Tampere University of Applied Sciences

Software Testing

Basic Technology and Practical Application

Xiaomin Wu

Software Implementation and Testing
February 2023

Bachelor's Degree Programme in Software Engineering

CONTENTS

1 INTRODUCTION

This report relies on the study of software testing.

The aim of this report is to introduce beginners to the fundamental techniques and applications of software testing in different software development methodologies. To achieve this, the report has relied on William E. Lewis's Software Testing and Continuous Quality Improvement (Second Edition) as a reference, along with some online open sources.

Additionally, the report provides a brief overview of the test team structure prevalent in modern society, which can be useful for job seekers.

2 Software Quality and Testing

This chapter briefly introduces software quality, software quality assurance, and software testing.

Software quality refers to the degree to which a software product meets its specified requirements and is free from defects. It encompasses various aspects such as functionality, reliability, performance, and security. Software quality assurance, on the other hand, is a set of processes and activities aimed at ensuring that software products and processes meet the required quality standards. Finally, software testing is a crucial component of software quality assurance that involves and validating the software product or system to ensure it meets the specified requirements and works as expected.

2.1 Software Quality

The concept of quality is a critical aspect of any product, especially in the software industry. Quality can be defined in multiple ways, each with its own significance. One generally accepted definition of quality pertains to meeting requirements, which must be measurable. In this definition, the product's requirements are either met or not met, and it is up to the development team to ensure that they deliver a product that meets all the necessary requirements. However, this definition does not consider the customer's perspective, which is equally important.

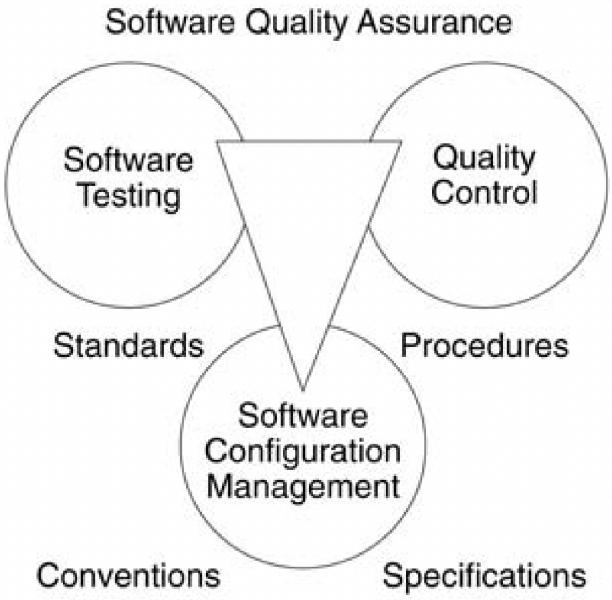
Another perspective of quality, particularly relevant in the software industry, is based on the customer's perception of how well the product satisfies their usage needs. This definition emphasizes the importance of a detailed description of the product's intended use and the quality attributes outlined in the customer's requirements specification. It is not just enough for the product to meet the requirements; it must also satisfy the customer's needs and expectations. This means that quality assurance must be a continuous process that involves the customer throughout the development cycle. This can be achieved through regular customer feedback and involvement in the development process.

Moreover, the perception of quality can vary among customers, and it is important to consider the different usage scenarios and the user's context. Quality must be evaluated not just on technical parameters but also on usability, accessibility, and the overall user experience. This is particularly relevant in today's world, where user expectations are constantly evolving, and software products must keep up with these changes.

Therefore, it is crucial for the development team to define quality requirements from the customer's perspective and continuously measure and improve the product's quality. Quality assurance plays a critical role in ensuring that the product meets the customer's needs and expectations, and a focus on quality can lead to higher customer satisfaction, increased sales, and greater success for the business.

2.2 Software Quality Assurance

Software quality assurance is the systematic activities providing evidence of the fitness for use of the total software product, which is achieved using established guidelines for quality control to ensure the integrity and prolonged life of the software. In other words, it is the collection of activities and functions used to monitor and control a software project so that specific objectives are achieved with the desired level of confidence. It is a planned effort to ensure that a software product fulfills these criteria and has additional attributes specific to the project, for example, portability, efficiency, reusability, and flexibility. Most software quality assurance activities can be categorized into software testing, that is the rest of the report, software configuration management, and quality control. (Lewis, W. 2005)



PICTURE 1. Software quality assurance (Lewis, W. 2005).

2.3 Software Testing

Software testing is a widely adopted risk management strategy to validate that functional requirements have been fulfilled. The testing process includes a thorough inspection of the requirements to ensure that all possible combinations of

inputs and system states have been tested. Despite such comprehensive testing, it is still possible for some defects to go unnoticed.

The primary goal of software testing is to ensure that the design, code, and documentation of the software meet all the imposed requirements. These requirements can be derived from user needs, specifications, code review and inspection criteria, modular and subsystem testing, integrated software testing, and acceptance testing once the code has been fully integrated with the hardware.

Testing is one of the stages of software development lifecycle, its primary objective is to detect defects as early as possible in the software development lifecycle. Early detection and resolution of defects can help reduce costs, improve software quality, and shorten the development cycle.

3 Software Testing Techniques

It is important for software development teams to carefully consider the selection of testing techniques and how they will be used to ensure the quality and reliability of the software being developed. Unit testing, for example, involves testing individual components or units of code to ensure that they function correctly in isolation. Integration testing, on the other hand, involves testing the interaction between different components to ensure that they work together as intended. Similarly, functional testing focuses on ensuring that the software meets its intended functional requirements, while performance testing assesses the software's ability to perform under different load and stress conditions.

With so many different types of testing available, it is crucial for development teams to understand their specific needs and choose the appropriate testing techniques accordingly. The selection criteria are manual or automatic, static or dynamic, functional or structural (or both). The form is placed in Appendix 1.

This chapter provides an overview of popular testing techniques used in the testing community. These techniques include black-box testing, white-box testing, and gray-box testing, each with its own approach to software testing. The chapter also discusses the differences between manual and automated testing.

3.1 Black-Box Testing

Black box testing, as the name suggests, only observes input data and output data, so it is also called functional testing.

Black-box testing is a testing technique that focuses on testing the functionality of a program or system without knowledge of its internal structure or logic.

The tester develops test conditions based on the program's specifications and verifies that the program's functionality conforms to those specifications without any knowledge of the internal structure or logic. The tester creates test cases based on input data and expected output data and verifies whether the program or system behaves as expected.

In addition to its simplicity and ease of understanding, black-box testing also offers the advantage of being independent of the programming language and technology used to develop the software. This means that the testers do not require in-depth knowledge of the internal code and structure of the program, making it easier to involve non-technical stakeholders in the testing process. Furthermore, black-box testing can be performed in parallel with the development process, allowing for early detection and resolution of defects before they become more costly to fix. This can ultimately save time and money in the software development process.

However, it is not possible to achieve exhaustive input testing since every possible input condition or combination cannot be tested. Additionally, black-box testing may not detect errors or deliberate mischief on the part of a programmer, as there is no knowledge of the internal structure or logic of the program.

3.2 White-Box Testing

White-box testing, also known as structural testing, involves testing the internal structure and logic of a program or system.

The tester examines the code paths and logic, such as basis path analysis, statement coverage, branch coverage, condition coverage, and multiple condition coverage. By doing so, the tester can ensure that the code is functioning as intended and can identify any errors or deliberate sabotage by the programmer.

An advantage of white-box testing is that it provides thorough coverage of the code and can detect errors that may not be apparent in black-box testing. Additionally, because the tester has knowledge of the internal structure, they can create more targeted and efficient tests.

However, one limitation of white-box testing is that it does not verify the correctness of the specifications. It only focuses on the internal logic of the code and may not detect errors that are not related to the code structure. Furthermore, it may not identify missing paths or data-sensitive errors unless there are detailed specifications available.

It's important to note that white-box testing alone cannot test all possible code paths since there are too many combinations. Therefore, it's often used in combination with black-box testing to achieve better coverage and ensure that the program meets the requirements.

3.3 Gray-Box Testing

Gray-box testing is a testing technique that combines both black-box and white-box testing approaches. Unlike black-box testing, where the tester focuses only on the program's functionality against the specification and has no knowledge of

the internal structure, and unlike white-box testing, where the tester examines the program's paths of logic. Gray box testing is a combination of both black box testing and white box testing, allowing for more flexibility in the testing process.

The tester studies the requirements specifications and communicates with the developer to gain a better understanding of the internal structure of the system. By doing so, the tester can design implied tests that clear up ambiguous specifications and reduce the number of necessary tests. For example, if the tester notices that a certain functionality seems to be reused throughout the application, communicating with the developer and understanding the internal design and architecture can eliminate many tests because the functionality may only need to be tested once.

One advantage of gray-box testing is that it combines the strengths of both black-box and white-box testing. The tester can design tests based on the program's requirements while having knowledge of the internal structure of the system.

However, a disadvantage of gray-box testing is that it relies on communication between the tester and the developer, which can be time-consuming and may not always be effective in clarifying ambiguous specifications.

3.4 Manual versus Automated Testing

It is more crucial to differentiate between manual and automated testing.

Manual testing involves a human tester interacting with the software or APIs, which can be expensive and prone to human error.

Automated testing, on the other hand, involves a machine executing a pre-written test script, which can range in complexity from testing a single method to executing complex UI actions. Automated testing is more reliable than manual testing, but the quality of the tests depends on the quality of the test scripts. Automated

testing is a vital aspect of continuous integration and continuous delivery, enabling scalable QA processes.

However, manual testing, particularly exploratory testing, still has value and can complement automated testing to provide a comprehensive approach to software testing.

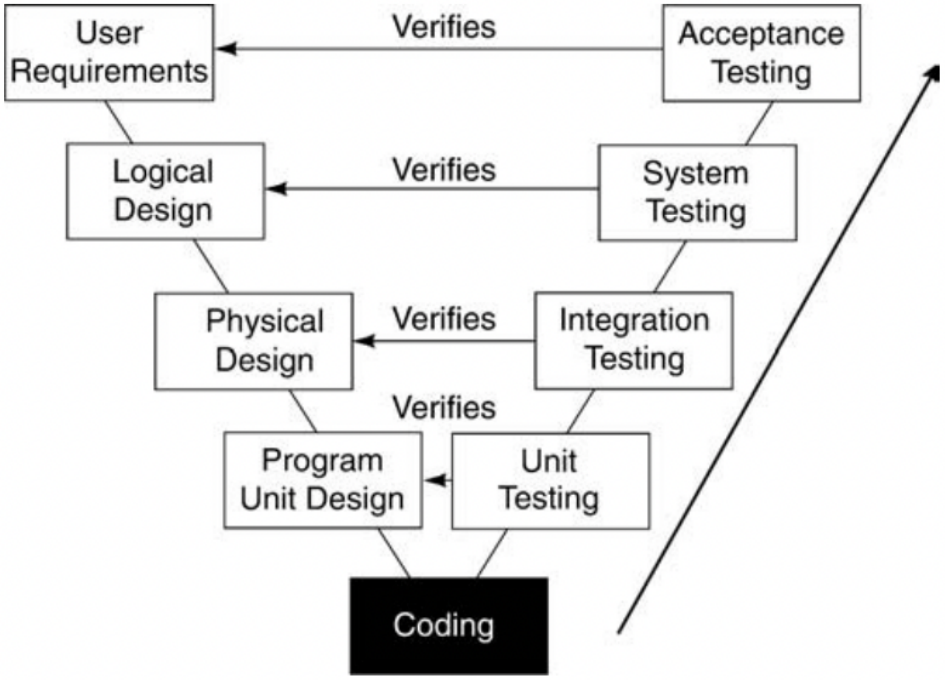
4 Application of Software Testing

This chapter mainly describes the practical application of software testing in the process of software product development. Taking the three development methods of waterfall development method, spiral development method, and DevOps as examples, how to realize software testing among them is explained in turn. Some of the testing techniques mentioned are also briefly explained here.

4.1 Waterfall Development Methodology

The waterfall approach to software development is a linear, sequential approach that divides the development cycle into discrete phases with a rigid beginning and end. The phases typically include requirements gathering, analysis, design, coding, and testing can accompany each step to ensure requirements are met. The waterfall working method that introduces the testing process is also known as the V method.

It's worth noting that while the waterfall approach is still used in some contexts, it has largely been superseded by more iterative and flexible development methodologies such as agile and DevOps. These newer approaches emphasize collaboration, flexibility, and continuous improvement over rigid phases and sequential handoffs.



PICTURE 2. Testing process during waterfall development (Lewis, W. 2005).

4.1.1 Acceptance Testing Verifies User Requirements

Acceptance Testing is a critical phase and the final testing stage in the waterfall model of software development, where the software system is evaluated to determine if it meets the user's requirements.

The User Requirements phase is the first step in the waterfall model where the needs of the users are analyzed and documents detailing their needs are produced.

Acceptance testing is a type of user-run testing that employs black-box techniques to ensure that the system operates according to its specifications. The test plan outlines the acceptance test procedure that needs to be strictly followed. If errors are found, acceptance testing continues until all relevant functionality has been tested.

However, some projects do not require formal acceptance testing, especially when end users are continuously involved throughout the development cycle. Acceptance tests are usually a subset of one or more system tests and can be measured using parallel testing or benchmarking.

Parallel testing is a way to compare the new system with the old system to make sure it works as well or better than the old system. Benchmarks are a way to compare the new system with a standard set of results.

4.1.2 System Testing Verifies Logical Design

System testing plays a significant role in the software development process by validating the structural integrity of the software system before it is deployed to end-users. The primary objective of system testing is to ensure the accuracy of the software's logical design.

During the requirements phase, the business requirements are defined. The logical design phase then refines these business requirements to prepare for the system specification that will be used during physical design and coding. In other words, the logical design phase takes the business requirements and translates them into a technical specification that developers can use to build the software system.

Once the software system is built, it undergoes system testing. System testing involves testing the system to ensure that it functions correctly and is fit for use. This testing is based on the system/acceptance test plan, which defines the procedures for executing the tests.

The system tests are based on the quality attributes that were specified in the software quality assurance plan. These tests verify that the functions of the software system are carried out correctly and that certain non-functional characteristics are present. Examples of non-functional characteristics include usability, performance, stress, compatibility, conversion, and document testing.

The three testing techniques mentioned above: black box testing, white box testing and gray box testing, may be used in this step.

Overall, system testing is an essential step in the software development process that verifies the logical design of the software system. It ensures that the software system is functioning correctly and is fit for use before it is released to end users.

4.1.3 Integration Testing Verifies Physical Design

Physical design is evaluated during integration testing, which is a crucial part of software testing that assesses whether all software components interact correctly. Integration testing is conducted to identify any errors that may be introduced by merging individual program unit-tested modules.

The physical design phase comes after the logical design phase and focuses on how the requirements can be automated. In this phase, the architecture of the system is developed to ensure that the design is structurally sound and can achieve the intended results.

Integration testing is a type of testing that is designed to ensure that all software components interface properly. This type of testing does not verify whether the system is functionally correct, but it checks that it performs as designed. Integration testing should not begin until all individual program units are known to perform according to their specifications.

The primary goal of integration testing is to ensure that the units integrate correctly, parameters are passed correctly, and file processing is correct. This test is essential to identify errors introduced by combining individual program unit tested modules. There are various integration testing techniques such as top-down, bottom-up, sandwich testing, and thread testing. These techniques ensure that the integration testing process is thorough and that all software components interface properly.

4.1.4 Unit Testing Verifies Program Unit Design

Unit testing is a vital aspect of software testing that focuses on testing individual units or modules of a program or system. The goal of unit testing is to ensure that each unit performs its designated function accurately before integrating it into the larger system.

During the design phase, the physical architecture or structure of the system is developed. In the program unit design phase, the detailed design is created by making specific algorithmic and data structure choices. This detailed design specifies the flow of control that will make it easy to translate into program code using a programming language.

Unit testing has several advantages, including the ability to manage the integration of smaller units into larger ones more efficiently, the ability to fully test the logic of the code with fewer tests, and the ability to automate testing for increased efficiency and reusability.

Examples of units that can be tested include modules, GUI components like windows and menus, batch programs, online programs, and stored procedures.

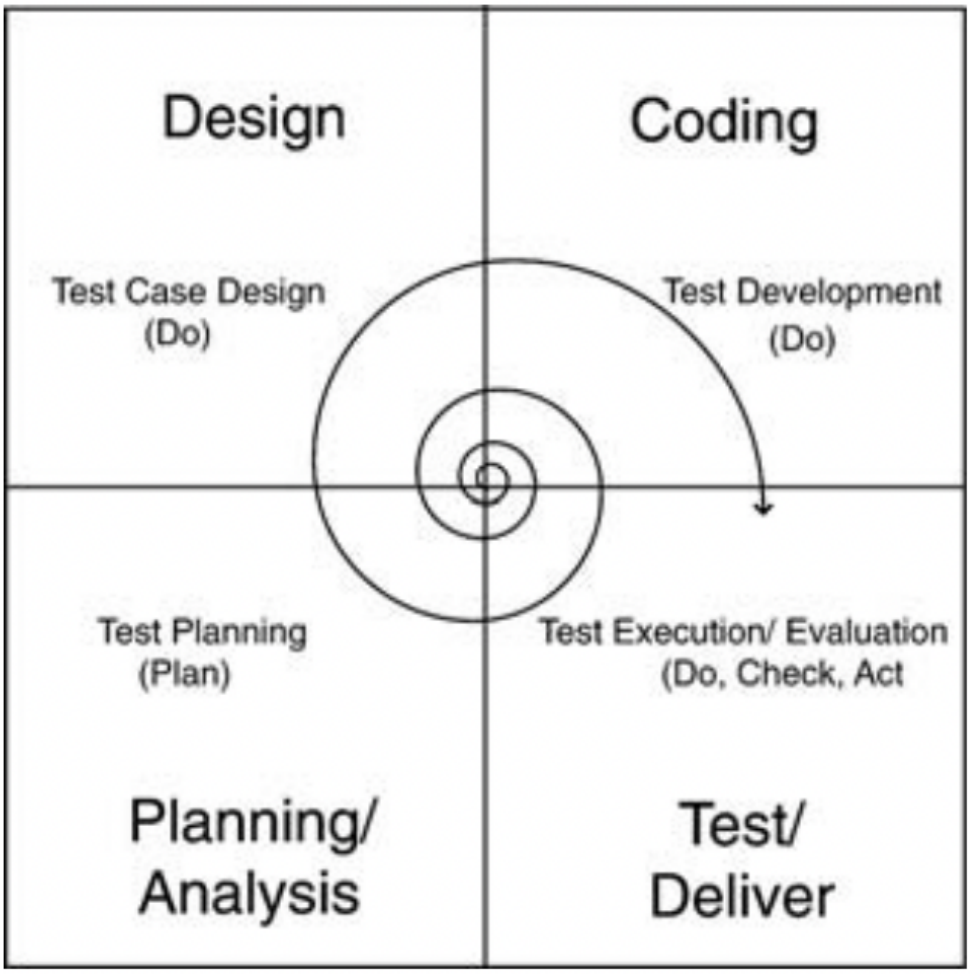
4.2 Iterative/Spiral Development Methodology

Spiral methodologies are an alternative to the traditional waterfall approach in systems development, which is a sequential solution development approach.

The main problem with the waterfall model is that the development process is too rigid resulting in the long elapsed time for delivering the product. In contrast, the spiral approach expedites product delivery by building a small but functioning initial system that is quickly delivered and then enhanced in a series of iterations.

With iterative feedback, users do not have to define every feature correctly and in full detail at the beginning of the development cycle.

The spiral approach performs the analysis-design-code-test phases on a microscale within each spiral or cycle and is often associated with prototyping and rapid application development.



PICTURE 3. Spiral Development Methodology (Lewis, W. 2005).

4.2.1 Spiral Testing

Spiral testing is a process of iterative development in which developers build a system incrementally and revise it at the end of each phase. The four major phases of system development are planning/analysis, design, coding, and test/deliver, which are represented in quadrants in Picture 3.

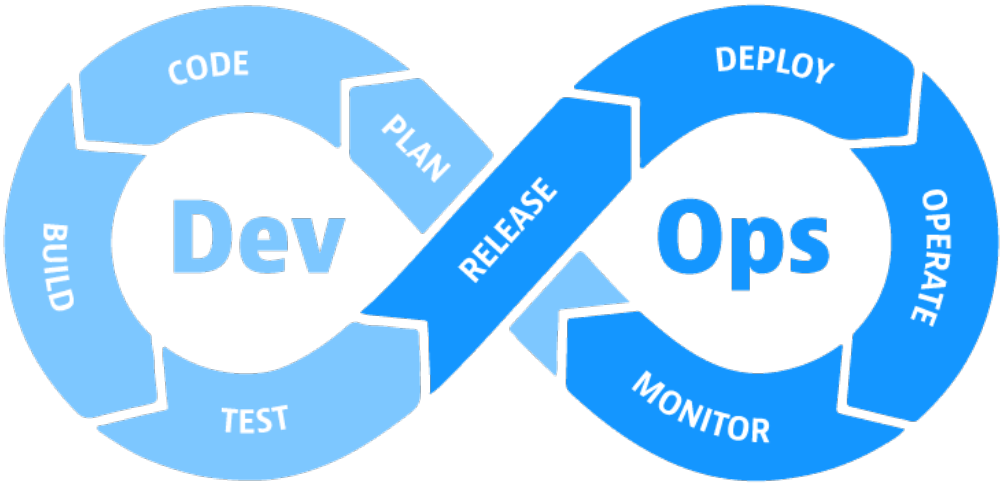
Spiral testing involves test planning, test case design, test development, and test execution/evaluation. Unlike traditional testing, spiral testing is dynamic and may never be completed in the traditional sense of a fully delivered system.

The approach emphasizes cooperation between quality assurance and the development organization to define the test criteria when requirements are not fully available.

The spiral approach, while more flexible than the waterfall methodology, can lead to the development team overlooking user requirements and ultimately failing user verification. Quality assurance plays a crucial role in ensuring that user requirements are met in a spiral approach.

4.3 DevOps

DevOps is a modern methodology that aims to bridge the gap between software development and IT operations by employing a set of practices, tools, and cultural values that promote teamwork, collaboration, and automation. With DevOps, developers and operations teams work together throughout the product lifecycle, creating a continuous process that enhances the speed and quality of software deployment.



PICTURE 4. DevOps (Saif G, 2021).

This approach replaces the traditional siloed model, where development and operations teams work in isolation, with a multidisciplinary team that possesses a range of skills across the application lifecycle. As a result, DevOps can provide many benefits, including faster and smoother releases, greater efficiency, improved security, and higher quality products, leading to greater satisfaction among teams and customers alike.

4.3.1 Testing in DevOps

DevOps Testing is a methodology that enables organizations to streamline their software delivery processes by automating various testing processes, facilitating collaboration among teams, and reducing delivery times. It is a perfect blend of development and operations, with a primary focus on fostering the needs of clients in the best possible ways.

The DevOps Testing process involves the automation and streamlining of the entire software delivery lifecycle. Many companies adopt DevOps testing strategies by starting with the agile practice of Continuous Integration (CI). This practice requires developers to check their code into a shared repository multiple times throughout the day. An automated build then verifies each of these check-ins, allowing teams to quickly detect conflicts and errors.

Automation testing in the DevOps tools and culture is used to reduce the need for human intervention, with the goal of verifying the overall functionality of a product. This process involves using automation frameworks and DevOps testing tools to test the product's functionality. Automation testing helps to detect bugs, minimize human errors, and increase the overall reliability of the product. The introduction of DevOps test automation has changed the role of the QA team, who previously had to wait until the product was perfected before releasing it. With DevOps, multiple teams can work simultaneously to develop and test the product, reducing delays in product launches. (Intellipaat, 2023)

The term DevOps is derived from the words "Development" and "Operations." DevOps Testing brings together the Development and Operations teams to create a well-coordinated collaboration. Quality engineering and security are other crucial aspects of DevOps Testing that ensure the development of high-quality products.

There are several reasons why DevOps Testing has gained popularity in organizations worldwide. One of the primary reasons is its ability to accelerate software delivery, as automation reduces the delivery time. DevOps Testing also facilitates collaboration between development and operations teams, leading to a more efficient workflow. Moreover, DevOps Testing enables companies to respond quickly to changes in the market and customer requirements.

In conclusion, DevOps Testing is a powerful methodology that has transformed the software delivery process. By automating testing processes, facilitating collaboration, and reducing delivery times, DevOps Testing enables companies to deliver better quality products. The DevOps Testing lifecycle is an iterative process that involves several stages, ensuring that software is delivered continuously and reliably.

5 Test Team

The roles and responsibilities are described in this chapter. From the above, testing plays a vital role in the success of a software product. The structure of the test team is composed differently in many companies and even in each project. The assignment of software testing roles depends on the specific tasks of the project and the testing budget. But beyond that, there's the team location, knowledge, skills, and hands-on experience in software testing. The main factors of influence may also change over time. (Olga, 2022)

5.1 Manager

The roles of Test Manager and Software Quality Assurance (SQA) Manager are critical in ensuring the success of software testing and overall quality of the product.

While Test Manager plays a key role in designing the test strategy, organizing the test process, and tracking the test progress, SQA Manager takes on a more senior and strategic role in managing the test managers and ensuring that the QA strategies and methods are formulated and applied across all products of the company.

As a senior manager, the SQA Manager takes ultimate responsibility for the quality of the software products, and must ensure that manual and automated testing efforts are successful across all testing teams in the organization. This involves collaborating with other departments such as development, product management, and customer support to establish best practices and standards for testing and ensuring that they are being followed.

By working together, Test Managers and SQA Managers can create a strong foundation for software quality and testing, ensure that the final product meets the highest standards of quality and customer satisfaction.

5.2 Test Engineer

There may be many test engineers in different positions in the test team, depending on the specific structure of the project. There may be several Quality Control Engineer (QC), Quality Assurance Engineers (QA), Test Analysts, Test Architects, and other QA roles. Sometimes they divide their authority and responsibilities with each other. Here are a few common test engineer positions.

The term Quality Control Engineer (QC) comes from manufacturing. QC in the software industry is responsible for testing new products and determining

whether these products meet business requirements in terms of reliability and functionality, and testing efforts focus on finding defects.

The responsibilities of the Quality Assurance Engineer (QA), including determining test procedures for testing software applications in the most effective manner, designing test suites, creating test cases and documentation, and executing all levels of testing. QA runs tests according to established testing standards and utilizes various testing techniques to ensure high quality, analyze and report on test results to identify and troubleshoot errors before the product is released into production.

Test Analysts analyze requirements and acceptance criteria, design software test documentation, including test plans, link tests to requirements, run tests, analyze and record results. This role focuses on business issues and should have a big picture perspective, good planning, and organizational skills. In addition to the aforementioned responsibilities, Test Analysts also collaborate with other team members to identify areas for testing improvement and optimization. They work closely with developers and business analysts to ensure that testing is thorough and comprehensive, and that the final product meets customer expectations. Test Analysts are also responsible for creating and maintaining test data, monitoring the performance of the test environment, and communicating testing progress and results to relevant stakeholders. In short, Test Analysts are a crucial component of any software development team, helping to ensure that products are of high quality and meet customer needs.

Test Automation Engineers are an important part of the testing team responsible for developing and implementing automated testing procedures to improve testing efficiency and accuracy. They work closely with the Test Analysts to identify which test cases are suitable for automation and then create scripts to automate them. The role is responsible for designing and maintaining the test automation frameworks, identifying and reporting on defects found during automated testing, and collaborating with the development team to ensure that the software product meets the specified quality standards.

5.3 Collaborator

Ensuring product quality is not just limited to the testing team; rather, it is the responsibility of every participant involved in the project. This implies that all stakeholders, including the development team (Devs), project management team (PM, PO), and the team responsible for collecting, processing, and testing the requirements, should have a higher degree of concern for product quality. All parties involved must work together to ensure that the software meets all specified requirements, and quality is maintained throughout the development lifecycle. Therefore, a collaborative approach between all teams involved is critical to delivering high-quality software.

6 CONCLUSIONS AND DISCUSSION

Software testing is a crucial aspect of software development that ensures that the final product meets the desired quality standards. This report delves into the various aspects of software testing, starting with an introduction to software quality and testing.

The chapter on 'Software Quality and Testing' emphasizes the importance of software testing in achieving the desired software quality. It also explores the relationship between software quality, software quality assurance, and software testing. While software quality refers to the characteristics of software that make it useful and reliable, software quality assurance is a set of activities that ensure that the software development process follows established standards and procedures. Software testing, on the other hand, is the process of evaluating the software to ensure that it meets the specified requirements.

Moving on to the chapter on 'Software Testing Techniques,' the report delves into the five most common testing techniques. Black box testing is a technique that

involves testing the software without any knowledge of its internal workings. White box testing, on the other hand, involves testing the software's internal logic, code structure, and implementation details. Gray box testing is a combination of both black box and white box testing, where the tester can more flexibly test whether the software meets the requirements. Manual testing is the process of manually testing the software to identify any defects or bugs, while automatic testing is a process of testing the software using automated tools and scripts.

In the chapter on the 'Application of Software Testing,' the report provides examples of three popular software development working methods in different eras and analyzes the application of software testing in them. The traditional waterfall work mode, which was popular in the past, has been eliminated by most companies. However, software testing is still instructive to us because it introduces what functions the application of the test needs to meet at different stages of the project. Spiral and DevOps are popular working methods that are widely used in technology companies today. Both methods are designed to enable rapid delivery and frequent team communication, making them ideal for agile software development. Both Spiral and DevOps methodologies recognize the importance of software testing in ensuring the quality of the final product. Unlike the Waterfall model, in which testing is performed only at the end of the development cycle, both Spiral and DevOps integrate testing throughout the development process. This approach enables early detection and resolution of bugs and issues, reducing the risk of delays and cost overruns.

Finally, in the last chapter, 'Test Team,' the report describes the structure of the current mainstream test team, providing a perspective for job seekers. The test team is responsible for ensuring that the software meets the desired quality standards. The team is typically composed of a test manager, test engineer, and test analyst. The test manager oversees the entire testing process, while the test lead coordinates the testing activities. The test engineer is responsible for designing and executing the tests, while the test analyst is responsible for analyzing the test results and reporting defects. The structure of the test team can vary

depending on the organization and project requirements. Ensuring product quality is not only the work of the testing team, but also a topic that the coordination department needs to pay attention to.

In conclusion, software testing is a critical process that ensures the quality of software products. This report has provided an overview of the basic knowledge of software testing, including the different testing techniques and their applications in various software development methodologies. Additionally, we have discussed the structure of the current mainstream test team, providing insights for job seekers. It is essential to select the appropriate testing technique and software development methodology based on project requirements and constraints to ensure the desired quality standards are met.

REFERENCES

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

Olga Sheremeta. 2022. Roles & Responsibilities in a Software testing team. Read 26 February 2023. https://testomat.io/blog/roles-responsibilities-in-a-software-testing-team/

Saif Gunja. 2021. What is DevOps? Unpacking the purpose and importance of an IT cultural revolution. Read 7 March 2023. https://www.dynatrace.com/news/blog/what-is-devops/

Intellipaat. 2023. DevOps Testing Tutorial. Read 7 March 2023. https://intellipaat.com/blog/devops-testing-tutorial/

APPENDICES

Appendix 1. Testing Technique Categories (Lewis, W. 2005).

Exhibit 2.1. Testing Technique Categories

Technique	Manual	Automated	Static	Dynamic	Functional	Structural
Acceptance testing	x	x		x	x	
Ad hoc testing	x				x	
Alpha testing	x			x	x	
Basis path testing		x		x		x
Beta testing	x			x	x	
Black-box testing		x		x	x	
Bottom-up testing		x		x		x
Boundary value testing		x		x	x	
Branch coverage testing		x		x		x
Branch/condition coverage		x		x		x
Cause-effect graphing		x		x	x	
Comparison testing	x	x		x	x	x
Compatibility testing	x	x				x
Condition coverage testing		x		x		x
CRUD testing		x		x	x	
Database testing		x		x		x
Decision tables		x		x	x	
Desk checking	x			x		x
End-to-end testing	x	x			x	

(continues)

Appendix 1. Testing Technique Categories (Lewis, W. 2005).

Exhibit 2.1. Testing Technique Categories (Continued)

Technique	Manual	Automated	Static	Dynamic	Functional	Structural
Equivalence partitioning		x		x		
Exception testing		x		x	x	
Exploratory testing	x			x	x	
Free form testing		x		x	x	
Gray-box testing		x		x	x	x
Histograms	x				x	
Incremental integration testing	x	x		x	x	
Inspections	x		x		x	x
Integration testing	x	x		x	x	
JADs	x				x	x
Load testing	x	x		x		x
Mutation testing	x	x		x	x	
Orthogonal array testing	x		x		x	
Pareto analysis	x				x	
Performance testing	x	x		x	x	x
Positive and negative testing		x		x	x	
Prior defect history testing	x		x		x	
Prototyping		x		x	x	
Random testing		x		x	x	

(continues)

Appendix 1. Testing Technique Categories (Lewis, W. 2005).

3 (3)

Range testing		x		x	x					
Recovery testing	x	x		x		x				x
Regression testing				x		x				
Risk-based testing	x		x				x			
Run charts	x		x				x			
Sandwich testing		x		x						x
Sanity testing	x	x		x			x			
Security testing	x	x								x
State transition testing		x		x			x			
Statement coverage testing		x		x						x
Statistical profile testing	x		x				x			
Stress testing	x	x		x						
Structured walkthroughs	x			x			x			x
Syntax testing		x	x	x			x			
System testing	x	x		x			x			
Table testing		x		x						x
Thread testing		x		x						x
Top-down testing		x		x			x			x
Unit testing	x	x	x							x
Usability testing	x	x		x			x			
User acceptance testing	x	x		x			x			
White-box testing		x		x			x			x