The logo of TAMK UAS. 

Software Testing  
Basic Technology and Practical Application

O R Imon

Software Implementation and Testing

March 2023

Bachelor’s Degree Programme in Software Engineering

CONTENTS

[1 Abstract 4](#_Toc38544594)

[2 Overview of Software Testing 5](#_Toc38544595)

[2.1 Types of Software Testing 5](#_Toc38544596)

[2.1.1 Approaches to Software Testing 5](#_Toc38544597)

[2.1.2 Techniques Used in Software Testing 5](#_Toc38544598)

[2.2 Challenges in Software Testing 5](#_Toc38544599)

[2.2.1 Latest Trends in Software Testing 5](#_Toc38544600)

[2.2.2 Unit Testing 6](#_Toc38544601)

[2.2.3 Functional Testing 6](#_Toc38544602)

[2.2.4 Integration Testing 6](#_Toc38544603)

[3 System Testing 6](#_Toc38544604)

[3.1 Acceptance Testing 6](#_Toc38544605)

[3.1.1 Principles of Testing 6](#_Toc38544606)

[3.1.2 Exhaustive testing is impossible 6](#_Toc38544607)

[3.1.3 Early testing saves time and money 6](#_Toc38544608)

[3.2 Early testing saves time and money 6](#_Toc38544609)

[3.2.1 Defect clustering 6](#_Toc38544610)

[3.2.2 Pesticide paradox 6](#_Toc38544611)

[4 Case Study 7](#_Toc38544612)

[4.1 Write a test case 7](#_Toc38544613)

[4.1.1 Frameworks 7](#_Toc38544614)

[4.1.2 Test-Driven Development (TDD) 7](#_Toc38544615)

[4.1.3 Write code, Run, Test the code, Refactor, Repat 9](#_Toc38544616)

[4.2 Unit Test 10](#_Toc38544617)

[4.2.1 Examples, Test Case, code 11](#_Toc38544618)

[4.2.2 Write code, Run, Test the code, Refactor](#_Toc38544619) 12

[5 DISCUSSION 12](#_Toc38544620)

[REFERENCES 14](#_Toc38544621)

**GLOSSARY or ABBREVIATIONS AND TERMS (choose one or other)**

TAMK Tampere University of Applied Sciences

TDD Test driven development.

# 1 Abstract

Software testing is a critical process in software development that ensures the software is working as expected and meets the desired quality standards. The purpose of this report is to provide a comprehensive study of software testing, including the different approaches and techniques used in testing, the challenges associated with testing, and the latest trends in the field.

The first section of this report provides an overview of software testing, including its definition, objectives, and benefits. The second section covers the different types of software testing, including functional, non-functional, and performance testing. The third section explores the different approaches to software testing, such as manual and automated testing, and discusses the advantages and disadvantages of each approach.

The fourth section of this report delves into the different techniques used in software testing, including black-box testing, white-box testing, and grey-box testing. The fifth section discusses the challenges associated with software testing, including the cost of testing, the time required for testing, and the difficulty of testing complex systems.

The sixth section of this report covers the latest trends in software testing, such as the use of artificial intelligence and machine learning in testing, the integration of testing into the DevOps process, and the use of testing metrics to improve the efficiency and effectiveness of testing.

The final section of this report provides a case study of software testing in a real-world scenario, demonstrating how the different approaches, techniques, and challenges discussed in this report can be applied in practice.

Overall, this thesis report provides a comprehensive study of software testing, including the different approaches and techniques used in testing, the challenges associated with testing, and the latest trends in the field. It serves as a valuable resource for software developers, testers, and quality assurance professionals.

Keywords: software testing, approaches, techniques, challenges, trends, artificial intelligence, DevOps, metrics.

# 2 Overview of Software Testing

Software testing is the process of evaluating the software's behaviour against its expected behaviour. The main objectives of software testing are to identify defects, errors, and failures in the software and ensure that the software meets the desired quality standards. Software testing helps to improve the software's reliability, maintainability, and performance.

2.1 Types of Software Testing:

There are different types of software testing, including functional testing, non-functional testing, and performance testing. Functional testing ensures that the software meets the functional requirements specified in the software specification. Non-functional testing checks the software's non-functional aspects, such as usability, reliability, and security. Performance testing evaluates the software's performance under different load conditions.

2.1.1 Approaches to Software Testing:

There are different approaches to software testing, including manual testing, automated testing, and a combination of both. Manual testing involves human testers executing test cases on the software manually. Automated testing involves using software tools to execute test cases automatically. The choice of approach depends on the software's complexity, time, and cost constraints.

2.1.2 Techniques Used in Software Testing:

There are different techniques used in software testing, including black-box testing, white-box testing, and grey-box testing. Black-box testing involves testing the software's functionality without knowing its internal structure. White-box testing involves testing the software's internal structure and code. Grey-box testing is a combination of black-box and white-box testing.

2.2 Challenges in Software Testing:

There are different challenges associated with software testing, including the cost of testing, the time required for testing, and the difficulty of testing complex systems. Testing can be expensive, especially for complex systems. Testing also takes time, which can delay the software's release. Testing complex systems can be challenging because of their complexity.

2.2.1 Latest Trends in Software Testing:

There are different latest trends in software testing, including the use of artificial intelligence and machine learning in testing, the integration of testing into the DevOps process, and the use of testing metrics to improve the efficiency and effectiveness of testing. Artificial intelligence and machine learning can help automate testing tasks and improve the testing process's accuracy. DevOps is an approach that integrates software development and operations to improve the software's quality and speed up its release. Testing metrics help measure the efficiency and effectiveness of testing.

There are different types of testing, including unit testing, functional testing, integration testing, system testing, and acceptance testing.

2.2.2 Unit Testing: Unit testing is a testing method that focuses on testing individual components or modules of the software. It is the smallest and most basic type of testing and helps to ensure that each component of the software is functioning correctly.

2.2.3 Functional Testing: Functional testing is a type of testing that focuses on testing the software's functionality. It ensures that the software is working according to the functional requirements specified in the software specification.

2.2.4 Integration Testing: Integration testing is a type of testing that focuses on testing the integration between different modules or components of the software. It ensures that the different components are working together correctly and communicating with each other as expected.

3. System Testing: System testing is a type of testing that focuses on testing the entire software system. It ensures that the software is working correctly in different environments and under different conditions.

3.1 Acceptance Testing: Acceptance testing is a type of testing that focuses on testing the software's acceptance by the end-users. It ensures that the software meets the user's requirements and expectations.

3.1.1 Principles of Testing:

There are different principles of testing, including:

Testing shows the presence of defects: Testing cannot prove that software is defect-free, but it can identify the presence of defects.

3.1.2 Exhaustive testing is impossible: It is impossible to test all possible combinations of inputs, outputs, and conditions.

3.1.3 Early testing saves time and money: It is more cost-effective to find and fix defects early in the development lifecycle than later.

3.2.1 Defect clustering: Defects tend to cluster together in certain areas of the software.

3.2.2 Pesticide paradox: Repeating the same tests over and over will eventually stop finding new defects.

4 Case Study

The final section of this report provides a case study of software testing in a real-world scenario, demonstrating how the different approaches, techniques, and challenges discussed in this report can be applied in practice. The case study focuses on the testing of a web-based e-commerce application and demonstrates the use of different testing approaches, techniques, and tools to ensure the software's quality.

How to Apply Testing to Software:

One approach to applying testing to software is Test-Driven Development (TDD). TDD is a software development process that emphasizes writing tests before writing the code. The process involves the following steps:

4.1 Write a test case: Write a test case for a specific piece of functionality that the software should implement.

Run the test case: Run the test case and ensure that it fails.

Write the code: Write the code to implement the functionality.

Run the test case: Run the test case again and ensure that it passes.

Refactor the code: Refactor the code to ensure that it is maintainable and efficient.

Repeat the process: Repeat the process for the next piece of functionality.

4.1.1 Frameworks

There are various tools and frameworks available for software testing, and it's important to choose the appropriate tool for the specific testing needs. Some of the commonly used testing frameworks include JUnit for Java, NUnit for .NET, PyTest for Python, and Jasmine for JavaScript.

3.1.3 Test-Driven Development (TDD)

One of the most popular approaches to software testing is Test-Driven Development (TDD), which we discussed earlier. TDD emphasizes writing tests before writing the code and helps to ensure that the software meets the desired quality standards. TDD can also lead to more efficient and maintainable code since the tests act as a form of documentation and help ensure that changes to the code do not break existing functionality.

In addition to TDD, there are other testing methodologies and techniques, such as Behaviour-Driven Development (BDD), Acceptance Test-Driven Development (ATDD), and Exploratory Testing. Each of these approaches has its unique benefits and can be applied based on the specific testing needs.

To write unit tests using a software language framework in the spirit of Test-Driven Development (TDD), the following steps can be followed:

Define the requirements: First, define the requirements for the software module that needs to be tested. This will help to identify the test cases that need to be written.

Write the test case: Next, write a test case that covers the requirement. The test case should be written in the form of a unit test, which means it should test a single piece of functionality in the software module.

Run the test: Once the test case is written, run it to ensure that it fails. This is an important step in TDD because it ensures that the test case is testing the functionality that it is supposed to test.

Write the code: After the test case has been written and has failed, write the code that will make the test case pass. It's important to write only enough code to make the test case pass, and not to add any extra functionality at this stage.

Run the test again: After the code has been written, run the test again to ensure that it passes. If it does not pass, continue to modify the code until the test case passes.

Refactor: Once the test case has passed, it's time to refactor the code to make it more maintainable and efficient. Refactoring is an important step in TDD because it ensures that the code is of high quality and can be easily maintained in the future.

Repeat: Repeat the above steps for each additional requirement and test case until all of the requirements have been met and all of the test cases pass.

Some popular software language frameworks for unit testing in TDD include JUnit for Java, NUnit for .NET, PyTest for Python, and Jasmine for JavaScript. These frameworks provide a set of tools and features that make it easier to write and run unit tests, and they can also generate reports that provide information on test coverage and performance.

writing unit tests using a software language framework in the spirit of TDD involves defining requirements, writing test cases, running tests, writing code, running tests again, refactoring, and repeating for each additional requirement and test case. This process helps to ensure that the software meets the desired quality standards and functions correctly.

To further understand the process of writing unit tests using TDD, let's take a simple example of a function that calculates the sum of two numbers. We can follow the steps outlined above to write unit tests using JUnit for Java in the spirit of TDD:

Define the requirements: The function should take two integers as input and return their sum.

Write the test case: Write a test case that tests the function with two input values and checks that the output is equal to the sum of the input values.

For example:

java

Copy code

@Test

public void testSum() {

int result = MyClass.sum(3, 4);

assertEquals(7, result);

}

Run the test: Run the test to ensure that it fails.

Write the code: Write the code for the sum() function that will make the test pass. For example:

java

Copy code

public class MyClass {

public static int sum(int a, int b) {

return a + b;

}

}

Run the test again: Run the test again to ensure that it passes.

Refactor: Refactor the code to make it more maintainable and efficient. For example, we could rename the MyClass to something more descriptive.

Repeat: Repeat the above steps for additional test cases and requirements until all of the requirements have been met and all of the test cases pass.

By following this process, we can ensure that our software meets the desired quality standards and functions correctly. Writing unit tests using a software language framework in the spirit of TDD helps to identify and fix defects early in the development process, which can save time and effort in the long run.

In addition to writing unit tests using TDD, it's also important to understand other types of testing and how they fit into the software development life cycle.

3.1.4 Unit Test

software testing is an essential part of the software development process. By writing unit tests using a software language framework in the spirit of TDD, developers can catch defects early on and ensure that their code meets the desired quality standards. Additionally, understanding the different types of testing and how they fit into the software development life cycle can help to ensure that the software is of high quality and meets the needs of the end-users or stakeholders.

I can give you an example of how to design and implement a program with unit testing using a programming language and its framework. Let's take Java and JUnit as an example.

First, we need to define the requirements of our program. Let's say we want to create a simple calculator program that can perform addition, subtraction, multiplication, and division operations.

Next, we need to create a Java project and add the JUnit dependency to our project. We can do this by adding the following dependency to our Maven pom.xml file:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

Once we have added the JUnit dependency, we can create a class for our calculator program. Let's call it Calculator.

public class Calculator {

public static int add(int a, int b) {

return a + b;

}

public static int subtract(int a, int b) {

return a - b;

}

public static int multiply(int a, int b) {

return a \* b;

}

public static int divide(int a, int b) {

return a / b;

}

}

Now that we have implemented our calculator program, we can write unit tests for each of the methods using JUnit. Here's an example test case for the add() method:

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class CalculatorTest {

@Test

public void testAdd() {

int result = Calculator.add(2, 3);

assertEquals(5, result);

}

}

In this test case, we are testing the add() method with input values of 2 and 3, and checking that the output is equal to 5 using the assertEquals() method from JUnit.

We can write similar test cases for the subtract(), multiply(), and divide() methods.

Once we have written our test cases, we can run them using the JUnit test runner. If all of the test’s pass, we can be confident that our program works as expected.

In this designing and implementing a program with unit testing using a programming language and its framework involves defining the requirements, creating the program, and writing test cases using the testing framework. By following this process, we can ensure that our program works correctly and meets the desired quality standards.

After writing our test cases and running them using the JUnit test runner, we may encounter failures or errors. This is a normal part of the testing process, and it indicates that there is an issue with our code. We need to identify the root cause of the failure or error and fix it in our code. Once we fix the issue, we can re-run the test cases to ensure that the issue has been resolved.

In addition to writing test cases for individual methods, we can also write test cases for the overall functionality of our program. For example, we can create test cases for different combinations of inputs to ensure that our program works correctly in all possible scenarios. This is known as integration testing.

Another important aspect of unit testing is code coverage. Code coverage measures how much of our code is covered by our test cases. We want to ensure that our test cases cover as much of our code as possible to catch any potential issues. We can use tools such as JaCoCo or IntelliJ IDEA's built-in code coverage tool to measure code coverage.

In summary, unit testing is a crucial part of software development as it helps us to catch defects early on and ensure that our code meets the desired quality standards. By using a programming language and its testing framework, we can write test cases for our code and ensure that it works correctly. Additionally, by writing test cases for overall functionality and measuring code coverage, we can increase the effectiveness of our testing and improve the overall quality of our code.

5 DISCUSSIONS:

Software testing is an essential part of the software development process that involves executing a program or system to identify any defects or errors that may impact its functionality or performance. The objective of software testing is to ensure that the software meets the specified requirements and is free of defects or bugs that could cause problems for end-users.

There are different types of software testing, such as unit testing, integration testing, system testing, acceptance testing, and regression testing. Each type of testing serves a different purpose and is performed at different stages of the software development life cycle.

Software testing has several benefits. It helps to identify and fix defects early in the development process, which reduces the cost and effort required to fix them later. It also helps to ensure that the software meets the specified requirements and is reliable, efficient, and user-friendly.

However, software testing also has some limitations. It cannot guarantee that the software is completely free of defects or that it will perform flawlessly in all situations. Testing can only identify the defects that have been explicitly defined and tested for, and it cannot anticipate all the potential issues that could arise in the real world.

In conclusion, software testing is an essential process that helps to ensure the quality and reliability of software. It is a continuous process that should be carried out at different stages of the software development life cycle to identify and fix defects early. While software testing cannot guarantee perfect software, it can significantly reduce the likelihood of defects and improve the user experience.

REFERENCES

Lewis, W. 2005. Software Testing and Continuous Quality Improvement. 2nd edition. Boca Raton: Auerbach Publications

Pittet, S. 2023. “The different types of software testing”. Atlassian.

Read on 20.03.2023. Available at:

https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing