OULUN AMMATTIKORKEAKOULU

Jere Karvonen

Enhancing Software Quality: A Comprehensive Study of Modern Software Testing Methods

A study on the methods used to design and execute modern software testing and their impact on software projects.

Enhancing Software Quality: A Comprehensive Study of Modern Software Testing Methods

A study on the methods used to design and execute modern software testing and their impact on software projects.

Jere Karvonen
Bachelor’s Thesis
Spring 2024
Bachelor of Engineering Information Technology
Oulu University of Applied Science

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Bachelor of Engineering, Information Technology

Tekijä(t): Jere Karvonen
Opinnäytetyön nimi: Enchancing Software Quality: A Comprehensive Study of Modern Software Testing Methods
Työn ohjaaja(t): Olli Himanka
Työn valmistumislukukausi ja -vuosi: Kevät 2024 Sivumäärä: 36

Tämä opinnäytetyö pyrkii tarjoamaan kattavan yleiskuvan nykyaikaisista ohjelmistotestausmenetelmistä, työn tavoitteena on antaa lukijalle parempi ymmärrys näiden menetelmien hyödyistä sekä haitoista.

Alkaen kirjallisuuskatsauksella, ohjelmistotestauksen kehitystä sen alkuajoista 1940-luvun puolivälistä nykyaikaisiin lähestymistapoihin tarkastellaan, korostaen siirtymistä perinteisistä testausmenetelmistä nykyaikaisiin lähestymistapoihin.

Lisäksi tämä opinnäytetyö pyrkii tuomaan esille nykyaikaisten ohjelmistotestausmenetelmien yleisiä ominaisuuksia, kuten vahvaa riippuvuutta testiautomaatiosta, shift-left- ja shift-right-testaus, sekä jatkuva- ja tutkivatestaus, antamalla perusteellisen erittelyn jokaisesta näistä menetelmästä.

Perinteisten ja nykyaikaisten testauslähestymistapojen vertaileva analyysi suoritetaan viiden keskeisen kriteerin perusteella: luotettavuus, skaalautuvuus, ylläpidettävyys, tehokkuus ja sopeutumiskyky. Tämän analyysin avulla molempien metodologioiden edut sekä haitat tuodaan esiin, mikä auttaa lukijaa ymmärtämään paremmin, mikä menetelmä sopii parhaiten heidän tarpeisiinsa.

Nykyaikaisten testauskäytäntöjen omaksumisen haasteet, kuten tarvittava kulttuurimuutos, taitovaatimusten kasvu ja infrastruktuurivaatimusten muutokset, tunnistetaan ja käydään läpi. Lisäksi selvitetään ohjelmistotestauksen tulevaisuuden suuntaukset sekä uudet trendit, kuten tekoälyn käyttö, kooditon automaatio ja koneoppimisen hyödyntäminen.

Tämä opinnäytetyö korostaa nykyaikaisten ohjelmistotestausmenetelmien mahdollisia etuja sekä haittoja ohjelmiston kokonaislaadun ja luotettavuuden parantamisessa. Sen tarkoituksena on toimia oppaana alan ammattilaisille päätöksenteossa siitä, mikä testausmenetelmä sopii parhaiten heidän tarpeisiinsa.

Asiasanat: Ohjelmistotestaus, Manuaalinen testaus, Automaattinen testaus, Ohjelmistokehityksenelinkaari

ABSTRACT

Oulu University of Applied Sciences
Bachelor of Engineering, Information Technology

Author(s): Jere Karvonen
Title of thesis:
Enhancing Software Quality: A Comprehensive Study of Modern Software Testing Methods
Supervisor(s): Olli Himanka
Term and year when the thesis was submitted: Spring 2024
Number of pages: 36

This thesis aims to provide a comprehensive overview of modern software testing methods, with the end goal of providing the reader with a better understanding of the advantages and disadvantages of utilizing these methods.

Starting with a literature review, the evolution of software testing from its beginnings in the mid-1940's to the modern-day approaches in use today is analyzed, highlighting the gradual shift from traditional testing methods to more modern approaches.

Additionally, this thesis aims to underline the common characteristics of modern software testing approaches such as the heavy reliance on test automation, the use of shift-left and shift-right testing, and the practices of continuous, and exploratory testing, by giving a thorough breakdown on each of these methods.

Comparative analysis of both traditional and modern testing approaches is conducted based five key criteria: reliability, scalability, maintainability, efficiency, and adaptability. Through this analysis, the advantages, and disadvantages of both these methodologies are outlined, giving the reader a better understanding on which methodology may fit their needs better.

The challenges of adopting modern testing practices, such as the cultural shift needed, increase in skill requirements, and changes in infrastructure requirements are identified and addressed. Additionally, the potential future direction and emerging trends in software testing field such as the use of artificial intelligence, codeless automation, and the use of machine learning are investigated.

This thesis highlights the potential advantages as well as the disadvantages of the use of modern software testing methods in enhancing overall software quality and reliability. It aims to serve as a guide for practitioners to use when making the decision on which testing methodology fits their specific needs the best.

Keywords:

Software Testing, Manual Testing, Automated Testing, Software Development Life Cycle

CONTENTS

VOCABULARY

GSMA: GSM Association

STLC: Software Testing Life Cycle

SDLC: Software Development Life Cycle

PRD: Product Requirements Document

SRD: Software Requirements Document

Jira: An agile project management tool with functionality for bug tracking

Bugzilla: A web-based bug tracking system

ClickUp: A project management platform

ISO: International Organization for Standardization

AI: Artificial Intelligence

1 INTRUDUCTION

In the modern age, software and its quality have become an integral part of our everyday lives. From the applications we use in our everyday lives to pay our bills or to communicate with other people all around the world. According to GSM Associations (GSMA) 2023 study *The State of Mobile Internet Connectivity 2023* the number of people worldwide using a mobile internet was 4.6 billion people or 57% of the global population. The fact that 57% of the world ‘s population is already using a smartphone capable of connecting to the internet only highlights the importance of the production of quality software and thus the importance of quality software testing. Failing to provide secure and good quality software could result in a large financial loss and other security threats to our society. (1.)

Software testing is the act of testing a specific functionality on a given platform or application. Its purpose is to ensure that the quality of the software product is good and meets the client's expectations and requirements. Software testing is typically conducted in tandem with the rest of the development team to provide quick and high-quality feedback on the product being developed.

Software testing is a critical step of the software development lifecycle since it gives valuable feedback on the potential faults that might occur during the development process of the product. Failing to design and implement an efficient software testing solution may cause the project to be delayed or to fail altogether due unforeseen issues and lack of quality assurance.

During this study we will investigate on how to execute quality software testing, which tools to use, which methodologies to use and how to put all this information together in a comprehensive manner. The objective of this thesis is to compare and review the advantages and disadvantages of more modern testing methods when compared to more traditionally used methods. We will also investigate how Artificial Intelligence or AI will play a role in the future of software testing.

2 LITERATURE REVIEW

This chapter acts as an introduction to software quality assurance and the evolution of different software testing methods. During this chapter a thorough comparison between the application of modern software testing methodologies to the more traditionally used methods is made.

2.1 Introduction to Software Quality Assurance

When beginning a project and deciding on how to design and execute testing, Software Testing Life Cycle or STLC guideline is the most used method. STLC can be broken down into six distinct phases.

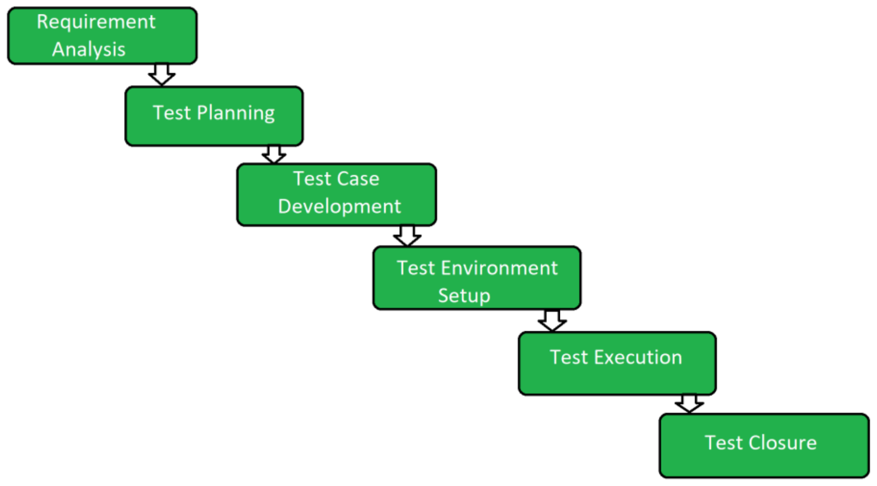


FIGURE 1, Software Testing Life Cycle (2).

The first phase is Requirement Analysis, during this phase the software testing team analyzes and aims to understand the requirements and expectations set for the project. This goal is usually met by reviewing the Product Requirements Document (PRD) or the Software Requirements Document (SRD). During this phase the software testing team may also interview the customer for any additional requirements or preferences which may need to be considered during the next phases. During phase one of the STLC the testing team may also need to fill in missing or incomplete requirements defined in the PRD or SRD.

The second phase is Test Planning, the goal of this phase is to define the testing plan. During the test planning phase all test cases are defined and an estimation on the working effort and cost is created. Typically, the test planning phase begins once requirement analysis is completed. During test planning the test team should identify the objectives and scope of the plan being created, create a distinct plan on which methods are to be used during testing, calculating the resources needed for testing and setting milestones for the testing team to meet the requirements of the project.

The third phase is the Test Case Development, during which the testing team designs test cases to meet the requirements of the PRD or SRD. During Test Case Development clear and easy to understand test cases are written by the test team. To create clear and easily understandable test cases needed action and expected result of the action are defined. The test cases written by the test team are also usually reviewed to ensure highest quality possible. (2.)

The fourth phase in the STLC is Test Environment Setup, during this phase the conditions in which the software is tested will be decided. While this is an important step, the testing team is usually not involved since this decision and development is done either by the customer or the development team.

The fifth phase is Test Execution, during this phase the previously developed test cases are executed. While the tests are executed the software testing team keeps a log of issues found during the testing. This is usually done using tools such as Jira, Bugzilla or ClickUp. Test execution can be done in various ways and broken down into unit, feature and system testing runs depending on the scope desired. During test executions typically both manual and automated tests are run, and their results are logged based on the expected result of the test case defined earlier during the STLC. After testing the results of the test execution are analyzed and if necessary retested for further insight on the issue. After this the results of the test execution will be reported to the relevant authorities in the organization.

The sixth and final step in the STLC is Test Closure, in this final phase all relevant data is collected and documented. It is also made sure that all the necessary test cases were executed, and possible defects or anomalies documented. Possible changes to the test cases previously defined are made and a summary of the test execution is created. During this phase it is also good practice to document lessons learned and to give feedback on the test execution and previous phases of the STLC. (2.)

Overall, when following the STLC software testing a straightforward and efficient process to ensure the quality of a given product. Well planned and documented execution of these six phases will ensure that test cases created, and lessons learned can be used in making possible future projects easier and more efficient to test.

2.2 Evolution of Software Testing

As a practice, software testing has existed for as long as software has been developed. But the terms computer bug and debugging were first used in 1945 by the computer scientist and mathematician Grace Brewster Hopper. However, Joseph Juran is considered the father of software testing. In the year 1951 Joseph Juran released his first book on the subject "Juran's Quality Control Handbook" in which he defined the first three parts of quality assurance in software, quality planning, control, and improvement. In his book Juran defines the meaning of quality as *"Quality means those features of products which meet customer needs and thereby provide customer satisfaction. In this sense, the meaning of quality is oriented to income. The purpose of such higher quality is to provide greater customer satisfaction and, one hopes, to increase income. However, providing more and/or better-quality features usually requires an investment and hence usually involves increases in costs. Higher quality in this sense usually costs more." (3.)*

Six years later in the year 1957 Charles L. Baker was the first computer scientist to officially distinguish the difference between software testing and debugging in a review of a book *Digital Computer Programming* by Dan McCracken. A year later in 1958 Gerald M. Weinberg is credited on assembling the first software testing specific team to work on Project Mercury.

Finally, in the year 1979 Glenford J Myers published his book *Art of Software Testing* which further defined software testing as a practice. In his book Myers outlined at the time latest methodologies for designing test cases, code inspections and debugging.

In 1999, Mark Fewster and Dorothy Graham were the first to publish a book on automated software testing *Software Test Automation: Effective Use of Test Execution Tools*. In their book Fewster and Graham outlined how to design and implement an automated testing regime.

By this time software quality had become a critical step in the Software Development Life Cycle or SDLC. Issues in software had caused catastrophes such as the Ariane flight V88 which had failed due to a software bug in 1996 which resulted in an estimated $370 million financial loss. Due to the potential risks in faulty and low-quality software the International Organization for Standardization (ISO) founded a team to develop a new standard for software testing ISO 29199.

Today, software testing is estimated on being a $45 billion industry globally. It is also estimated that 74% of software testers are using automation. Due to the ever-growing market for quality software and the emergence of AI-based test automation software testing may be on the brink of another revolution in the way test cases are designed and executed. (4.)

2.3 Traditional Software Testing Methods

Traditional Software testing methods typically follow the waterfall model. The phases in the traditional waterfall model project are design, requirement analysis, coding, testing and release.

Traditional software testing is conducted in its own separate phase of the STLC after development of the given product has finished. When using the waterfall model testing relies heavily on well-designed test cases and is less flexible for changing requirements during the project.

The fact that traditional software testing typically takes place towards the end of the project, multiple issues may emerge. During the testing phase large project breaking issues may emerge and cause delays in software delivery. Using this method estimating costs of developing products is also harder since testing is not constantly executed during the development project.

However, there are some upsides on using traditional software testing. Since the testing is conducted in its own phase after all the development work has finished, well-made testing plan can detect and document defects efficiently in the finished product before it is shipped to the customer.

Some may argue that following this methodology results in a more polished and better tested product when comparing to its more modern counterpart Agile Testing.

2.4 Modern Software Testing Methods

A more modern approach to conducting software testing would be Agile Testing. Agile testing is an approach which aligns with the idea of Agile software development, its goal is achieve cyclical testing during the software development process. When using the Agile method, the software developed is constantly being tested by everyone involved in the development process including testers, developers, and business analysts.

Agile testing is typically chosen when developing a product using Agile development methods such as Scrum or Kanban. Agile testing starts from the very beginning the project and continues until the very end. It is conducted in cycles during the project to ensure high quality throughout the development process. Since this method of testing is continuously taking place during the project, it is more flexible to changes in the requirements set by the customer.

In Agile testing, testing is prioritized based on risks the defects found present to the project. As an example, on Jira defects are rated from risk rating Lowest to Highest. Higher risk rating tasks are worked on before the lower risk ones to ensure quick and good quality changes to the product.

Typically, Agile testing is conducted on release basis. During the development process multiple releases or versions of the product are created for testing and later release, this is known as release testing. During the project there may also be larger test sets executed, these are commonly known as System or Feature testing depending on the scope of the test execution.

In their 2021 study *15th annual State of Agility report* Digital.ai found that 86% of software development teams have adopted Agile. This shows that large part of software development teams is now using Agile methodologies in their development and the rise of prevalence of agile testing. *(5.)*

2.5 Comparison between Traditional and Modern Software Testing Methods

When starting a software development project, a very important choice on which testing methodology to use during the project must be made. To make this decision the projects managerial team should take a critical look at the requirements, expectations, and the organization to make an educated choice on which methodology to use.

While Agile software testing may have its upsides when it comes to quick development it might not always be the right choice for the project in question. If the project has well defined requirements and if the requirements are not expected to change during the development process more traditional testing might be the better option since it provides a way to ensure the highest software quality.

If the requirements of the project are not as well defined and prone to changing during the development process, an agile approach to testing is more suitable. Since agile software testing is conducted in a cyclical and continuous manner, it is more flexible to implement in a situation such as this.

While there currently are no studies on the adaptation of Agile methodologies in software testing itself, the studies made on Agile adaptation in software development teams as a collective indicate that it is the more common methodology to use nowadays. Since 86% software development teams have adopted Agile in their projects it only makes sense that agile software testing is following the trend.

However, traditional software testing has its place even today, in large projects where the requirements and expectations are well defined and not prone to change choosing to test software utilizing traditional methods may be a valid choice. Using traditional methods of having a separate phase in the project solely dedicated for testing the product made provides highest quality possible. In projects where minor defects could result in tragedies dedicating more time to software testing is necessary.

OVERVIEW ON MODERN SOFTWARE TESTING METHODS

In this chapter, modern agile software testing methods are investigated. The characteristics of these methods are classified, and approaches on how to apply agile software testing methodologies into a given software development project are outlined.

While outlining these methods in reference to modern software testing, it is important to acknowledge that they are not solely unique to them. Some of the classifications and characteristics such as system testing, or security testing can also be found in more traditionally used approaches to software testing. However, during this chapter said characteristics will be considered in their modern form in the context to modern software testing methodologies.

2.6 Classification of Modern Testing Methods

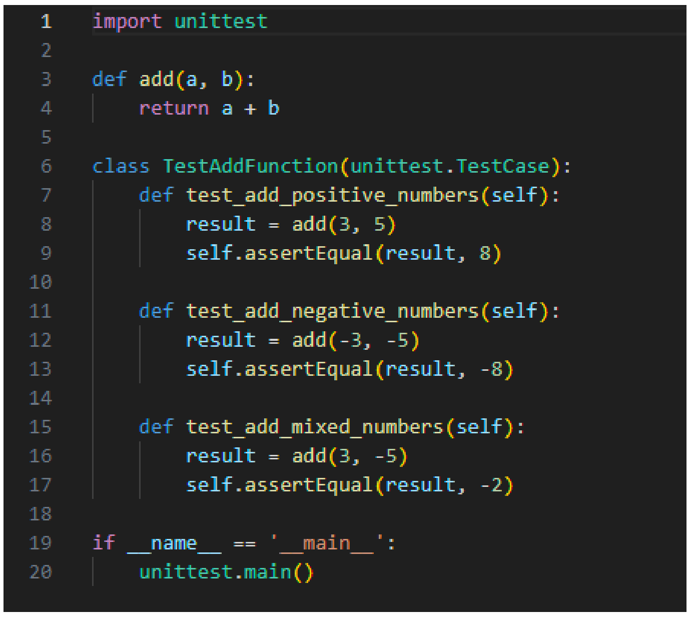
Modern software testing is typically broken down into 8 separate methodologies, these are Unit Testing, Integration Testing, System Testing, Acceptance Testing, Performance Testing, Security Testing, Usability Testing and Compatibility Testing.

2.6.1 Unit Testing

Unit Testing is the first type of testing which occurs during a software development project. In an agile environment, Unit Testing is typically performed by the developers themselves, while development is still ongoing. Unit Testing is used to ensure that the software or feature works as expected before being moved forward in the software development pipeline.

Unit Testing can be executed manually, or it can also be automated to speed up the testing and development process. Well, planned and executed unit testing speeds up the delivery time of the project and helps expand the testing coverage on the product being developed ensuring quality code. (6.)

The example below demonstrates a unit test for a simple add function. It includes three test cases: one for adding two positive numbers, one for adding two negative numbers, and one for adding a positive and a negative number. Each test case uses the assertEqual method to verify that the result of the add function matches with the expected result.



```python
import unittest

def add(a, b):
    return a + b

class TestAddFunction(unittest.TestCase):
    def test_add_positive_numbers(self):
        result = add(3, 5)
        self.assertEqual(result, 8)

    def test_add_negative_numbers(self):
        result = add(-3, -5)
        self.assertEqual(result, -8)

    def test_add_mixed_numbers(self):
        result = add(3, -5)
        self.assertEqual(result, -2)

if __name__ == '__main__':
    unittest.main()
```

2.6.2 Integration Testing

After Unit Testing, integration testing is executed to integrate smaller units into a larger module or component while ensuring that integrating them together does not cause issues. After integrating the units tested earlier the larger segment of the project is tested.

Typically, during integration testing the module is tested via user scenarios mimicking the behavior and actions of the end user. Integration testing may be conducted by the developers themselves or by a separate testing team. (7.)

The goal of integration testing is to ensure that different modules of the software work together seamlessly, for instance, in an online store application, integration testing may involve verifying that the websites user authentication interacts smoothly with the product catalog, enabling users to sign in to the website and browse its catalog without issues. Integration of other functionality of the website such as adding items to the websites shopping cart or progressing to the website's payment screen need also to be tested via integration testing so that they work seamlessly with when integrated to the website's code base.

2.6.3 System Testing

System testing is a black box testing method used to ensure that the completed and integrated system meets the requirements set for the project. Black box testing mainly focuses on the functionality of a given system without delving deep into the inner workings of it.

Typically, during System Testing the finished systems or releases functionality is thoroughly tested by an independent software testing team. (8.)

2.6.4 Acceptance Testing

Acceptance testing is typically considered the final phase of software testing. During the finished software project is tested to determine that it meets the previously defined project requirements. The finished software is tested both internally and externally, this phase may also be called an alpha or a beta phase of the project's lifetime. During acceptance testing the final changes before release should be made while considering any end-user data or usability issues. (9.)

2.6.5 Performance Testing

During performance testing the software's behavior is evaluated under differing conditions. The goal of this phase is to ensure the software's stability and responsiveness. Typically, performance testing is broken down into four different segments which are, Load Testing, Stress Testing Endurance testing and spike testing.

During load testing the software's performance is tested under conditions expected from the user, it is used to evaluate the software's performance in a typical end-user use case.

Stress testing is used to test the software's behavior in a situation where the load on the software is heavier than usual. The goal of stress testing is to break the software to gauge its behavior in heavy load conditions.

Endurance testing is the act of keeping the software running under a specific load for long periods of time at a time. The goal of endurance testing is to ensure the software's stability under prolonged use.

Spike testing a type of performance testing where the software is flooded with activity be it concurrent users or system activity. It is used to study how the software works during sudden spikes of demand. (10.)

2.6.6 Security Testing

Security testing is the act of evaluating the software for its vulnerabilities. This kind of testing is conducted to ensure the safety and security of the software being developed. Its goal is to find potential loopholes and security risks in the system. If not properly executed the system might have unknown security risks which may lead to data breaches and unauthorized access to the system. (11.)

The most common types of security vulnerabilities which must be taken into consideration during development and testing are SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Broken Authentication and Sensitive Data Exposure (12.)

SQL injection is a type of security vulnerability which occurs when malicious SQL queries can be injected into input fields of an application or service. Through these queries, attackers may gain unauthorized access to sensitive data or in some cases even delete data stored within the SQL database being accessed.

Cross-Site Scripting or XSS involved the injection of malicious scripts to a software system being used by other users. Said scripts can then be executed in the other user's web browser. Often leading to leaking of sensitive data such as user login credentials.

Cross-Site Request Forgery or CSFR attacks are used to trick already authenticated users into executing unwanted actions a software platform. These kinds of attacks exploit the user's trusted web browser by creating forged requests which appear legitimate to gain access to the user's account.

Broken authentication vulnerabilities are the result of improper implementation of user authentication on a software system. These kinds of vulnerabilities can potentially lead to issues such as weak password complexity requirements or lack of protection to brute-force attacks on the system allowing exploiters to gain unauthorized access to user accounts on the platform.

Sensitive Data Exposure vulnerabilities occur when a software system fails to properly protect sensitive information stored on its data storage platforms. Leading causes in sensitive data leaks are improper use encryption, storing sensitive data as plain text and failures in access control on the system.

2.6.7 Compatibility Testing

Compatibility testing is used to ensure that the software works in the wanted environments. This may include ensuring that the system works on different operation systems, web browsers or platforms as expected. The goal of compatibility testing is to ensure a good end-user experience on the platforms supported by the system. (13.)

2.7 Characteristics of Modern Testing Approaches

In recent years, the advancements in technology, changes in software development methodologies and increasing complexity of systems being developed have forced software testing methodologies to evolve. In this section, the focus is on underlying the key features of modern and characteristics of modern software testing methodologies.

2.7.1 Test Automation

Test Automation has become an integral part of software testing. Unlike traditional software testing, automated testing is conducted by utilizing tools and scripts to validate parts of the software system. Using automation in software testing offers multiple advantages when comparing it to manual testing, some of these advantages are increased test case coverage, repeatability and efficiency in testing cases which may take days or even weeks to complete. By automating repetitive test cases, when possible, software testing teams can streamline the testing process, accelerate the release cycles, and improve the software quality while focusing on test cases requiring manual testing on the side. (14.)

2.7.2 Shift-Left Testing

Shift-Left testing is the act of early integration of testing into the SDLC. Traditionally while utilizing the waterfall method of software development, testing is focused on the end of the software development project, this may lead to delays in identifying bugs and cause higher costs on fixing said defects. Shift-Left Testing is the opposite of this methodology, it advocates for integrating software testing as early as possible. Ideally software testing would be implemented during the requirement analysis phase of the project. Utilizing the Shift-Left Testing methodology allows for earlier detection of defects and reduces and workload during the different phases of the SDLC. (15.)

2.7.3 Shift-Right Testing

Shift-Right testing complements shift-left testing by focusing on post-deployment phase of software testing. It is a form of user-centric testing focused on collecting real-time user data and gathering feedback from the end user of the product. The collected data is then used to make continuous improvements and possible changes to the software system. Utilizing Shift-Right testing ensures that the functionality and user-experience of the product meets the expectations of the end-user. (16.)

2.7.4 Continuous Testing

Continuous Testing is another typical characteristic of modern software testing methodologies. It is used to continuously test the code being developed in real time to get rapid feedback on the code changes made. Continuously testing provides the possibility of detecting defects early and gives important data on the stability and robustness of the system being build. (17.)

2.7.5 Exploratory Testing

Exploratory testing is an adaptive approach to software testing in which the testers creativity and knowledge of the product being developed is put to the test. Unlike predetermined test cases exploratory testing encourages the testing team top explore the software system dynamically, potentially detecting defects and identifying possible risks in the software. Exploratory testing works the best in an open environment in which quick feedback can be given to the development team. In this form of software testing, a deep knowledge and understanding of the system being developed is required to differentiate between features and defects in the system. Exploratory testing is a highly important methodology since it can effectively uncover complex defects, corner cases and potential usability issues in the system being developed. (18.)

2.7.6 Shift-left Performance Testing

Nowadays with the ever-rising demand for high-performance software, shift-performance testing has become a critical testing methodology. Shift-performance testing involves testing the software system for its performance, scalability, and reliability under differing load conditions. Its goal is to identify potential bottlenecks, optimize resource utilization, and ensure that the system meets the performance requirements set. Shift-Performance testing is a more modern approach to the traditional performance testing outlined earlier. If integrated early in the SDLC Shift-Performance testing can mitigate issues relating to the performance or user experience of the system later in the project. (19.)

2.7.7 Conclusion

Modern software testing methodologies have distinct characteristics which reflect the needs and expectations of the ever-evolving Software Development Life Cycle. From automated software testing to continuous and exploratory testing, modern software testing methodologies emphasize flexibility, agility and collaboration between the development teams and testing teams. By utilizing and adapting modern software testing methods organizations can adapt to the fast-changing environment of modern software development and accelerate delivery cycles while also providing the best quality software possible.

3 EVALUATION AND COMPARISON

In this chapter, the evaluation and comparison between traditional and modern approaches to software testing are conducted. The advantages and disadvantages of both traditional and modern testing methods will be highlighted to aid in making an informed decision on which methodology to follow in a software development project.

3.1 Evaluation Criteria for Testing Methods

This section outlines a framework for the evaluation of software testing methods by breaking them down into five key criteria: reliability, scalability, maintainability, efficiency, and adaptability. Reliability refers to the accuracy and consistency of test results. Scalability evaluates a given methods adaptability to projects of differing scopes and complexities. Maintainability evaluates how easily existing testing methods can be managed and updated over time. Efficiency measures the effectiveness of a given testing method in achieving project objectives and resource constraints. Adaptability evaluates the methods flexibility in a changing project environment.

3.2 Comparative Analysis of Traditional vs. Modern Software Testing Methods

This section will act as a detailed comparative analysis on traditional and modern software testing methods, considering their key characteristics, principles, and best practices.

Traditional Software Testing Methods are typically characterized by their sequential phases and an emphasis on manual test execution. Traditionally software testing is conducted using the waterfall development model in which software testing is its own individual phase of the project only starting after the software development lifecycle has concluded. While being simple to implement, traditional software testing lacks the capability to change and adapt to quickly changing requirements. (20.)

Modern Software Testing Methods such as agile testing, continuous integration or test-driven development have gained popularity in the recent years in response to the shortcomings of more traditional testing methods. These methodologies place a larger emphasis on team collaboration, test automation and exploratory testing offering quicker feedback loops, improved software quality and adaptability to changes in project requirements. (21.)

Test Coverage: Utilizing modern testing methods often offers a wider range of testing coverage through the use of test automation and continuous testing. By using automation in testing effectively, modern software testing methods can test various aspects of the software in question more efficiently. Test automation allows for the simultaneous execution of test cases, making it easier to detect defects earlier in the development process and more thorough exploration of potential corner-cases. In contrast, more traditional software testing methods rely heavily on labor intensive manual testing potentially leading to limitations on the scalability of the testing coverage. Due to manual testing relying on hands on testing, the risk of human error and the overlooking of defects is higher than while using test automation.

Automation Capabilities: Modern software testing methodologies typically prioritize continuous testing and to achieve this a larger emphasis on test automation must be made. Test automation often leads to faster testing cycles and higher reliability of test results. With the use of automation on testing, simple and repetitive test cases are executed swiftly and consistently while also reducing the potential for human error, test automation also frees up time for the testing team to execute test cases which cannot be automated.

Cost-Effectiveness: Adopting modern software testing methods into a development project requires a larger initial investment than utilizing traditional software testing methods. However, the long-term benefits outweigh the upfront costs, the investment into modern software testing translates to improved testing efficiency, enhancement in software quality and reduced development time-to-market. By developing a scalable and robust test automation infrastructure, organizations can drastically reduce testing expenses in the long run. Thorough and continuous testing also helps to reduce the risk of costly post-release defects.

Suitability in Different Development Environments: Modern testing methodologies are better suited for an agile and adaptive development environment. Agile methodologies, such as Scrum or Kanban emphasize iterative development and continuous feedback-loops which align perfectly with the principles of modern software testing. Traditional software testing methods are characterized by their sequential and rigid testing process lacking the capability to adopt into an environment with frequent changes in project requirements. In such environments a more agile approach is more suitable due to the nature of its flexibility and reliance on test automation allowing software testing teams to quickly adapt to requirement changes and to keep up with the quick pace of software releases.

3.3 Advantages and Disadvantages of Modern Software Testing Approaches

The adaptation of modern software testing methodologies is typically seen as a progressive step in to ensuring software quality and accelerating the software development lifecycle. However, it is highly important to recognize that the transition from using traditional software testing methods does not come without its own challenges and potential drawbacks. Understanding these challenges is vital when making the choice of which methodologies to use in a software development project.

One of the most significant challenges is the initial learning curve often associated with modern software testing methodologies. Adopting these approaches may require the development and testing staff to receive external training through the use of skill development programs. While the upfront cost might seem daunting, training these skills equip teams with the knowledge and skills to use in utilizing modern testing methods achieving faster and more consistent test results. (22.)

Implementing test automation tools and infrastructure represents another upfront investment which organizations must take into consideration. The establishment of robust and scalable automation infrastructure requires a large investment of resources and dedication to set up and to maintain. However, after the initial investment the benefits typically outweigh the costs. The use of automation accelerates testing cycles, improves testing coverage and results in consistent and reliable test results due to reduces potential of human error. (23.)

Integrating modern testing methods into an existing development project, particularly projects working in a legacy environment may lead to challenges in compatibility. While the shift from traditional testing to modern software testing may initially disrupt workflows due to incompatibility in documentation and project requirements, actively addressing these issues organizations place themselves in for a greater position of flexibility for future changes in project requirements. Due to these potential difficulties when first adopting modern testing methods, organizations may decide to use methods from both traditional and modern testing simultaneously to ease the change from one methodology to another. (24.)

However, it is vital to acknowledge that the heavy emphasis on test automation and cyclical nature of testing may not be suitable for all types of projects. Overly relying on automated testing has the potential of leading to blind spots in test coverage and the overlooking of defects, especially if the system being tested is complex with multiple interdependencies.

In conclusion, when making the choice on if to use traditional or modern software testing methods, the choice should be made based on the nature and requirements of the project. While modern testing methods offer a more flexible environment for testing, traditional testing methods still hold their own in projects with well-documented and stable requirement base. By careful comparison of the advantages and disadvantages presented by each of these approaches to testing, organizations can make an informed decision on which to adopt for their specific needs in a project.

4 CHALLENGES AND FUTURE DIRECTIONS

The implementation of modern software testing methodologies may present several kinds of challenges which have to be addressed to achieve their full potential. Some of the most prevalent issues an organization may phase in the adaptation are the need for cultural shift, higher skills set requirements, new tools and infrastructure, test coverage maintenance and the integration of testing into the software development lifecycle.

4.1.1 Cultural Shift

Transitioning from traditional testing to modern software testing methodologies often requires a cultural shift within the organization. Achieving said cultural shift requires commitment from everyone involved in the project. Project participants should be informed on tools, upcoming changes and encourage clear communication within teams. (25.)

4.1.2 Skill Set Requirements

Modern software testing relies heavily on automating the testing process, achieving this requires the software testing team to have skills in programming, test automation frameworks and knowledge of DevOps practices. When migrating from waterfall to modern software testing, training, and upskilling the staff may be required. As of March of 2024, SAFe Agilist (SA) certification meant for enterprises looking to adopt agile, comes at the cost of $995 to $1,295 per person trained adding to the upfront cost of adopting agile into a project. (26.)

4.1.3 Tooling and Infrastructure

The adaptation of modern testing methods requires the use of more specialized testing tools and the development of a test automation infrastructure. This may be a resource intensive task to achieve, especially if working within a legacy environment built while using a more traditional approach to development and testing. However, in the recent decade this has become a lot easier due to the emergence of open-sourced test automation frameworks such as Robot Framework.

4.1.4 Maintaining Test Coverage

While test automation has drastically sped up the testing process, ensuring a comprehensive test coverage remains a difficult and resource heavy task. Challenges in identifying critical test scenarios, maintaining test suites, and addressing potential gaps in the test coverage requires a lot of work. In the recent years tools such as Jira X-ray specifically meant for maintaining and monitoring test coverage have been created to make maintaining test sets and coverage easier.

However, the reliance on test automation presents its own challenges, updates in the software being tested and potential product variants may cause the existing test automation scripts to become incompatible with newest update or variant, leading to more resources required in updating the scripts.

4.1.5 Integration with Development Lifecycle

The integration of software testing into the development lifecycle, especially in agile and DevOps environments is critical for achieving continuous feedback loop and rapid iterations. The synchronization of testing activities with development sprints and or release cycles requires close nit collaboration, communication and coordination amongst cross-functional development and testing teams.

4.2 Future Direction and Emerging Trends in Software Testing

In this section, a deeper look into the potential future directions and emerging trends in the software testing market is taken. Some of the potential future key directions include:

4.2.1 Artificial Intelligence and Machine Learning in Software Testing

In the recent couple of years, the use of Artificial Intelligence (AI) and Machine Leaning (ML) have quickly gained traction in the software testing field. According to a study conducted by Katalon, “The State of Software Quality Report 2024”, 58% of managers and members of senior management see the integration of AI into software testing as a key goal in the coming years. According to the same study, AI will mainly be utilized in test case generation for both manual and automated software testing. (27.)

4.2.2 Codeless Test Automation

Codeless Test Automation is a form of automated software testing which does not require the writing of code to create automated testing. In the recent years, Graphical User Interface (GUI) based tools have emerged for the use in software testing. These kinds of tools are making the software testing process easier to approach and making the initial skill set needed to get into the field lower. Tools such as Selenium IDE and Katalon Studio are an easy way to implement early testing into a lightweight software development project. (28.)

5 CONCLUSION

This thesis has acted as a comprehensive exploration of different software testing methodologies, spanning from traditionally used methods to more modern approaches in use today, evaluating their effectiveness as well as addressing the potential challenges presented in the use and adaptation on them.

This thesis started with an examination of the base foundations of software quality assurance and the historical evolution of software testing from its beginnings to modern day. This investigation highlighted the change from traditional software testing to modern methods to meet the rising requirements in the software development landscape.

Through a literature review, the complexities of traditional and modern software testing methodologies were broken down, revealing their respective strengths and limitations. After which a deeper look into the classifications and characteristics of modern software testing methodologies was made.

Evaluation and comparison of traditional versus modern software testing methodologies provided insights into the quickly changing landscape of software development and testing. This analysis was used to identify the common advantages and disadvantages of modern approaches to software testing, emphasizing the importance of understanding their applicability within different contexts.

However, it is important to acknowledge that the implementation of modern software testing into a development project does not come without challenges. From the cultural shift required to the rise in skill set requirements, multiple obstacles need to be overcome to ensure successful and cost-effective adaptation.

Looking ahead to the future, the rising use of artificial intelligence and machine learning technology in software testing has the potential to revolutionize the software testing field, trough the AI augmented automation of the test case development process and enhancements in AI powered data-analytics and much more.

In conclusion, the drive for ever better quality in software development and testing is ongoing, characterized by challenges and opportunities for innovation. By embracing and following emerging trends and cultivating adaptability, individuals and organizations can navigate the evolving field of software testing with better confidence and resilience to change. The optimistic future direction of software testing is driven by the collective efforts of researchers and testing practitioners in pushing the boundaries of possibility and innovation.

REFERENCES

1. GSMA 2023. The State of Mobile Internet Connectivity Report 2023. Search Date 13/05/2024
   https://www.gsma.com/r/somic/
2. GeeksforGeeks 2024. Software Testing Life Cycle (STLC). Search date 13/04/2024.
   https://www.geeksforgeeks.org/software-testing-life-cycle-stlc/
3. Juran, Joseph Moses 1951. Juran's Quality Control Handbook, p. 20.
4. Radix 2024. Software Testing Statistics. Search date 13/04/2024.
   https://radixweb.com/blog/software-testing-statistics
5. Digital.ai 2021. The State of Agile [PDF]. Search date 13/04/2024.
   https://info.digital.ai/rs/981-LQX-968/images/SOA15.pdf
6. Amazon Web Services. What is Unit Testing? Search date 13/04/2024. https://aws.amazon.com/what-is/unit-testing/
7. TechTarget 2022. Integration Testing. Search date 13/04/2024.
   https://www.techtarget.com/searchsoftwarequality/definition/integration-testing
8. GeeksforGeeks 2023. System Testing. Search date 13/04/2024.
   https://www.geeksforgeeks.org/system-testing/
9. Testsigma. Acceptance Testing Guide. Search date 13/04/2024.
   https://testsigma.com/guides/acceptance-testing/
10. Tricentis 2024. Performance Testing. Search date 13/04/2024.
    https://www.tricentis.com/learn/performance-testing
11. Guru99 2024. What is Security Testing? Search date 13/04/2024.
    https://www.guru99.com/what-is-security-testing.html
12. GeeksforGeeks 2021. Compatibility Testing in Software Engineering. Search date 13/04/2024. https://www.geeksforgeeks.org/compatibility-testing-in-software-engineering/
13. Gergly Kalman. 10 Common Web Security Vulnerabilities. Search Date 28/04/2024
    https://www.toptal.com/cybersecurity/10-most-common-web-security-vulnerabilities
14. Perfecto 2022 What is Test Automation? Search date 13/04/2024.
    https://www.perfecto.io/blog/what-is-test-automation
15. IBM. Shift Left Testing. Search date 13/04/2024.
    https://www.ibm.com/topics/shift-left-testing

16. Katalon. Shift Right Testing. Search date 13/04/2024.
https://katalon.com/resources-center/blog/shift-right-testing
17. IBM. Continuous Testing. Search date 13/04/2024.
https://www.ibm.com/topics/continuous-testing
18. Atlassian. Exploratory Testing. Search date 13/04/2024.
https://www.atlassian.com/continuous-delivery/software-testing/exploratory-testing
19. Webomates 2021. Shift Left Performance Testing. Search date 13/04/2024.
https://www.webomates.com/blog/performance-testing/shift-left-performance-testing/
20. Project Management Academy 2022. How to Transform from Waterfall to Agile Search Date 22/04/2024
https://projectmanagementacademy.net/resources/blog/how-to-transform-from-waterfall-to-agile/
21. GeeksForGeeks 2019. Waterfall Software Testing. Search Date 25/04/2024
https://www.geeksforgeeks.org/waterfall-software-testing/
22. Gury99 2024. What is Agile Testing? Process & Life Cycle. Search Date 25/04/2024
https://www.guru99.com/agile-testing-a-beginner-s-guide.html
23. GSA. Agile Adoptation Challenges and Best Practices. Search Date 13/05/2024
https://tech.gsa.gov/guides/agile_adoption_challenges_and_best_practices/
24. Abstracta 2015. The True ROI of Test Automation. Search Date 13/05/2024
https://abstracta.us/blog/test-automation/true-roi-test-automation/
25. Chisel. The Transition from Waterfall to Agile. Search Date 13/05/2024
https://chisellabs.com/blog/transition-from-waterfall-to-agile/
26. Simplilearn 2024. Agile Certification Cost: Is It Worth the Investment? Search Date 22/04/2024
https://www.simplilearn.com/agile-certification-cost-article
27. Katalon 2024. The State of Software Quality Report 2024. Search Date 23/04/2024
https://katalon.com/state-quality-2024
28. TestGrid 2024. Codeless test automation: A Comprehensive Guide. Search Date 23/04/2024
https://testgrid.io/blog/codeless-test-automation/