is done in chapter 6 supports this but also shed light on some drawbacks such as no encryption offered by Jira, no integrated documentation and no warning that the user's data is not being backed up by default.

7.4 Implementation of the prioritisation algorithm

In chapter 5.2 the implementation of the prioritisation algorithm was tested. The tests show that the proof of concept operates correctly with the Xray REST API. The results in table 5 and figure 7 show that the algorithm correctly considers methods covered by other tests. In table 5 observations can be made that the two tests TestCaseTest and XMLParserTest both contain more test methods than the test JiraClientTest. Yet, the two previously mentioned tests get a lower priority. The tests get a lower priority because the methods they cover are also covered by other tests, as shown in figure 7. TestCaseTest contains two unique methods, but they're small methods such as the constructor and a setter for the priority weight. The size of the uniquely covered methods is another reason for the TestCaseTest having a lower priority than JiraRestClientTest even though TestCaseTest have covered more methods.

7.5 Method discussion

The methods used in this project have successfully been to achieve the goals for the project. There are a few improvements to the methods used that are worth mentioning.

The research study used three scientific databases, as mentioned in chapter 3.2.2. Given more time, this study could have been expanded to several other databases and contributed to the result.

A complete grading of Xray as a test management tool could have been obtained by evaluating all eighth characteristics defined in the ISO/IEC 25010:2011 standard. However, doing such an evaluation would require

access to proprietary information. Even though only five out of the eighth characteristics could be evaluated, the ISO/IEC 25010:2011 standard still indicates the quality of Xray from the perspective of the user. Other methods for evaluation could have been combined with the ISO/IEC 25010:2011 standard to make up for the missing characteristics. But, there where not enough time to research those methods and apply them to Xray.

The survey only had five responses. Unfortunately, it was harder than expected to find companies willing to complete the survey. Given more time, a more extensive survey could have been conducted, providing more reliable results. Another improvement would be to replace the survey with interviews so that the responses of the participants could be improved.

7.6 Social, ethical and scientific aspects

The result has social value because it can be used by companies to show what research says about automated testing, how to implement a conventional test case prioritisation algorithm and make that algorithm work with Xray tests.

An ethical aspect has been to make sure that the survey participants and the company they work with remain anonymous. This anonymity is needed because the survey points out flaws in the reasoning of some of the participants. Thus the companies and the participants are not described in the report.

The project has followed a scientific approach by following a known search process for collecting data, as shown in chapter 3. Thus the result is based on scientific research. The analysis of Xray was conducted based on external analysis of the plugin with criteria from the ISO/IEC 25010:2011 standard mentioned in chapter 3.2.3 and later critically analysed.

7.7 Future work

A follow-up survey asking the same participants similar questions after they've successfully moved to automated testing and gained more experience with the Xray tool could provide new information about Xray and automated testing.

Getting the input for the prioritisation algorithm proved to be complicated. The input that is needed is coverage data from the perspective of a test case. Such as how many statements a specific test covered. Most tools focus on the coverage data from the perspective of the application. That is some method in the application is x% covered by tests. One area

of future work can be to examine different approaches to extracting data from the perspective of test cases.

Another idea for future work can be to analyse how the human element affects how test cases are prioritised and if other factors need to be considered to make test case prioritisation more effective.

References

[1] Dror G Feitelson, Eitan Frachtenberg, and Kent L Beck. "Development and deployment at facebook". In: *IEEE Internet Computing* 17.4 (2013), pp. 8–17.

[2] Sten Pittet. *The different types of software testing*. URL: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing (visited on 2020-03-24).

[3] Deepak Parmar. *Exploratory testing*. URL: https://www.atlassian.com/continuous-delivery/software-testing/exploratory-testing (visited on 2020-04-06).

[4] Jerry Gao, H.-S. J. Tsao, and Ye Wu. *Testing and Quality Assurance for Component-based Software*. Artech House, 2003.

[5] Anirban Basu. *Software Quality Assurance, Testing and Metrics*. Asoke K. Ghosh, PHI Learning Private Limited, 2015.

[6] Gregg Rothermel, Roland H Untch, Chengyun Chu, et al. "Test case prioritization: An empirical study". In: *Proceedings IEEE International Conference on Software Maintenance-1999 (ICSM'99).'Software Maintenance for Business Change'(Cat. No. 99CB36360)*. IEEE. 1999, pp. 179–188.

[7] Atlassian. *A brief overview of Jira*. URL: https://www.atlassian.com/software/jira/guides/getting-started/overview#jira-software-hosting-options (visited on 2020-03-24).

[8] Roy Thomas Fielding. *Representational State Transfer (REST)*. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (visited on 2020-03-27).

[9] Red Hat. *What is an API?* URL: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces (visited on 2020-03-27).

[10] Atlassian. *JIRA Server platform REST API reference*. URL: https://docs.atlassian.com/software/jira/docs/api/REST/8.5.4/ (visited on 2020-03-24).

[11] R. Fielding and J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. URL: https://tools.ietf.org/html/rfc7231 (visited on 2020-04-08).

[12] Bruno Conde. *Xray Cloud Documentation Home*. URL: https://confluence.xpand-it.com/display/XRAYCLOUD/Xray+Cloud+Documentation+Home (visited on 2020-03-24).

[13] Bruno Conde. *REST API*. URL: https://confluence.xpand-it.com/display/XRAYCLOUD/REST+API (visited on 2020-03-27).

[14] Max Rehkopf. *Working with issues in Jira Software*. URL: https://www.atlassian.com/agile/tutorials/issues (visited on 2020-03-24).

[15] Bruno Conde. *Test Process*. URL: https://confluence.xpand-it.com/display/XRAYCLOUD/Test+Process (visited on 2020-04-07).

[16] Bruno Conde. *Pre-Condition*. URL: https://confluence.xpand-it.com/display/XRAY360/Pre-Condition (visited on 2020-04-07).

[17] Bruno Conde. *Test*. URL: https://confluence.xpand-it.com/display/XRAY360/Test (visited on 2020-04-07).

[18] Bruno Conde and Isabel Moreira. *Test Set*. URL: https://confluence.xpand-it.com/display/XRAY35/Test+Set (visited on 2020-04-07).

[19] Bruno Conde and António Melo. *Test Plan*. URL: https://confluence.xpand-it.com/display/public/XRAY/Test+Plan (visited on 2020-04-08).

[20] Bruno Conde and Diamantino Campos. *Execute Tests*. URL: https://confluence.xpand-it.com/display/public/XRAY/Execute+Tests (visited on 2020-04-08).

[21] Cucumber. *What is Cucumber?* URL: https://cucumber.io/docs/guides/overview/#what-are-step-definitions (visited on 2020-03-27).

[22] Cucumber. *Gherkin Reference*. URL: https://cucumber.io/docs/gherkin/reference/#example (visited on 2020-03-27).

[23] Cucumber. *Gherkin Reference*. URL: https://cucumber.io/docs/gherkin/reference/#steps (visited on 2020-03-27).

[24] JUnit. *Frequently Asked Questions*. URL: https://junit.org/junit4/faq.html#faqinfo_1 (visited on 2020-03-24).

[25] OpenClover. *About Code Coverage*. URL: https://openclover.org/doc/manual/latest/general--about-code-coverage.html (visited on 2020-04-14).

[26] OpenClover. *OpenClover*. URL: https://openclover.org/index (visited on 2020-04-14).

[27] OpenClover. *Configuring reports*. URL: https://openclover.org/doc/manual/4.2.0/maven--configuring-reports.html (visited on 2020-04-14).

[28] OpenClover. *Features*. URL: https://openclover.org/features (visited on 2020-04-14).

[29] Postman. *The Postman API Platform*. URL: https://www.postman.com/api-platform (visited on 2020-03-31).

[30] Wojtek Seliga. *JIRA REST Java Client Library*. URL: https://ecosystem.atlassian.net/wiki/spaces/JRJC/overview?homepageId=27164679 (visited on 2020-04-27).

[31] iso. *STANDARDS*. URL: https://www.iso.org/standards.html (visited on 2020-04-27).

[32] International Standard ISO: ISO/IEC 25010–2011. "Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models". In: 1 (2011).

[33] Atlassian. *Confluence*. URL: https://www.atlassian.com/software/confluence (visited on 2020-04-27).

[34] Google. *Lighthouse*. URL: https://developers.google.com/web/tools/lighthouse (visited on 2020-04-30).

[35] Atlassian. *OAuth*. URL: https://developer.atlassian.com/server/jira/platform/oauth/ (visited on 2020-04-30).

[36] The Apache Software Foundation. *Welcome to Apache Maven*. URL: https://maven.apache.org/index.html (visited on 2020-03-31).

[37] The Apache Software Foundation. *Introduction to the POM*. URL: https://maven.apache.org/guides/introduction/introduction-to-the-pom.html (visited on 2020-03-31).

[38] The Apache Software Foundation. *What is Maven?* URL: https://maven.apache.org/what-is-maven.html (visited on 2020-03-31).

[39] The Apache Software Foundation. *Introduction to Build Profiles*. URL: https://maven.apache.org/guides/introduction/introduction-to-profiles.html (visited on 2020-04-15).

[40] The Apache Software Foundation. *Maven Surefire Plugin*. URL: https://maven.apache.org/surefire/maven-surefire-plugin/ (visited on 2020-04-15).

[41] Max Rehkopf. *What is Continuous Integration*. URL: https://www.atlassian.com/continuous-delivery/continuous-integration (visited on 2020-03-31).

[42] Johannes Gmeiner, Rudolf Ramler, and Julian Haslinger. "Automated testing in the continuous delivery pipeline: A case study of an online company". In: *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE. 2015, pp. 1–6.

[43] Kristina Alexanderson. *Källkritik på nätet*. URL: https://internetstiftelsen.se/guide/kallkritik-pa-internet/kallkritik-pa-natet/ (visited on 2020-03-27).

[44] Rainer Nyberg and Annika Tidström. *Skriv vetenskapliga uppsater, examensarbeten och avhandlingar*. Studentlitteratur AB, 2012.

[45] jr Floyd J. Fowler. *Survey Research Methods*. SAGE publication Inc, 2014.

[46] Bogdan Vasilescu, Yue Yu, Huaimin Wang, et al. "Quality and productivity outcomes relating to continuous integration in GitHub". In: 2015-08, pp. 805–816. DOI: 10.1145/2786805.2786850.

[47] ScottWambler. *Introduction to Test Driven Development (TDD)*. URL: http://agiledata.org/essays/tdd.html (visited on 2020-04-02).

[48] SmartBear. *Behaviour-Driven Development*. URL: https://cucumber.io/docs/bdd/ (visited on 2020-04-02).

[49] Divya Kumar and KK Mishra. "The Impacts of Test Automation on Software's Cost, Quality and Time to Market". In: *Procedia Computer Science* 79 (2016), pp. 8–15.

[50] Dudekula Mohammad Rafi, Katam Reddy Kiran Moses, Kai Petersen, et al. "Benefits and limitations of automated software testing: Systematic literature review and practitioner survey". In: *2012 7th International Workshop on Automation of Software Test (AST)*. IEEE. 2012, pp. 36–42.

[51] Kristian Wiklund, Daniel Sundmark, Sigrid Eldh, et al. "Impediments for automated testing–an empirical analysis of a user support discussion board". In: *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*. IEEE. 2014, pp. 113–122.

[52] Emil Alegroth, Robert Feldt, and Helena H Olsson. "Transitioning manual system test suites to automated testing: An industrial case study". In: *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*. IEEE. 2013, pp. 56–65.

[53] Ossi Taipale, Jussi Kasurinen, Katja Karhu, et al. "Trade-off between automated and manual software testing". In: *International Journal of System Assurance Engineering and Management* 2.2 (2011), pp. 114–125.

[54] Rudolf Ramler and Klaus Wolfmaier. "Economic perspectives in test automation: balancing automated and manual testing with opportunity cost". In: *Proceedings of the 2006 international workshop on Automation of software test*. 2006, pp. 85–91.

[55] Hadi Hemmati, Zhihan Fang, and Mika V Mantyla. "Prioritizing manual test cases in traditional and rapid release environments". In: *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE. 2015, pp. 1–10.

[56] Emelie Engström, Per Runeson, and Andreas Ljung. "Improving Regression Testing Transparency and Efficiency with History-Based Prioritization–An Industrial Case Study". In: *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*. IEEE. 2011, pp. 367–376.

[57] Sebastian Elbaum, Alexey G Malishevsky, and Gregg Rothermel. "Test case prioritization: A family of empirical studies". In: *IEEE transactions on software engineering* 28.2 (2002), pp. 159–182.

[58] Bo Jiang, Zhenyu Zhang, Wing Kwong Chan, et al. "Adaptive random test case prioritization". In: *2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE. 2009, pp. 233–244.

[59] Dan Hao, Lu Zhang, Lei Zang, et al. "To be optimal or not in test-case prioritization". In: *IEEE Transactions on Software Engineering* 42.5 (2015), pp. 490–505.

[60] Qi Luo, Kevin Moran, and Denys Poshyvanyk. "A large-scale empirical comparison of static and dynamic test case prioritization techniques". In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 2016, pp. 559–570.

[61] Lingming Zhang, Dan Hao, Lu Zhang, et al. "Bridging the gap between the total and additional test-case prioritization strategies". In: *2013 35th International Conference on Software Engineering (ICSE)*. IEEE. 2013, pp. 192–201.

[62] Dan Hao, Lingming Zhang, Lu Zhang, et al. "A unified test case prioritization approach". In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 24.2 (2014), pp. 1–31.

[63] Andy Heinzer. *Encryption options*. URL: `https://community.atlassian.com/t5/Jira-Core-questions/Encryption-options/qaq-p/661897` (visited on 2020-04-30).

[64] Adaptavist. *Encryption for Jira*. URL: `https://docs.adaptavist.com/encryption/jira/server/latest/` (visited on 2020-04-30).

A Appendix A

The pom.xml that used in 4.1

A.1 POM file for Maven integration

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         ↪ http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.example</groupId>
    <artifactId>MavenIntegrationTest</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
        <junit.version>4.13</junit.version>
        <xray.projectKey>MAV</xray.projectKey>
        <xray.testEnvironments>Java</xray.testEnvironments>
        <xray.revision>Build-1.0</xray.revision>
        <xray.resultsFormat>JUNIT</xray.resultsFormat>

        ↪ <xray.surefire.location>${basedir}/target/surefire-reports</xray.surefire.location
    </properties>

    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-surefire-plugin</artifactId>
                    <version>2.19.1</version>
                    <configuration>
                        <testFailureIgnore>true</testFailureIgnore>
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>

```

```
    <reporting>
        <plugins>
            <plugin>
                <artifactId>maven-surefire-report-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>com.xpandit.xray</groupId>
                <artifactId>xray-maven-plugin</artifactId>
                <version>1.1.0</version>
            </plugin>
        </plugins>
    </reporting>

    <pluginRepositories>
        <pluginRepository>
            <id>xpand-plugins</id>
            <name>xpand-plugins</name>
            <url>http://maven.xpand-it.com/artifactory/releases</url>
        </pluginRepository>
    </pluginRepositories>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

</project>
```

A.2 JSON file for Maven integration

```
{
  "fields": {
    "project": {
      "key": "MAV"
    },
    "summary": "Needs to be done",
    "description": "This is a high priority test execution, now updated",
    "priority":{
      "name": "High"
    },
```

```
11      "assignee":{
12        "name": "Siber92"
13      },
14      "issuetype": {
15        "name": "Test Execution"
16      }
17    }
18  }
```

B Appendix B

B.1 Implementation of the Jira REST client

```
1  import static utility.IssuePriority.*;
2  public class JiraClient {
3      private final String username;
4      private final String password;
5      private final String URL;
6      private final JiraRestClient restClient;
7      public JiraClient(String username, String password, String URL){
8          this.username = username;
9          this.password = password;
10         this.URL = URL;
11         this.restClient = getJiraRestClient();
12     }
13
14     private URI getJiraUri() {
15         return URI.create(this.URL);
16     }
17
18     /**
19      *
20      * @return A Jira rest client that can handle Jira requests.
21      */
22     private JiraRestClient getJiraRestClient() {
23         return new AsynchronousJiraRestClientFactory()
24
                 ↪ .createWithBasicHttpAuthentication(getJiraUri(),username,password);
25     }
26
27     /**
28      * Updates the priority of a issue
29      * @param issueKey key to identify the issue to update.
30      */
31     public String updatePriority(String issueKey, Long priority){
32         try {
33             IssueInput input = new IssueInputBuilder()
34                     .setPriorityId(priority)
35                     .build();
36             restClient.getIssueClient()
37                     .updateIssue(issueKey, input)
38                     .claim();
```

```
39          }catch(RestClientException e){
40              if(e.getStatusCode().get().intValue() == 401){
41                  System.out.println(username + "  " + password + "    " +
                    ↪  URL);
42                  return "error";
43              }
44              else{
45                  System.out.println(e);
46              }
47          }
48
49          return "issue with key: " + issueKey + " was updated";
50      }
51
52      /**
53       * Maps the priority weight from the prioritisation algoirthm to an
    ↪  Xray priority.
54       * @param prioWeightMax The priority weight of the test with the
    ↪  highest priority weight.
55       * @param testPrioWeight The priority weight of the current test.
56       * @return an Xray priority
57       */
58
59      public static Long getPriroityMapping(double prioWeightMax, double
         ↪  testPrioWeight){
60          int nrOfpriorities = 5;
61          double prioWeightMin = prioWeightMax / nrOfpriorities;
62          if(testPrioWeight <= prioWeightMin){
63              return LOWEST.toLong();
64          }
65          else if(testPrioWeight > prioWeightMin && testPrioWeight <=
             ↪  prioWeightMin*2){
66              return LOW.toLong();
67          }
68          else if(testPrioWeight > 2*prioWeightMin && testPrioWeight <=
             ↪  prioWeightMin*3){
69              return MEDIUM.toLong();
70          }
71          else if(testPrioWeight > 3*prioWeightMin && testPrioWeight <=
             ↪  prioWeightMin*4){
72              return HIGH.toLong();
73          }
74          else if(testPrioWeight > 4*prioWeightMin && testPrioWeight <=
             ↪  prioWeightMax){
```

```
75              return HIGHEST.toLong();
76          }
77          return -1L;
78      }
```

B.2 Implementation of the prioritisation algorithm

```
1   public class PrioritisationAlgorithm {
2       private final Vector<Double> probability; //contains the probability
        ↪  of selecting units.
3       private final Vector<Boolean> selected; //if a test case has been
        ↪  selected its index will be true.
4       private final Vector<TestCase> priority; // to stores the prioritized
        ↪  test cases
5       private final Vector<TestCase> input; //contains unprioritized test
        ↪  cases.
6       private final Vector<String> methods;
7
8       private final int nrOfUnits;
9       private final int nrOfTestCases;
10      private final int metricMin;
11      private final int metricMax;
12      private int testCaseIndex;
13      private int unit;
14
15      private double f_p;
16      private double sum;
17
18
19
20      public PrioritisationAlgorithm(ProjectData projectData,
        ↪  Vector<TestCase> input) {
21          probability = new Vector<>();
22          selected = new Vector<>();
23          priority = new Vector<>();
24          this.methods = projectData.projectMethods;
25          this.nrOfUnits = methods.size();
26          this.metricMin = projectData.metricMin;
27          this.metricMax = projectData.metricMax;
28          this.input = input;
29          this.f_p = 1;
30          nrOfTestCases = input.size();
```

```
31
32          for (int test = 0; test < nrOfTestCases; test++) {
33              selected.add(false);
34          }
35          for (int unit = 0; unit < nrOfUnits; unit++) {
36              probability.add(1.0);
37          }
38      }
39
40      /**
41       * Normalise the f_p to be in the range lowEnd <= highEnd
42       * @param testCaseMetric the metric used,
43       * in this example the number of statements a method   covers is
    ↪  used.
44       * @return a normalised f_p
45       */
46      private double normalizeF_p(int testCaseMetric){
47          double numerator = testCaseMetric - metricMin;
48          double denominator = metricMax - metricMin;
49          if(denominator == 0){
50              denominator = 1;
51          }
52          double lowEnd = 0.6; //recommended range from the study is 0.65
            ↪  <= f_p <= 0.95
53          double highEnd = 0.95;
54          return 1 - ((highEnd-lowEnd) * (numerator / denominator));
55      }
56
57      /**
58       * Main loop for test case prioritization.
59       * @return a prioritised vector.
60       */
61      public Vector<TestCase> prioritize() {
62          for (int testCase = 0; testCase < nrOfTestCases; testCase++) {
63              findUnchosenTestCase();
64              checkForBetterFit();
65              updatePriority();
66          }
67          return priority;
68      }
69
70      /**
71       * Finds a previously not chosen test case and calculates the
    ↪  probability that the units covered by
```

```
72          * that test case contains undetected faults.
73          */
74         private void findUnchosenTestCase() {
75             testCaseIndex = 0;
76             while (selected.elementAt(testCaseIndex)) {
77                 testCaseIndex++;
78             }
79             sum = 0;
80             TestCase current = input.get(testCaseIndex);
81             for (unit = 0; unit < nrOfUnits; unit++) {
82                 String currentUnit = methods.elementAt(unit);
83                 int nrOfOccurrences = current.occurrence(currentUnit);
84                 if (nrOfOccurrences != 0) {
85                     f_p =
                        ↪  normalizeF_p(current.getUnitStatementCoverage(currentUnit));
86                     sum += probability.elementAt(unit) *
                        ↪  (1-(Math.pow((1-f_p),nrOfOccurrences)));
87                 }
88             }
89         }
90
91         /**
92          * Checks the remaining test cases if there exist a test case with
    ↪  higher probability, then the one first chosen
93          * to contain undetected faults.
94          */
95         private void checkForBetterFit() {
96             for (int otherTestCase = testCaseIndex + 1; otherTestCase <
                ↪  nrOfTestCases; otherTestCase++) {
97
98                 TestCase current = input.get(otherTestCase);
99                 if (!selected.elementAt(otherTestCase)) {
100                     double newSum = 0;
101                     for (unit = 0; unit < nrOfUnits; unit++) {
102                         String currentUnit = methods.elementAt(unit);
103                         int nrOfOccurrences =
                            ↪  current.occurrence(currentUnit);
104                         if (nrOfOccurrences != 0){
105                             f_p =
                                ↪  normalizeF_p(current.getUnitStatementCoverage(currentUnit));
106                             newSum += probability.elementAt(unit) *
                                ↪  (1-(Math.pow((1-f_p),nrOfOccurrences)));
                                ↪  //changes
107                         }
```

```
108                 }
109
110                 if (newSum > sum) {
111                     sum = newSum;
112                     testCaseIndex = otherTestCase;
113                 }
114             }
115         }
116     }
117
118     /**
119      * Updates the priority vector and sets new probabilities for already
    ↪  covered units
120      *
121      */
122     private void updatePriority() {
123         TestCase current = input.get(testCaseIndex);
124         current.setPriorityWeight(sum);
125         priority.add(current);
126         selected.set(testCaseIndex, true);
127         for (unit = 0; unit < nrOfUnits; unit++) {
128             String currentUnit = methods.elementAt(unit);
129             int nrOfOccurrences = current.occurrence(currentUnit);
130             if (nrOfOccurrences != 0){
131                 f_p =
                    ↪  normalizeF_p(current.getUnitStatementCoverage(currentUnit));
132                 double prob = probability.get(unit) * Math.pow((1 -
                    ↪  f_p),nrOfOccurrences);
133                 probability.set(unit, prob);
134             }
135         }
136     }
137 }
```

B.3 Implementation of the test case class

```
1  /**
2   * A class that represents a test case. A test case contains methods
    ↪  stored in the units map.
3   * The pair stores occurrences of a method and how many statements the
    ↪  method covers.
4   */
5  public class TestCase {
```

```
6       private final String name;
7       private final Map<String,Pair<Integer, Integer>> units; //units and
        ↪  there occurence
8       private double priorityWeight;
9
10
11      public TestCase(String name, Map<String, Pair<Integer, Integer>>
        ↪  units){
12          this.name = name;
13          this.units = units;
14          this.priorityWeight = 0;
15      }
16
17      public void testCasePrinter(){
18          System.out.println("----- test cast printer -----");
19          System.out.println("name: " + name);
20          System.out.println("coverageSum: " + priorityWeight);
21          units.forEach((key,value) -> System.out.println(key + "
            ↪  OCCOURENCE: " +  value.getFirstValue()
22          + "METHOD COVERAGE: " + value.getSecondValue()));
23      }
24
25      /**
26       *
27       * @param unit unit to check.
28       * @return occurrences for the given unit
29       */
30      public int occurrence(String unit){
31          if(units.get(unit) != null){
32              Pair<Integer, Integer> temp = units.get(unit);
33              return temp.getFirstValue();
34          }
35          return 0;
36      }
37      /**
38       *
39       * @param unit
40       * @return the statement coverage for the given unit.
41       */
42      public int getUnitStatementCoverage(String unit){
43          if(units.get(unit) != null){
44              Pair<Integer, Integer> temp = units.get(unit);
45              return temp.getSecondValue();
46          }
```

```
47          return 0;
48      }
49      public String getName() {
50          return name;
51      }
52      public void setPriorityWeight(double priorityWeight){
53          this.priorityWeight = priorityWeight;
54      }
55      public double getPriorityWeight(){
56          return priorityWeight;
57      }
58  }
```