K. KORKMAZ ONAT

ATILIM UNIVERSITY 2024

UTILIZING THE SOFTWARE TESTING LEVELS: INSIGHTS FROM THE SOFTWARE INDUSTRY

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ATILIM UNIVERSITY

KÜBRA KORKMAZ ONAT

A MASTER OF SCIENCE THESIS
IN
THE DEPARTMENT OF COMPUTER ENGINEERING

JANUARY 2024

UTILIZING THE SOFTWARE TESTING LEVELS: INSIGHTS FROM THE SOFTWARE INDUSTRY

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ATILIM UNIVERSITY

BY

KÜBRA KORKMAZ ONAT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
THE DEPARTMENT OF COMPUTER ENGINEERING

JANUARY 2024

Approval of the Graduate School of Natural and Applied Sciences, Atilim University.

Prof. Dr. Ender KESKİNKILIÇ
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of **Master of Science in Computer Engineering, Atılım University**.

Prof. Dr. Gökhan ŞENGÜL
Head of Department

This is to certify that we have read the thesis UTILIZING THE SOFTWARE TESTING LEVELS: INSIGHTS FROM THE SOFTWARE INDUSTRY submitted by KÜBRA KORKMAZ ONAT and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Damla Topallı
Supervisor

Examining Committee Members:

Assoc. Prof. Dr. Cansu Çiğdem Ekin
Computer Eng. Department, Atılım University

Asst. Prof. Dr. Damla Topallı
Computer Eng. Department, Atılım University

Assoc. Prof. Dr. Gül Tokdemir
Computer Eng. Department, Cankaya University

Date: *04/01/2024*

Name, Last Name : Kübra, Korkmaz Onat

Signature :

ABSTRACT

UTILIZING THE SOFTWARE TESTING LEVELS: INSIGHTS FROM SOFTWARE INDUSTRY

Korkmaz Onat, Kübra
M.S., Department of Computer Engineering
Supervisor : Asst. Prof. Dr. Damla Topallı

January 2024, 48 pages

Software testing levels play a crucial role in assuring software quality, by identifying defects and bugs early, improving stability of the software, validating compliance with requirements, enhancing user satisfaction, and lowering costs and risks related to software defects. If the defects are detected timely, in an earlier stage of the software development process, the software developed will be in higher quality and the risks of the system failures will be reduced. Additionally, the testing levels ensure that the components of the system work together correctly, which leads to improved reliability of the software. Another important aspect is to ensure that the software meets the user expectations, meeting the intended functionality and performance. Hence, the quality of the software being created is directly affected by the proper implementation of testing levels in the Software Development Life Cycle (SDLC). In the literature, four main testing levels are discussed: unit testing for testing the individual units or components, integration testing for testing the interactions and interfaces between different components and modules, system testing to test the system as a whole and user acceptance testing to test the software against user's requirements and expectations. Accordingly, the main aim of this research is to review testing levels and methods suggested in the literature, understand their impact on the software quality and analyze how these testing levels are used in the software industry. In this respect, a semi-structured interview is conducted with ten software experts from the Software Industry. The questions of the interview include: which of the test levels contribute the most to software quality, which of test levels are used in their current projects, the testing strategy used in their projects, which factors are affecting the choice of the testing strategy and who performs the tests, the developer or an independent team is better in this consideration. The results give insight to the software community regarding the use of test levels and how effective the test levels are in terms of software quality for specific domains of the projects.

Keywords: Software Quality, Software Testing, Testing Levels, Software Testing Pyramid.

ÖZ

YAZILIM TEST SEVİYELERİNİN UYGULANMASI: YAZILIM SEKTÖRÜNDE BİR ÖRNEK ÇALIŞMA

Korkmaz Onat, Kübra

Yüsek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi : Dr. Öğr. Üy. Damla Topallı

Ocak 2024, 48 sayfa

Yazılım test seviyeleri, kusurları ve hataları erken tespit ederek, yazılımın kararlılığını artırarak, gereksinimlere uygunluğu doğrulayarak, kullanıcı memnuniyetini artırarak ve yazılım kusurlarıyla ilgili maliyet ve riskleri azaltarak yazılım kalitesinin güvence altına alınmasında çok önemli bir rol oynar. Hataların zamanında tespit edilmesi durumunda, yazılım geliştirme sürecinin daha erken bir aşamasında, geliştirilen yazılım daha kaliteli olacak ve sistem arızası riskleri azalacaktır. Ek olarak, test seviyeleri sistem bileşenlerinin birlikte doğru şekilde çalışmasını sağlar ve bu da yazılımın güvenilirliğinin artmasına yol açar. Bir diğer önemli husus, yazılımın kullanıcı beklentilerini karşılamasını, amaçlanan işlevsellik ve performansı karşılamasını sağlamaktır. Bu nedenle, oluşturulan yazılımın kalitesi, Yazılım Geliştirme Yaşam Döngüsü'ndeki (SDLC) test seviyelerinin uygun şekilde uygulanmasından doğrudan etkilenir. Literatürde dört ana test düzeyi tartışılmaktadır: bireysel birimleri veya bileşenleri test etmek için birim testi, farklı bileşenler ve modüller arasındaki etkileşimleri ve arayüzleri test etmek için entegrasyon testi, sistemi bir bütün olarak test etmek için sistem testi ve test etmek için kullanıcı kabul testi. Yazılımın kullanıcının gereksinimlerine ve beklentilerine uygun olması. Buna göre bu araştırmanın temel amacı literatürde önerilen test seviyelerini ve yöntemlerini gözden geçirmek, bunların yazılım kalitesi üzerindeki etkilerini anlamak ve bu test seviyelerinin yazılım endüstrisinde nasıl kullanıldığını analiz etmektir. Bu doğrultuda Yazılım Sektöründen on yazılım uzmanıyla yarı yapılandırılmış bir görüşme gerçekleştirilmiştir. Mülakat soruları arasında yazılım kalitesine en çok hangi test seviyelerinin katkı sağladığı, mevcut projelerinde hangi test seviyelerinin kullanıldığı, projelerinde kullanılan test stratejisi, test stratejisi seçimini hangi faktörlerin etkilediği ve kimlerin kullandığı yer almaktadır. Testleri yapan geliştirici veya bağımsız bir ekibin bu konuda daha iyi olduğu görülmüştür. Elde edilen sonuçların, yazılım sektöründe test seviyelerinin uygulanması ve projelerin belirli alanları için test seviyelerinin yazılım kalitesi açısından ne kadar etkili olduğu konusunda fikir vermesi beklenmektedir.

Anahtar Kelimeler: Yazılım Kalitesi, Yazılım Testi, Test Seviyeleri, Yazılım Test Piramidi.

To My Family

ACKNOWLEDGMENTS

First and foremost, I would like to sincerely thank my supervisor, Asst. Prof. Dr. Damla Topallı, for her unwavering support, direction, and wisdom during my thesis research. I sincerely appreciate all of her expertise, excitement, and patience throughout this time. I'm really honored to be one of her pupils.

I would also express my appreciation to examination committee members; Asst. Prof. Dr. Cansu Çiğdem Ekin, and Asst. Prof. Dr. Gül Tokdemir for their valuable time, comments and suggestions.

Finally, I would like to express my appreciation to my cat and my husband for their endless love, patience and support. They have never ceased to inspire and believe in me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

E2E	-	End to End
UI	-	User Interface
API	-	Application Programming Interface

CHAPTER 1

INTRODUCTION

Software testing is a crucial aspect of the software development lifecycle, ensuring that the final product meets the desired quality and functionality. To achieve comprehensive testing, different levels of testing are employed, each focusing on specific aspects of the software. These testing levels provide a structured approach to verify and validate the software at various stages of development, helping to identify and rectify defects before the product reaches the end-users. Assuring that software projects produce goods of the desired quality is one of the most crucial concerns. As the project is being developed, specific testing levels are carried out to make sure of this.

According to the findings obtained from the literature review, it became clear that using the software testing pyramid is the ideal way to combine different test tiers in order to improve the quality of software projects. The Software Testing Pyramid is a conceptual framework that represents the ideal distribution of testing efforts across different levels, forming a pyramid shape. This model emphasizes the importance of a well-balanced testing strategy, with a higher concentration of tests at the lower levels and fewer at the higher levels. The goals of the software testing pyramid are to maintain the highest possible quality, identify faults early on, and minimize the costs and consequences of failures. By considering this viewpoint, it suggests which test level should be constructed starting at which moment.

In this work, we will explore how the software industry strategically employs various testing levels to guarantee the delivery of high-quality software products. From unit testing to system testing, and ultimately user acceptance testing, each level plays a crucial role in mitigating risks and enhancing the overall software development process. Understanding the nuances of these testing levels is essential for software professionals aiming to deliver software that not only meets but exceeds user expectations in an ever-evolving technological landscape.

In this thesis, it is aimed to understand how the software testing pyramid and testing levels are established in the literature, and how the software industry implements these levels. To this end, literature research was done to see whether any previous studies with a comparable design had been carried out. Additionally, surveys and interviews concerning the test levels were used to understand how they fit into the projects of the firms.

This thesis is organized as follows: in Chapter 1 the introduction section describes the context, significance, and objectives of the study. Chapter 2 discusses the related work and background of the study, offering an in-depth examination of the current trends related to software testing levels, implementation of testing methodologies and test pyramid. Chapter 3 outlines the methodology, research procedure and research questions of the thesis. In Chapter 4, the result of the study is presented to gather insights from the software industry by the questionnaire and interview results. Lastly, the thesis is finalized with the conclusion and discussion section, the findings, contributions, and limitations were explained.

CHAPTER 2

BACKGROUND OF THE STUDY

2.1 Software Testing

Software testing is the practice of assessing software with the goal of identifying errors in it. Software testing is a method used to assess a program's or product's capability or feature and determine whether it satisfies quality standards. Other software quality aspects, such as dependability, usability, integrity, security, capacity, efficiency, portability, maintainability, compatibility, etc., are also tested for in software testing.

We have been employing the same testing methods for many years. Some of them are not good engineering approaches, but rather created methods. Software testing can be expensive, but the cost of not testing it might be far higher. Software testing has specific objectives and guidelines that must be adhered to [1]. In order to create a high-quality and dependable product, software testing is crucial to the software development process. Software testing guarantees that the program operates as intended, complies with specifications, and is bug-free [2].

2.2 The Benefits of Software Testing

Software development is creating software in accordance with a set of specifications. To confirm and validate that the program has been constructed in accordance with these criteria, software testing is required. If not, we risk losing our client. Thus, we do testing to ensure that we give our client an appropriate software solution. Testing guarantees that the final product is what you intended to create. We investigate any issues or errors in the system that can render the client's software inoperable. This aids in keeping a system free from faults [1].

Software testing, which stands for the process of quality validation and verification of a software product, is an essential stage in the software development process. These days, this stage is even more important since software has to improve in quality because it is more complicated, mission- and safety-critical, and necessary for day-to-day operations [3].

Bug detection and correction: Through software testing, we may identify potential software flaws. These mistakes can have a negative impact on the user experience and the program's ability to work as intended. We can find and correct these mistakes thanks to the testing process.

Quality Assurance: Software testing is a tool used in quality assurance to raise the level of software. A quality piece of software will be dependable, effective, and able to satisfy user needs. An audit to confirm that the software complies with quality requirements is provided through the testing process.

Cost and Time Saving: Software testing helps prevent future issues by assisting in the early detection of bugs, which saves both money and time. Early error detection saves time and money by lowering the cost of rectification.

Trust and consumer Satisfaction: Proper software operation and the execution of anticipated functions provide consumer confidence. Customers desire a trustworthy and faultless software experience. Software testing is crucial to establishing this confidence and elevating client satisfaction.

2.2.3 The Basic Principles of Software Testing

The basic principles of software testing are foundational concepts that guide the testing process and contribute to the effectiveness of identifying defects and ensuring the quality of software products. These principles help testers, developers, and quality assurance professionals in designing and executing robust testing strategies.
These principles are given in below:

Accessibility: The testing team should have simple access to the software under test. This improves the effectiveness and efficiency of the testing procedure.

Complete Coverage: All functionalities and scenarios should be covered during software testing. Every feature and scenario should be tested to make sure they are functioning properly.

Independence: The software development process and the testing process should operate independently. The chance to find faults objectively is provided by a team of impartial testers.

Repetition: Test procedures and scenarios ought to be repeatable. This makes it possible to recursively check the results' accuracy and find errors.

Documentation: Test cases and outcomes must be documented as part of the testing process. This enables tests to be repeated and outcomes to be compared for potential changes in the future.

Software testing is a crucial stage in the creation of software that raises the level of quality. Error detection, quality control, cost and time savings, trust and customer happiness are just a few of its numerous benefits. A successful product is largely dependent on software testing conducted in accordance with its fundamental principles.

2.2.4 Software Testing Processes

Testing is the process of determining whether or not a certain system satisfies the requirements that were first stated. It is mostly a process that includes validation and verification to see if the created system satisfies the user-specified requirements. As a result, the outcome of this activity differs from what was anticipated. Software testing is the process of examining developed systems or software to identify defects, mistakes, or missing requirements. Thus, this inquiry gives the relevant parties precise information regarding the product's quality. Another way to think of software testing as a risk-based activity is. During the software testing process, it is crucial for testers to know how to reduce a huge number of tests into a manageable set and make informed decisions about which risks should be tested and which should not [4].

In the software development process, a testing process should be followed to ensure the quality of the software and detect errors at an early stage. The testing process aims to verify that the software works correctly, produces expected results, and meets user requirements.

Steps of the Testing Process:
Requirements Analysis: The testing process starts with understanding user requirements. User requirements are analyzed to determine what functions the software should perform and expectations.

Test Planning: The test plan determines how the testing process will be run and which tests will be performed. The test plan specifies the test objectives, scope, strategies, and resources.

Creation of Test Cases: Test cases are created to test different functions and scenarios of the software. Each scenario simulates a specific use case and defines expected results.

Preparation of the Test Environment: The test environment includes the hardware and software components to test the software. The test environment should simulate the real production environment and ensure that the software works correctly.

Test Execution: Test cases are executed in the test environment. At this stage, it is checked whether the software produces the expected results, whether the errors are detected and whether it works correctly.

Error Tracking and Management: Errors detected during the test are recorded and tracked. Bug reports are generated; errors are prioritized and forwarded to the development team for correction.

Test Reporting and Evaluation: Test results and error reports are compiled and evaluated in the form of test reports. Test reports provide information about the quality and test coverage of the software.

Retest and Regression Tests: After corrections are made, it is determined that the errors are corrected correctly.

2.3 Software Testing Pyramids and Tiers

Software testing is a crucial step in making sure a piece of software works correctly and is of high quality. How to organize and prioritize tests is a crucial topic in software testing [5]. In Figure 2.1, the software test pyramid shows how the tests are organized and balanced at various stages [6].

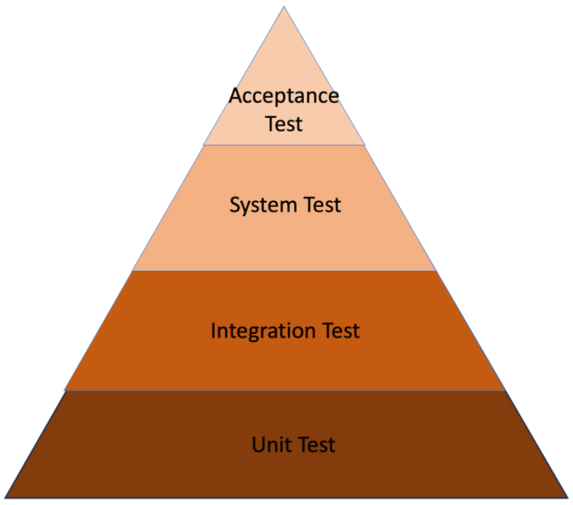


Figure 2.1 Software Testing Pyramid

A conceptual model known as the "test pyramid" explains how quality checks can be set up to guarantee that every system component is covered at every stage. The idea was progressively adopted into software engineering after it was initially developed to assist aerospace engineers in scheduling tests to ascertain how material changes affect system integrity. These days, the test pyramid is usually used to show that most tests should be run at the lowest level, which is the unit test; fewer integration tests and even fewer acceptance tests—which are the most costly to create and the slowest to run—should be conducted.

While the integrity of the underlying data, models, and pipelines is becoming more and more important for acceptance tests and integration tests, software development and data management organizations have historically been divided, and quality assurance procedures are less developed in the data operations space than they are in the software industry [7].

Unit Tests: Unit tests are used to test the smallest functional components of the software at the base of the testing pyramid. Unit tests typically test quickly-working code-level features. These tests are performed to make sure that every part of the software is operating properly and delivering the desired outcomes. Developers can test every component of software and find flaws early on thanks to unit tests.

Integration Tests: Component tests, which are placed atop unit tests and enable the assembly and testing of several unit components. The purpose of this layer is to test how the units interact and interface with one another. Unit tests can contain flaws that haven't yet been found thanks to component tests. With the use of these tests, the software's component integration and collaboration are specifically examined.

System Tests: The user experience and interface of the product are tested using user interface (UI) tests, which are at the second level of the software testing pyramid. These evaluations look at the software's functionality, usability, and user interactions. Utilizing automation techniques, user interface tests are typically carried out to determine whether software is compatible with various platforms (web, mobile, and desktop).

Acceptance Tests: A quality assurance procedure called acceptance testing establishes the extent to which a program satisfies end users' needs. Acceptance testing may be conducted as end-user, field, application, or beta testing, depending on the company.

The software test pyramid tries to maximize the quantity and cost of tests while improving the accuracy and quality of the product. Lower level unit and component testing have the benefit of spotting errors early on and offering quick feedback. The overall functioning and user experience are tested at higher levels of service and user interface testing. The software testing pyramid offers a useful framework for organizing and prioritizing the software testing process. In order to enhance quality and save expenses, this paradigm is frequently applied during the software development process [7].

2.4 Software Test Levels

2.4.1 Unit Test

Small, automatically executing unit tests are possible thanks to the unit testing frameworks that are available for practically every programming language these days. Unit testing is now considered standard procedure and is frequently required by development processes. Software quality is still a problem, though. Thus, academics in software engineering contend that testing automation needs to be advanced to the point where unit tests may be generated automatically [2].

Verifying that your software is working correctly, performs the expected functions and doesn't have any bugs during development will be important. In this validation process, software unit tests are very important. The most basic components in a software suite such as functions, methods or classes shall be tested separately using Software Unit Tests. Software unit testing is an automated process to check for the smallest components of software that may be tested. The unit tests are generally written by programmers so they can verify the functions and methods of software. Unit tests check that each unit component of the code works correctly, produces the expected results, and is free of bugs. It will enhance the quality of software and enable bugs to be detected more quickly.

The importance of software unit testing is explained in below. This type of testing is mostly suitable to test each individual unit of the software separately, to identify the bugs in the code.

Bug detection and repair: Software unit tests allow you to detect possible bugs in your code. Unit testing shall verify that the expected results are produced by each component, as well as catch any errors. This is how errors can be detected and corrected at an earlier stage.

Increase code coverage and confidence: Unit testing increases code coverage. For more thorough control of the code, it is possible to test each component individually. Unit testing also ensures that, if you change your code, it does not break the current functionality. This makes it more likely that the code will be trusted.

Documentation and Preparedness for Future Changes: Unit tests help document your code. Tests clearly show what the code should do and make the code easier to understand. Also, unit tests ensure that code is expected in future changes.

Best Practices and Approaches:
Creating Small and Isolated Test Cases: Create small and custom test cases to test each component in isolation. Test cases check whether the component produces the expected results and catches errors.

Automation: Automating software unit tests enables a faster and repeatable testing process. Automating test cases using automated testing tools and automatically evaluating results increases efficiency.

Frequent and Continuous Testing: Perform software unit tests frequently and continuously. You can quickly detect and fix bugs by running unit tests after each change.

Error Reporting and Management: Record and track errors detected during testing. Generate bug reports, prioritize bugs, and forward them to the development team for fixes.
Software unit tests are an important test method for testing the smallest units of software in isolation and detecting bugs at an early stage. These tests improve the quality of the software, fix bugs and ensure the reliability of the code. You can make your software more robust and reliable by regularly performing software unit tests.

The "lowest" level of testing is unit testing, which is intended to evaluate the units generated during the implementation phase. Sometimes, such when developing general-purpose library modules, unit testing is carried out without the software application it encapsulates being known. Similar to module testing, unit testing is typically the duty of the programmer in software development organizations [8].

Developers typically start by writing unit tests before moving on to coding software units to ensure that the customer receives a reliable product with which to conduct acceptance testing. Because they were created to make the software fail a requirement, unit tests are failure tests. In a paradoxical way, developers are forced to create software that fails in order to test the testing. Developers continue writing software that passes the unit tests after test harnesses are put in place.

The purpose of unit tests is to make the software fail. You can only start fixing the code such that it passes the tests by making sure your tests catch errors. The testing process—and a developer's confidence—depend on your unit tests' ability to detect problems. The developer can now experiment with various implementations while being confident that any errors will be caught by the unit tests.

Any code modifications should enhance the program rather than add bugs. In order to improve and streamline the code base, refactoring activities are also supported by the continuous testing idea. Continuous testing also produces confidence, the previously mentioned intangible benefit. Because you continually validate the code base with unit tests, the programming team feels more confident about it. Additionally, knowing that the code base consistently passes unit tests boosts the confidence of your clients in their investment [9].

The rise of object-oriented programming has changed how software testers are approached by programmers. It is known that object-oriented programming, which is primarily bottom-up, favors a testing approach that emphasizes classes. A unit test runs a "unit" of code in isolation and contrasts the outcomes with what was anticipated. The unit in Java is typically a class. Unit tests call one or more class methods to generate observable outcomes that are automatically checked [10].

2.4.2 Integration Test

In the software development process, it is important that the different components work harmoniously and function correctly together. Software integration testing is a testing phase used to test the integration and collaboration of different components. In this section, we'll cover the importance, benefits, and best practices of software integration testing. Software integration tests are automated or manual tests that are used to test collaboration and compatibility of different software components by combining them. These tests check the components' interfaces, data communication, database interactions, and other integration points. Integration tests aim to verify that components work together correctly and produce expected results [11].

The testing that is done after every module has been assembled into a functional program is known as integration testing. Instead of testing at the statement level like in unit testing, testing is done at the module level. The relationships between modules and their interfaces are the focus of integration testing [12].

Unexpected interactions between system components are a common cause of software and system errors. When a system has a lot of replaceable network components for each element, the risk goes up. To lessen the chance of interaction issues, a maker of these system components would want to test as many alternative system configurations as feasible. However, there are an exponentially increasing number of possible system configurations [13].

Importance of Software Integration Tests:

Collaboration and Interface Check: Software integration tests ensure that different components collaborate when they come together and interfaces work correctly. These tests check that the components work harmoniously with each other and that the data or information exchange is error-free.

Debugging and Catching Problems at an Early Stage: Integration tests aim to detect errors between components. It detects errors that may occur at the points where different components come together and allows these errors to be corrected at an early stage. This helps prevent bigger problems.

Performance and Reliability Check: Integration tests check whether components are performing together and working reliably. It helps to detect performance problems and error conditions that may occur with the combination of components.

Software Quality and Customer Satisfaction: Integration tests increase the quality of the software and ensure customer satisfaction. Compatible and integrated software provides the user with a seamless experience and demonstrates that the functionality is performed correctly.

Best Practices and Approaches:

Good Planning and Design: Good planning and design is essential to perform integration tests effectively. Which components will be tested, determining integration points and creating test scenarios are important steps.

Modular and Standalone Tests: It is important to use modular and independent tests, as integration tests test the interoperability of components. Testing each component in isolation makes it easier to identify and resolve issues.

Automation: Automating integration tests enables a faster and repeatable testing process. Automating test cases using automated testing tools and automatically evaluating results increases efficiency.

Error Tracking and Monitoring: It is important to track and monitor the errors detected in integration tests. Fixing, retesting, and tracking bugs improve the quality of software.

Software integration tests are important to ensure that different components of the software work harmoniously and function correctly. These tests improve the detection of bugs, the early resolution of problems and the quality of the software. Planning, designing, and automating integration testing leads to a more effective and efficient testing process.

2.4.3 System Test

Software and hardware systems are tested as a whole, integrated unit to determine whether or not they meet the requirements that have been set forth. Since system testing is a type of black box testing, it shouldn't be necessary to understand the logic or inner workings of the code. The main goal of system testing, which is essentially a collection of several tests, is to thoroughly test the computer-based system.

Even though each test serves a distinct objective, they all aim to confirm that system components have been correctly integrated and are carrying out their assigned tasks [1].

Software development projects are becoming increasingly complex and large. The successful completion of these projects depends on the complete completion of important steps such as software system testing. Software system testing is a critical phase of evaluating how the software works as an integrated system and checking whether the software meets business requirements.
When choosing testing procedures, the two most crucial factors are effectiveness and economy. Economics suggests that the test itself should use the least amount of time and resources possible, even yet efficacy requires the test to be able to reveal the greatest number of faults present in the software [14].

Finding flaws in the way the system to be tested functions is the goal of system testing. In order to realize a desired function for the system user, the intended functional behavior is determined by the functional needs of the system. When a system's behavior deviates from its functional requirements, mistakes are present [15].

Software System Testing

Software system testing is a process in which all components of a software application are brought together and tested. This testing focuses on checking whether different components of the software work together smoothly and meet specific business requirements. System testing also includes evaluating the performance, security, and overall stability of the software.

Software system testing is a type of testing performed to evaluate the functionality of a software application and verify that certain functions work as intended. This testing focuses on determining whether the software meets user expectations. Its main purpose is to test the functionality, usability and performance of the software and to ensure that it can be safely presented to users at the end of the development process.

Advantages of Software System Testing

Evaluating Integrated Functionality: System testing evaluated how different components of the software are brought together and how they worked as an integrated system. This is critical to ensure the different components work in harmony.

Checking Business Requirements: System testing is used to check whether the software meets the business requirements or not. This allows verification of the functionality and usability of the software.

Meeting User Expectations: Software system testing is important to verify whether the software meets users' expectations. This is critical to improving user experience and increasing customer satisfaction.

Improving Performance: System testing is used to evaluate the performance of the software. This helps optimize the response time, speed, and scalability of the software.

Improving Security: System testing is used to test the security of software and detect vulnerabilities. This ensures the protection of user data and business processes.

Software System Testing Process

Software system testing usually includes the preparation of the test plan, the test environment, execution of the test scenarios, evaluation of the test results, fixing the bugs found and re-testing and finally validation and the delivery of the software to the users. These processes are described in detail below.

Preparation of Test Plan: The first step is to prepare a test plan that determines the scope, objectives and plan of system testing. This plan should include test scenarios and test data.

Preparation of the Test Environment: A suitable test environment is created for system testing. This includes an environment that enables the integration of different components.

Execution of Test Scenarios: The prepared test scenarios are applied to the software system. These scenarios simulate different business processes and use cases.

Evaluation of Results: Test results are used to evaluate the performance and compatibility of the software. If any errors or deficiencies are detected, feedback is given to the development team.

Bug Fixing and Retesting: If bugs are found, these bugs are fixed, and the software is retested. This process can be repeated to verify whether the errors have been fixed.

Validation and Delivery: Finally, when the software is confirmed to be usable and the tests are successful, the software is delivered to users.

Software system testing is an essential part of the successful completion of software projects. This testing is used to evaluate the software's integrated functionality, business requirements, performance and security, as well as detect and fix errors. Good software system testing contributes to the successful completion of large projects and increased user satisfaction.

Software systems are tested as a whole, integrated unit to see if they meet the criteria as stated. This is known as system testing. Since system testing is a type of black box testing, it shouldn't be necessary to understand the inside workings of the code or logic. The goal of system testing, which is essentially a collection of several tests, is to thoroughly test the computer-based system. Even though each test serves a distinct objective, they all aim to confirm that system components have been correctly integrated and are carrying out their assigned tasks [1].

2.4.4 Acceptance Test

A number of tests are carried out during the software development process to ensure that the program satisfies user requirements and fulfills set standards. "Acceptance testing" is one of these testing procedures. Prior to the program being made available to customers or end users, a testing phase known as software acceptance testing is conducted [16].

Software Acceptance Testing

Software acceptance testing is a testing process performed to verify that the software works in accordance with the specified requirements, meets user needs and is generally functional. This testing is usually done by the customer or end user and confirms that the software is ready for use.

The interests of the client are represented by acceptance tests. The consumer can feel confident that the application has the necessary functionality and functions properly thanks to the acceptance tests. The project is finished in theory when every acceptance test is passed [17].

The user does manual testing as part of user acceptance testing. User acceptance testing is not usually mechanized. If not, it would be regarded as an automated test case for verifying the functionality of the program. However, we might think about automating some tests if users are too busy to test after every build or if our testing team is understaffed [18].

The Importance of Software Acceptance Testing

User Satisfaction: Software Acceptance Testing assesses how well the program satisfies user requirements. User satisfaction rises as a result.

Detection of Bugs: Acceptance testing offers the chance to find software bugs. Resolving these issues increases the software's dependability.

Reliability: Determines the degree of reliability of the software. Users favor using dependable software.

Software Acceptance Testing Process

Determining criteria: Knowing the criteria that the program is built upon is the first step in the acceptance testing process. The developer and the client should decide on these specifications.

Constructing Test Scenarios: In order to conduct acceptance testing, test scenarios must be constructed. These scenarios ought to encompass various software use cases and functionalities.

Test Execution: The requirements must be followed when implementing the test scenarios. To determine if the software performs as anticipated, each situation is examined.

Error Analysis and Reporting: An error report is created for each error that is discovered during testing. This report provides an opportunity for developers to address bugs.

Customer Approval: After being shown the test results, the customer chooses whether or not to accept the software.

Best Practices of Software Acceptance Testing

Early Communication: Accurate requirement understanding and efficient test case production are ensured by early communication with the customer.

Automation: By automating repeatable test cases, automation technologies can expedite the testing process and contribute to the delivery of more consistent results.

Examining Actual Users: Including actual users in the software's evaluation and feedback loop helps improve the acceptability testing procedure.

To make sure the program fulfills user expectations, software acceptance testing is an essential step in the software development process. Software quality can be raised and customer satisfaction can be guaranteed with precise and thorough acceptance testing [19].

2.5 Related Work

Previous studies on this subject in the literature have been examined. Although there are not many studies on this subject, the following study has been examined as a related work in the field. Mike Cohn's testing pyramid shown in Figure 2.1 was adjusted for dispersed information processing systems testing [7]. The adjusted version given in [7] expanded testing capabilities and applies distributed system characteristics.

There are now ideas for additional uses for the mechanisms included in the revised Mike Cohn's pyramid. In their study, the requirements provided were needed to ensure that the distributed computer system will always be able to supply the services of the distributed information processing system and that request packets will always be sent at a certain rate. They have developed a software testing paradigm for distributed systems that is based on the independent software component deployment [7]. This allows for a reduction in the number of bulk tests while simultaneously boosting testing efficiency [20], based on an altered version of Mike Cohn's pyramid given in Figure 2.2, described by [21].

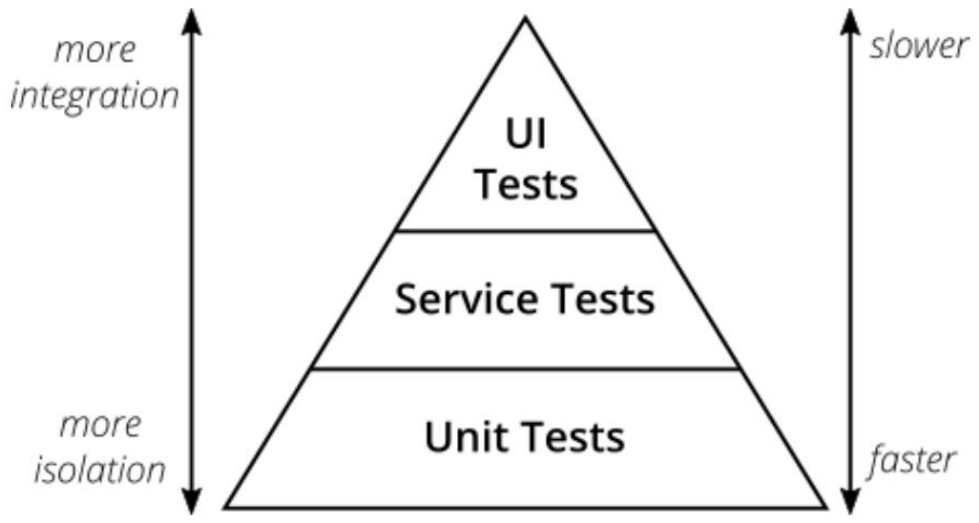


Figure 2.2 Cohn’s Test Pyramid as described by Vocke [21]

In this instance, the distributed systems architecture is taken into consideration when replacing end-to-end and service testing. It is obvious that end-to-end testing is an ineffective method for distributed systems architecture since it necessitates a guarantee that modifications won't interfere with the operation of other subsystems when a new subsystem is deployed in live applications. Using so-called "contracts" that are based on requests from the subsystem is one method to accomplish this without utilizing actual subsystems. A test code that operates in vending mode is called a contract.

In actuality, the pyramid is usually flipped, with the more costly and fragile UI tests occurring far more frequently. As seen in Figure 2.3, this "ice cream cone" model shows that there are a lot more automated UI tests at the top, a lot more automated unit tests at the bottom, and a lot more manual tests at the top. This means that a larger portion of the testing burden is placed on the testers or QA team because developers' unit test coverage is insufficient. The ice cream cone is frequently the outcome of lower management allocating less resources for unit testing in favor of end-to-end testing that shows the product's functioning [6].

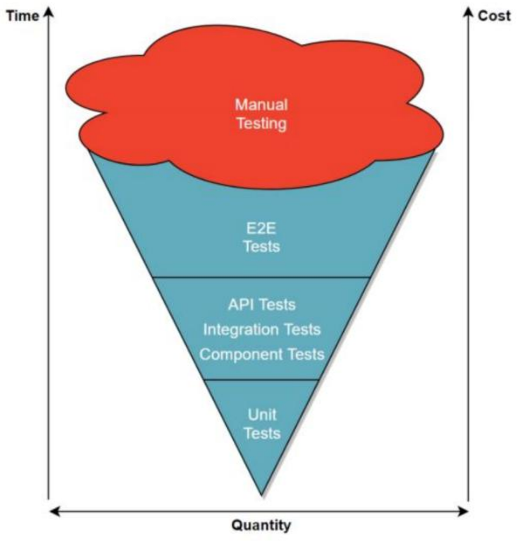


Figure 2.3 Testing in practice; the “inverted test pyramid” as described by Hartikainen [6]

The test strategy is centered on the location of the tests, not the types of tests that are designed, according to the suggested pyramid (see Figure 2.4). It is assumed that functional tests will be augmented by unit testing at every level of the pyramid. It acknowledges that, given the growing complexity of data-driven systems, uncertainty is likely to exist in both scenarios and does not discriminate between a user who is a person and a user who is a machine.

Additionally, it eliminates the possibility that unit tests can stand alone and do not require contract or functional testing at all levels because they should serve as the cornerstone for system behavior, performance, and stability. In reality, functional testing is becoming more and more necessary [7].

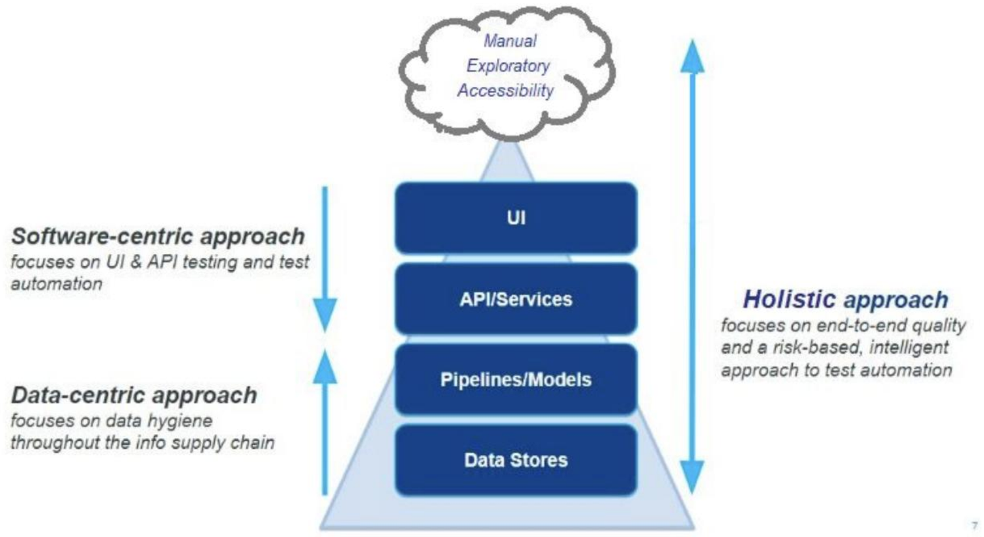

Manual
Exploratory
Accessibility
UI
API/Services
Pipelines/Models
Data Stores
Software-centric approach
focuses on UI & API testing and test automation
Data-centric approach
focuses on data hygiene throughout the info supply chain
Holistic approach
focuses on end-to-end quality and a risk-based, intelligent approach to test automation

Figure 2.4 A holistic test pyramid for data and traditional software testing [7]

The development of a testing methodology, the computation of the testing method selection based on the modified Mike Cohn's pyramid, and the creation of procedures for carrying out upper levels of testing are required in order to test the full distributed system functioning [20].

By combining both data and traditional software testing in a holistic test pyramid , organizations can ensure comprehensive test coverage that addresses the complexities of both software-centric and data-centric approaches. The software centric approach focuses on both the underlying functionality of software components (API testing) and the end-user experience (UI testing). A data-centric approach in software testing places a primary emphasis on the quality, integrity, and hygiene of the data used in testing processes. Combining them both, this holistic approach helps in building robust, reliable software systems and intelligent approach to test automation that effectively handle data processing and meet user expectations.

CHAPTER 3

METHODOLOGY

In order to provide a detailed understanding of how testing levels are utilized in software industry, a mixed-methods approach is used in this study by combining the qualitative and quantitative techniques. The research questions and the detailed research procedure are given under the following sections.

3.1 Research Questions

In this study, we aimed to investigate how the use of software testing levels varies across sectors. To investigate how the use of these levels varies and the general approach across sectors, we set the following research questions.

RQ1: What kinds of test levels are described in literature?
RQ2: What is the suggestion to apply these test levels?
RQ3: How are these test levels implemented in the software industry?

3.2 Research Procedure

In this research, two different methods were followed to investigate the research questions. Firstly, an online survey prepared by using google forms and conducted from May to Dec 2023. It was sent to people in different sectors for preliminary research. The purpose of this survey is to understand people's general knowledge about test levels and to analyze their experiences with these test levels. Secondly, interviews were conducted both face to face and online during November 2023 with people experienced in testing in different sectors. Out of 10 participants, 4 participants interviewed face to face and for remains, it was online. The online sessions are preferred for the participants living out of Ankara.

In these interviews, individuals were asked to personally evaluate which test level they had mastered and how well they had contributed to the quality of these test levels. In addition, their experiences were asked about how they used these levels in their current and previous projects and what they took into consideration when determining the test strategy. In this thesis work, I have prepared both the semi-structured interview and survey questions based on our research aim. Then for validating the survey and interview questions, three domain experts in the field of software engineering reviewed and provide feedback on the relevancy and appropriateness of the questions.

3.3 Participants

35 people participated in the survey, which was conducted to understand people's level of knowledge about test levels and to understand the test strategies that companies create within the scope of their projects and which test levels they develop by taking into account which criteria. While 40 percent of participants are female, 60 percent of them are male. The ages of the participants ranged between 18 and 55 years. Details about the participant information are given in Figure 3.1.

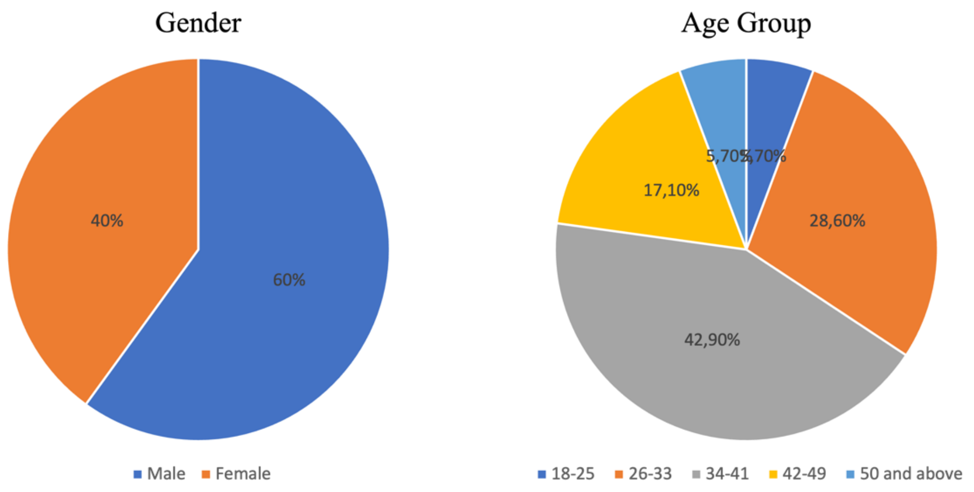


Figure 3.1 Gender and Age Distribution of Surve Participants

These participants consist of people who develop or implement at least one test level in their current projects. Average work experience varies between 5 and 18 years. While 50 percent of the people participating in the survey graduated from computer engineering, 30 percent from electrical and electronics engineering, and 10 percent from industrial engineering, the remaining 10 percent graduated from different departments. Again, 40 percent of the participants work in the defense industry, 30 percent work in energy systems and the other 30 percent work in different fields. These information is represented in Figure 3.2 and 3.3, respectively.

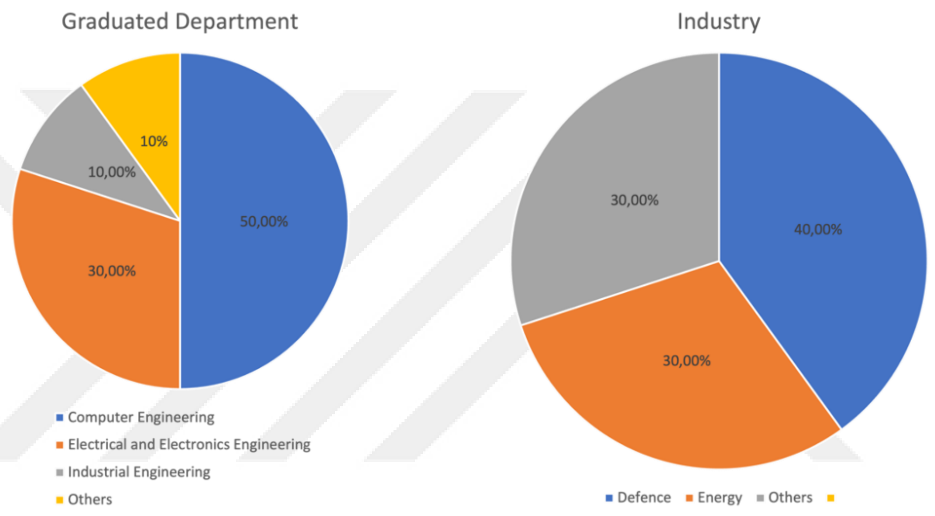


Figure 3.2 Graduated Departments and Industries of Survey Participants

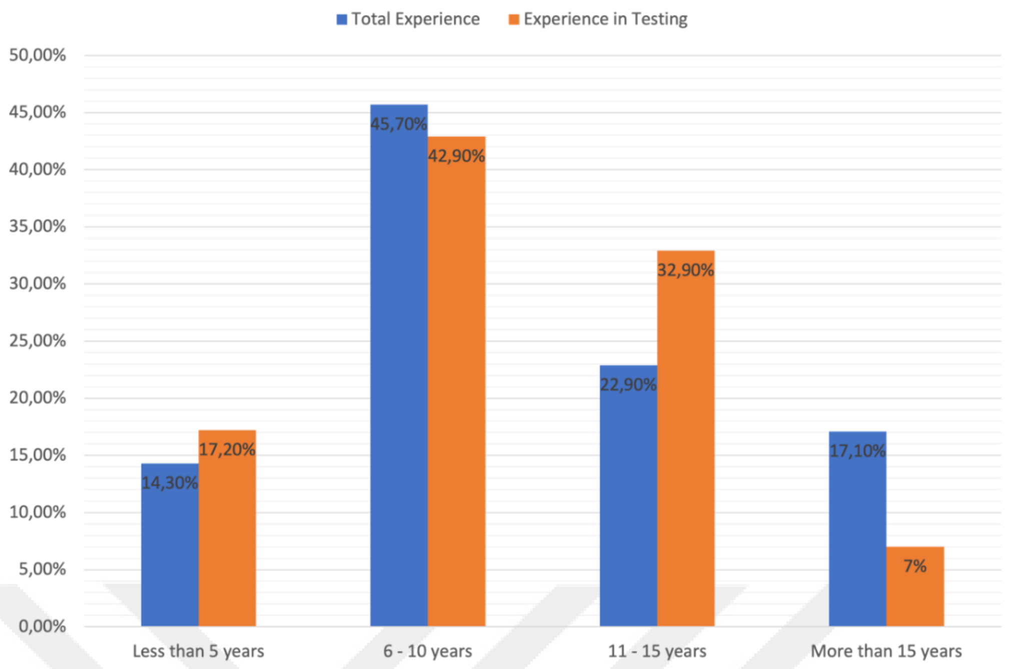
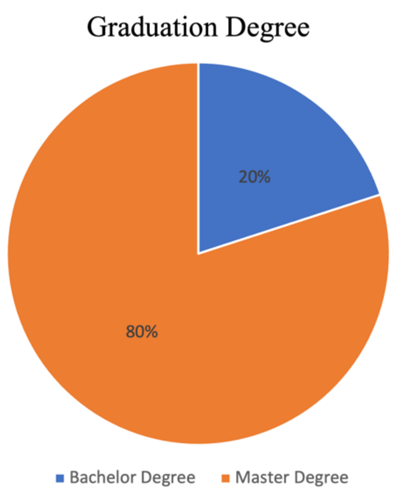


Figure 3.3 Experiences of Survey Participants

Details about the graduation levels of the interview participants is given in Figure 3.4. Among those participants, 40% of the participants graduated from the department of computer engineering, %20 from the department of electrical and electronics engineering, 20% from the department of Mathematics and remaining 10% from the department of Physics. Majortity of the participants (80%) obtained a master degree.



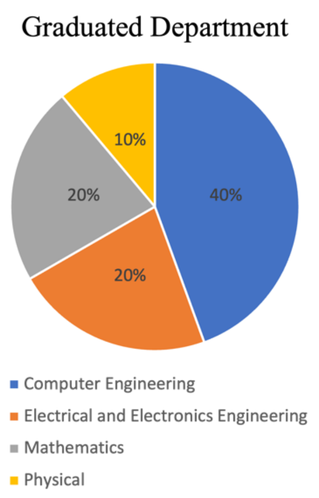


Figure 3.4 Graduation Degree and Departments of Interview Participants

As seen in Table 3.1, interview participants were selected as 10 people with software testing experience ranging from at least 8 years to 45 years.

Table 3.1 Experiences of Interview Participants

Percentage of the interview participants	Experience in software testing (year)
40%	8 - 10
40%	11 -15
20%	15+

These participants have experience in the fields of defense industry, finance, energy, telecom and management automation. The industries that the participants are working is given in Figure 3.5.

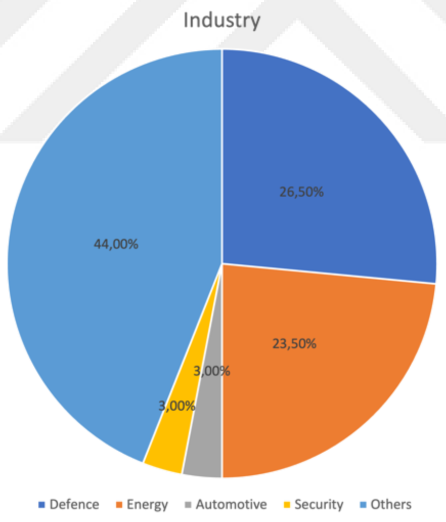


Figure 3.5 Industries of Interview Participants

Both survey and interview participants were selected considering their experience in the field of software testing. For this selection, Linkedln profiles were examined and suitable people were contacted.

CHAPTER 4

RESULTS

In this chapter, preliminary analysis from questionnare is given according to questions. In addition to that, the results of the interviews were presented in three sessions based on the answers to each of the research questions: firstly the results based on the test levels that are depicted in the existing literature is summarized and the findings on the diverse approoaches and methodologies employed by the software professionals were presented. Then, the results on the recommendations that are suggested for applying these test levels in the software industry practices were explained. Lastly, the methodologies and strategies adopted by software development professionals to implement these test levels effectively were explained based on the survey and interview results.

4.1 Preliminary Analysis: Results of the Questionnare

In the survey conducted for the preliminary research, it was aimed to understand people's experience with each test level and they were asked to evaluate the effectiveness of these test levels. Graphs of the collected results and the implications of these results are provided below. Participants were asked to indicate their experience with each test level on a scale from 1 to 5 (from Novice to Expert). Person distribution according to the answers given to the question is shown in Figure 4.1. It has been observed that each person chooses Competent and above (3 and above) at more than one test level. This shows us that these people are capable of evaluating the effectiveness of testing levels.

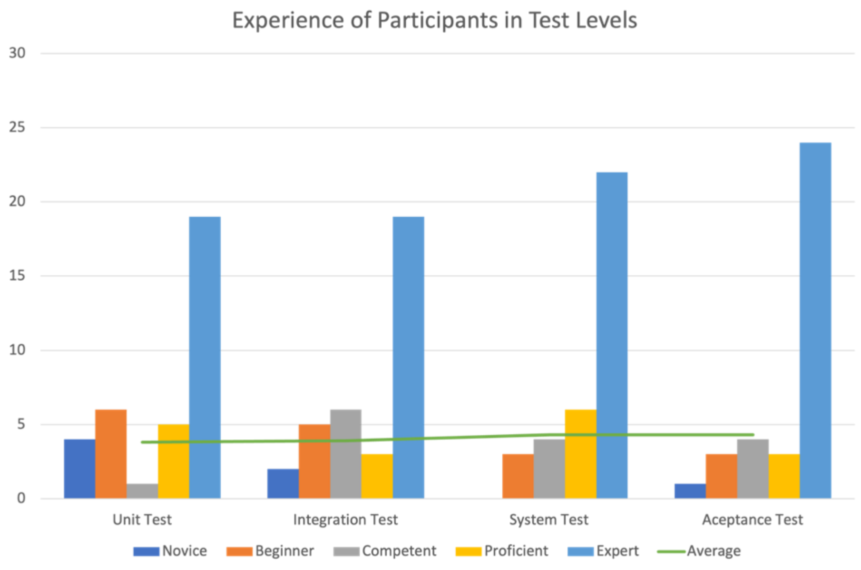


Figure 4.1 Experience of Participants in Test Levels

Participants were asked which testing levels they thought were most effective in terms of their impact on software quality, in line with their experiences. Based on the response from the participants (see Figure 4.2), it is seen that the most effective testing level in terms of software quality is unit testing, followed by integration testing and system testing. This result is exactly the same as the written test pyramid.

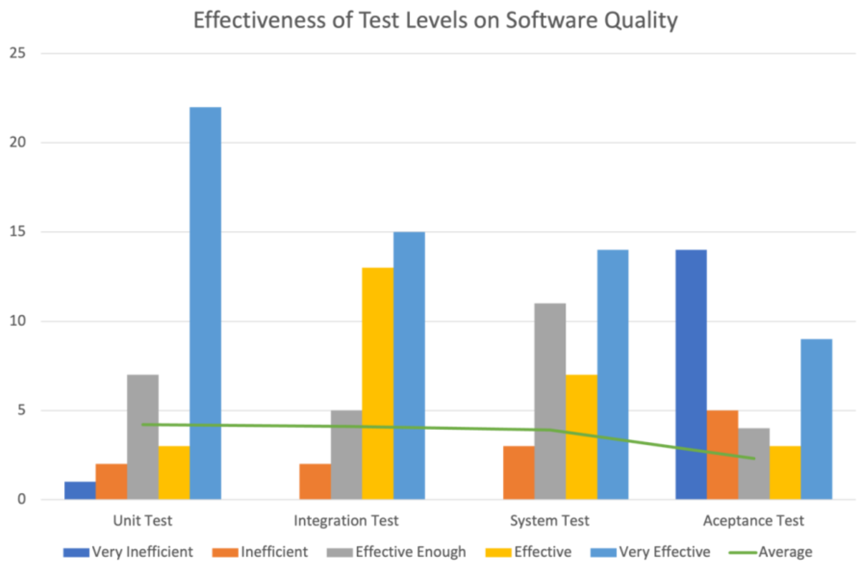


Figure 4.2 Effectiveness of Test Levels on Software Quality

Therefore, it can be concluded that determining a testing strategy in accordance with the software testing pyramid will be the most effective method in terms of software quality. In addition, participants were asked which test level they used and how often. As seen in Figure 4.3, the participants use unit tests most frequently in their current projects. It has been observed that it is followed by integration testing and system testing.

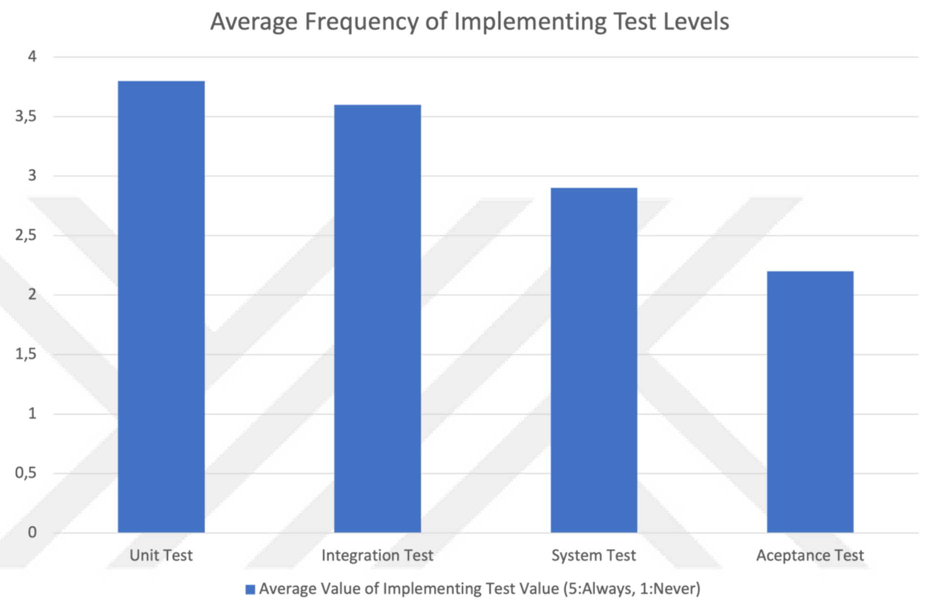


Figure 4.3 Average Frequency of Implementing Test Levels

Considering all these data, it can be thought that the survey participants generally adopted the software testing pyramid and a method suitable for its purpose.

4.2 Results of the Interview

In this section the results of the interview with our participant are discussed to answer the research questions in three sessions as given below.

4.2.1 What kind of test levels are described in the literature?

As seen in Figure 4.4, within the scope of the Software Testing Pyramid, four main testing levels are defined. These are stated as unit test, integration test, system test and acceptance test, from the lowest level to the highest level [4]. Each test level and their detailed quotes are discussed in detail in Chapter 2.

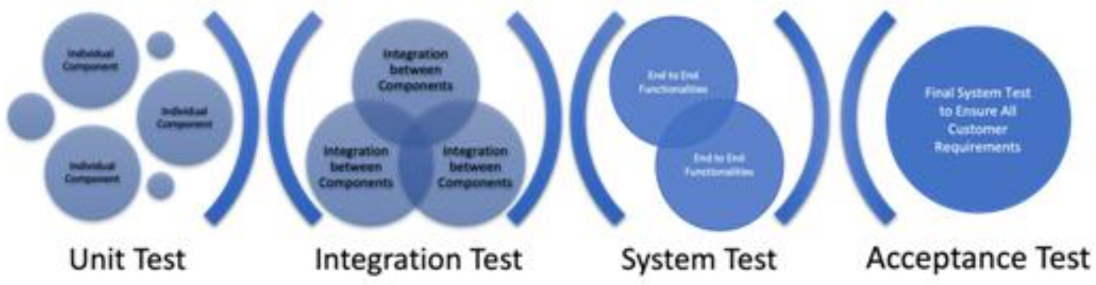

Individual Component
Individual Component
Individual Component
Integration between Components
Integration between Components
Integration between Components
End to End Functionalities
End to End Functionalities
Final System Test to Ensure All Customer Requirements
Unit Test
Integration Test
System Test
Acceptance Test

Figure 4.4 Software Test Levels and Their Functionalities

Each level of testing has a purpose that contributes to the quality of the developed product. These can be briefly summarized as follows. While unit testing aims to test each individual component within the software in isolation, integration testing checks whether the integration between these components works correctly. System test checks that each functionality works correctly end to end(e2e). Acceptance testing, on the other hand, ensures that customer requirements work correctly in the environment defined by the customer [1].

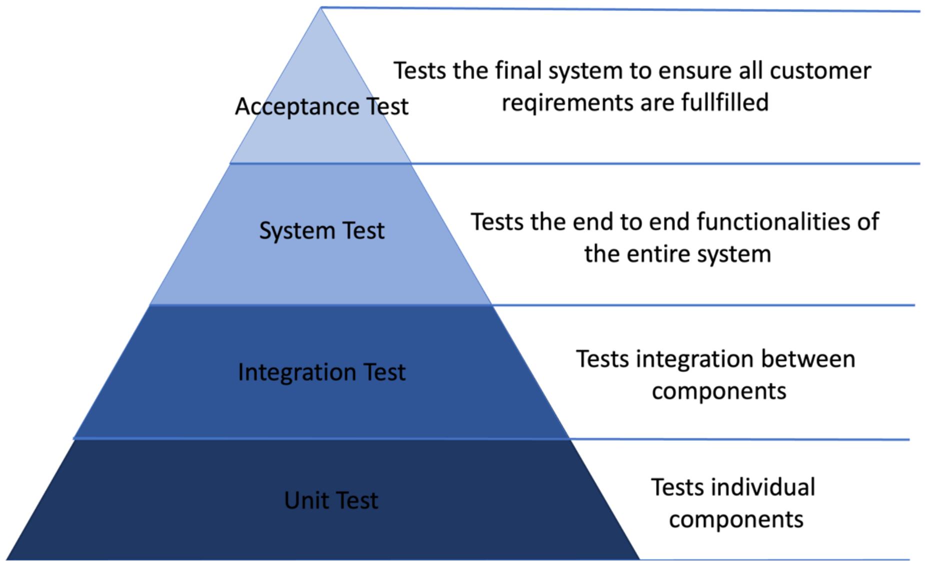

Acceptance Test
Tests the final system to ensure all customer reqirements are fullfilled
System Test
Tests the end to end functionalities of the entire system
Integration Test
Tests integration between components
Unit Test
Tests individual components

Figure 4.5 Software Test Pyramids and The Goals of the Each Levels

4.2.2 What is suggested to apply these test levels?

Within the scope of this research question, interview participants' opinions were taken on how to use the test levels in the most effective way. Participants were expected to answer this question considering their current work experience.

During the research, it was seen that the test levels that play the main role in ensuring software quality and offering a product that can meet the customer's expectations are unit test, component test, integration test, functional test and system/acceptance test. Considering the contribution of each level to the writing quality, it has been observed that the use of all of these tests in a certain combination gives the most effective results. Participant P8 explained that “Each different level finds different types of defects. You need them all to produce a high quality software product”.

How and in what quantity these test levels should be used is explained in detail in the concept called software testing pyramid. This concept describes a path to minimizing software maintenance costs, early detection of errors, and ultimately ensuring customer satisfaction, starting from the quality of the software code. Participant P4 said that “I think the need for testing levels may vary by industry. Considering the effort and benefit spent, the defense industry focuses more on unit and integration tests. However, I think that in sectors and systems with fewer users or where error effects are not very high, more customer requirement tests, that is, functional and system tests, will be more effective.”. For this reason, it is stated that the number of unit tests should be proportionally more than at other levels. Afterwards, it is stated that component, integration and at least system testing should be prepared. This method not only ensures product quality but also helps to solve any errors found at the least cost.

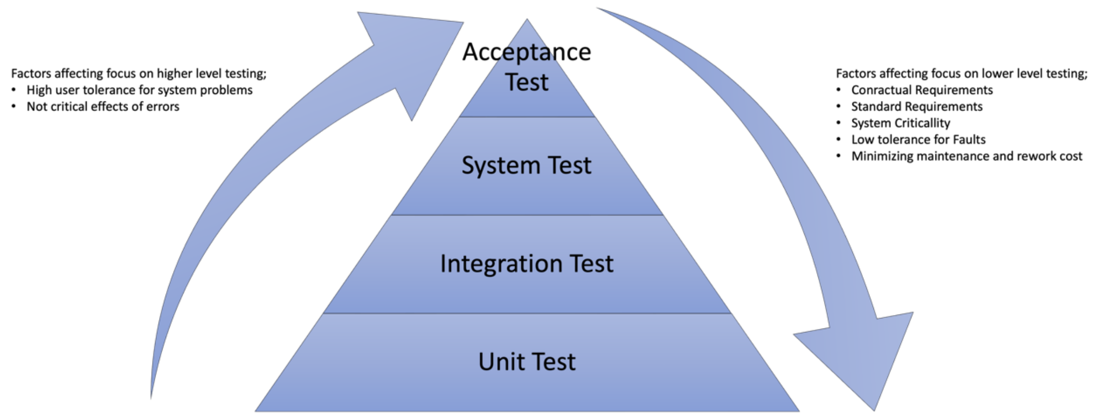

Factors affecting focus on higher level testing;
• High user tolerance for system problems
• Not critical effects of errors
Acceptance Test
System Test
Integration Test
Unit Test
Factors affecting focus on lower level testing;
• Conractual Requirements
• Standard Requirements
• System Criticallity
• Low tolerance for Faults
• Minimizing maintenance and rework cost

Figure 4.6 Factors to Decide Most Effective Testing Strategy

The factors affecting the focus on higher and lower level testing strategies are depicted in Figure 4.6. The importance of unit and integration testing was emphasized in facilitating the maintenance of the product and eliminating any errors found at an early stage. On the other hand, it has been stated that one of the best ways to ensure that customer demands are met is to focus on system and acceptance level tests. The importance of correctly realizing customer expectations in the environment determined by the customer is stated.

4.2.3 How are these test levels implemented in the software industry?

In the interviews, participants were asked whether they used a certain testing strategy in their projects and if these test strategies aligned with Test Pyramid. While a defined test strategy was used in 80 percent of the projects, it was observed that no strategy was defined in the remaining twenty percent. In addition, it was observed that 25 percent of this 80 percent determined a general testing strategy and used the same strategy in all their projects. It was learned that the remaining 75 percent defined the test strategy on a project basis. It has been learned that the most important factors in determining this strategy are determined according to the expectations of the project customer (contract), the industry (defense, financial energy, etc.) and the standards that the project must comply with. See the Figure 4.7.

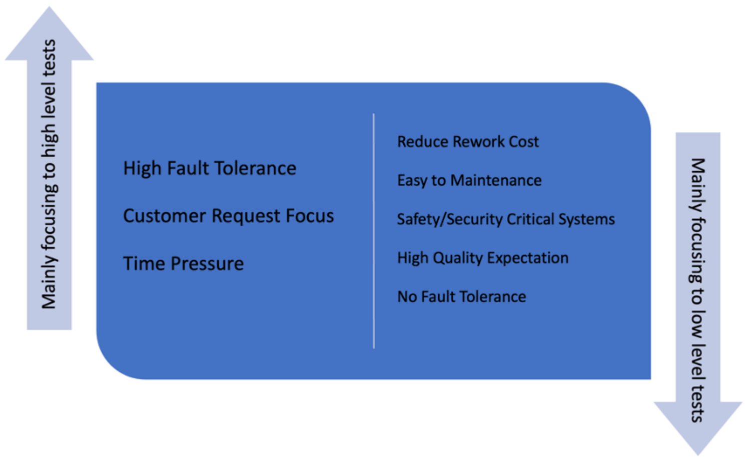


Figure 4.7 Possible Reasons to Focus High or Low Level Tests

This has shown us that the most effective factors in determining the testing strategy are the risk tolerance of the project and the acceptability of the effects that errors may cause. In addition, participants were asked whether the people and teams performing the testing were from the development team or an independent team. The results were provided in Table 4.1.

Table 4.1 How Test Level are Implemented Participants' Current Projects

Percentage of the Participants	Test Level Implementations
40%	All tests are developed by development team. There is no seperation or independency in team for implementing test levels.
60%	While unit and integration tests are developed by development team, there is also an independent team to perform system and aceptance tests.

While 40 percent of the participants reported that all test levels were developed by the development team, the remaining 60 percent reported that unit and integration tests were carried out by the development team, but system and acceptance tests were carried out by an independent testing team.

CHAPTER 5

DISCUSSION AND CONCLUSION

One of the most important issues in software projects is to develop products with the expected quality. To ensure this, certain levels of testing are performed while the project is being developed. It was seen during the literature research that the best combination of these test levels to increase the quality of software projects is the application of the software testing pyramid. The software testing pyramid focuses on keeping the quality at the best level, detecting errors in the early stages, and thus reducing the effects of errors while reducing their costs. It puts forward which test level should be developed from which point by taking this perspective into consideration. This thesis was written to understand how the software testing pyramid and testing levels are defined in the literature and how these levels are realized in the software industry.

As a result of the surveys and interviews, it has been seen that the application of software levels as they are in the software pyramid depends on some different criteria. For example, it has been observed that in projects developed in the defense industry, these test levels are actually used in the hierarchy defined in the literature.
The most important reasons for this are the criticality levels of the developed projects. Also, one of the reasons is the magnitude of the effects that errors will cause. Additionally, in this sector, customer expectations and standards that must be adhered to ensure that these test levels are used as defined in the literature. In addition, it is seen that in civil projects, not all levels in the test pyramid are always used, and sometimes this can be adapted according to the project. These types of projects generally have low error tolerance and impact. In these projects, the test strategy is typically determined by development and the test team's knowledge of the test levels.

Additionally, it is very important for the team to know which functionality will be tested at which level. Otherwise, it will be seen that the tests written do not really serve their purpose. This situation causes an issue that can be tested at the unit test level to be left to the upper levels and the test levels not to be used for their purpose. In order to achieve this balance, it is very important to take into account the experience and knowledge level of the team when determining the test levels and the intensity at which they will be used. Considering all the analyzes made, customer expectations, signed contract requirements, standards to be complied with and the effects of possible errors are important in determining and using the software testing strategy.

In this regard, in addition to the surveys and interviews about which test levels were applied and how within the scope of the companies' projects, it was investigated whether similar studies had been conducted through literature research. Related works with this study has been given in Section 2.5. In contrast to our findings, Radziwill and Freeman (2020) hypothesized that since unit tests should be the foundation for system behavior, performance, and stability, they may stand alone and do not need contract or functional testing at any level. Functional testing is really getting more and more important. This contradictory finding can be explained by the system which is focused for this study. In their study, the data-driven systems have been considered [7]. However, in our study, different people from different industries have been included.

5.1 Limitations of Study

In this study, surveys and interviews were conducted taking into account the experiences of the participants. For this reason, a limited number of participants could be reached for this study. Therefore, results of these interview and survey should be validated by additional studies, including higher number of participants, from different fields.

5.2 Threats to Validity

Although the results of our survey provide valuable insights about how test levels are determined in the software industry, it is crucial to acknowledge the potential threats to the validity based on or results. Several concerns should be noted which could impact the reliability and generalizability of our results. Most importantly, in this study, as participants have been selected according to their experience on software testing field. Both the survey and the interview were conducted with limited number of participants. Hence, our survey sample may not be fully representative of the entire software industry. When conducting the online survey, some of the participants may struggle to accurately recall and report details about their experiences with test levels, leading to potential inaccuracies in the data. In order to eliminate this concern, more detailed information is asked during the interviews to obtain more reliable answers.

REFERENCES

[1] A. A. Sawant, P. H. Bari, and P. M. Chawan, “Software Testing Techniques and Strategies.” *Journal of Engineering Research & Applications*, vol. 2, pp. 980-986, May-June 2012.

[2] E. Daka and G. Fraser, “A survey on unit testing practices and problems.” *Proceedings International Symposium on Software Reliability Engineering,* 2014, pp. 201-211.

[3] K. Hrabovská, B. Rossi, and T. Pitner, “Software Testing Process Models Benefits & Drawbacks: a Systematic Literature Review.” *arXiv preprint arXiv:1901.01450*, vol. 1, Jan. 2019.

[4] M. A. Jamil, M. Arif, N. S. A. Abubakar, and A. Ahmad, “Software testing techniques: A literature review,” in *6th International Conference on Information and Communication Technology for the Muslim World,* 2016, pp. 177-182.

[5] S. Elbaum, S., A. G. Malishevsky, and G. Rothermel, “Prioritizing test cases for regression testing,” in *Proceedings of the 2000 ACM SIGSOFT international symposium on Software testing and analysis,* 2000, pp. 102-112.

[6] V. Hartikainen, “Defining suitable testing levels, methods and practices for an agile web application project.” M.A. thesis, Lappeenranta-Lahti University of Technology, Finland, 2020.

[7] N. Radziwill and G. Freeman, “Reframing the Test Pyramid for Digitally Transformed Organizations,”, *arXiv preprint arXiv:2011.00655,* Nov. 2020.

[8] R. Bierig, S. Brown, E. Galván, and J. Timoney, *Introduction to Software Testing.* Cambridge University Press, 2021, pp. 283-295.

[9] J. L. Dalley, “The art of software testing,” in *IEEE Proceedings of the National Aerospace and Electronics Conference,* 1991, pp. 757-760.

[10] M. Olan, “Unit testing: test early, test often.” *Journal of Computer Science in College,* vol. 19, pp. 319-328, 2003.

[11] S. P. Shashank, P. Chakka, and D. V. Kumar, “A systematic literature survey of integration testing in component-based software engineering,” in *2010 International Conference on Computer and Communication Technology,* 2010, pp. 562-568.

[12] H. K. N. Leung and L. White, “A study of integration testing and software regression at the integration level,” in *Conference on Software Maintenance,* 1990, pp. 290-301.

[13] A. W. Williams, “Software Component İnteraction Testing: Coverage Measurement And Generation Of Configurations.” PhD thesis, University of Ottawa, Canada, 2002.

[14] M. Shi, “Software Functional Testing from the Perspective of Business Practice.” *Computer and Information Science*, vol. 3, p. 49, 2010.

[15] O. Bühler and J. Wegener, “Evolutionary functional testing.” *Computers & Operations Research*, vol. 35, pp. 3144-3160, 2008.

[16] J. Weiss, A. Schill, I. Richter, and P. Mandl, “Literature Review of Empirical Research Studies within the Domain of Acceptance Testing,” in *Proceedings - 42nd Euromicro Conference on Software Engineering and Advanced Applications*, 2016, pp. 181-188.

[17] I. Otaduy and O. Díaz, “User acceptance testing for Agile-developed web-based applications: Empowering customers through wikis and mind maps.” *Journal of Systems and Software*, vol.133, pp. 212-229, 2017.

[18] P. Pandit and S. Tahiliani, “AgileUAT: A Framework for User Acceptance Testing based on User Stories and Acceptance Criteria.” *International Journal of Computer Applications*, vol. 10, pp. 120, 2015.

[19] K. Naik and P. Tripathy, *Software Testing and Quality Assurance: Theory and Practice*. Canada, 2008, pp. 173-185.

[20] V. Mukhin, Y. Kornaga, Y. Bazaka, I. Krylov, A. Barabash, A. Yakovleva, and O. Mukhin, “The Testing Mechanism for Software and Services Based on Mike Cohn’s Testing Pyramid Modification,” in *Proceedings of the 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, 2021, pp.589-595.

[21] H. Vocke, “The Practical Test Pyramid.” Internet: https://martinfowler.com/ articles/practical-test-pyramid.html, Feb 26, 2018 [Dec. 22, 2023].

APPENDIX A INTERVIEW QUESTIONS

Section-1. Basic Information

1.1 Which department did you graduate from?

1.2 What is the last degree you completed?

1.3 How many years of work experience do you have?

1.4 How many years of software testing experience do you have?

1.5 Which sector do you work in?

Section-2.1 Personal Evaluation About Test Levels

2.1.1 To what extent do you know which test levels?

2.1.2 Which levels of testing do you think contribute the most to software quality?

Section-2.2 Using Test Levels in Current Work Experience in Line with Product Quality Goals

2.2.1 What testing levels are used in your current projects?

2.2.2 Does the testing strategy you use in your projects differ from project to project or do you use a single strategy for all of them? If you are applying a different strategy for each project, what are the inputs you use to decide on this strategy?

2.2.3 Does a team independent from the development team perform the tests, or are the developer and tester the same person? Does this vary depending on test levels?

APPENDIX B SURVEY QUESTIONS

Section-1. Basic Information

1.1. What age group are you in?

- 18 - 25
- 26 - 33
- 34 - 41
- 42 - 49
- 50 and over

1.2. What is your gender?

- Male
- Woman

1.3. What is the last degree you obtained?

- Associate degree
- Bachelor’s degree
- Master’s degree
- Doctorate

1.4. Select the program(s) you graduated from.

- Computer engineering
- Software engineering
- Information Systems Engineering
- Computer Science
- Associate degree
- Other:

1.5. What is your position in the company you work for?

- Developer (Software Developer)
- Test engineer
- System Architect
- Project manager
- Other:

1.6. How many years of experience do you have in the testing field?

- I have no experience with the test.
- Less than 5 years
- 6 - 10 years
- 11 - 15 years
- More than 15 years

1.7. How many years have you been practicing your profession? *

- 0 - 3 years
- 3 - 6 years
- 7 - 10 years
- more than 10 years

1.8. What is the industry you are working in?

- Academic
- Information technologies
- Electronic
- Energy
- Security
- Automotive
- Health
- Defense industry
- Other:........

Section-2.1 Personal Evaluation About Test Levels

2.1.1. Please mark to what extent you evaluate your mastery of the test levels given in the table below (1: Lowest, 5: Highest).

Quality Attribute:	**1**	**2**	**3**	**4**	**5**
Unit Testing					
Integration Test					
Functional Test					
System Test					
Acceptance Test					

2.1.2. Please mark how effective you think the test levels given in the table below are in terms of product quality (1: Lowest, 5: Highest).

Quality Attribute:	**1**	**2**	**3**	**4**	**5**
Unit Testing					
Integration Test					
Functional Test					
System Test					
Acceptance Test					

Section-2.2 Using Test Levels in Current Work Experience in Line with Product Quality Goals

2.2.1 Evaluate and mark how often you use the test levels given in the table below in your current job in terms of product quality (1: Lowest, 5: Highest).

Quality Attribute:	**1**	**2**	**3**	**4**	**5**
Unit Testing					
Integration Test					
Functional Test					
System Test					
Acceptance Test					

2.2.2 Please mark how effective you think the test levels in the table below are in terms of product quality in your current project (1: Lowest, 5: Highest).

Quality Attribute:	**1**	**2**	**3**	**4**	**5**
Unit Testing					
Integration Test					
Functional Test					
System Test					
Acceptance Test					

2.2.3 In the light of your experiences in the institution you work for, what would you like to add/suggestions, if any, regarding achieving the quality targets of test levels?

APPENDIX C INTERVIEW QUESTIONS IN TURKISH

Bölüm-1.Temel Bilgiler

1.1 Hangi bölümden mezun oldunuz?

1.2 En son tamamladığınız diploma derecesi nedir?

1.3 Kaç yıllık iş tecrübeniz bulunmaktadır?

1.4 Kaç yıllık yazılım testi tecrübeniz bulunmaktadır?

1.5 Hangi sektörde çalışmaktasınız?

Bölüm-2.1 Test Seviyeleri Hakkında Kişisel Değerlendirme

2.1.1 Hangi test seviyelerine ne derece hakimsiniz?

2.1.2 Hangi test seviyelerinin yazılım kalitesine en çok katkısı olduğunu düşünüyorsunuz?

Bölüm-2.2 Test Seviyelerinin Ürün Kalite Hedefleri Doğrultusunda Mevcut İş Deneyiminde Kullanılması

2.2.1 Mevcut projelerinde hangi test seviyelerini kullanılmaktadır?

2.2.2 Projelerinizde kullandığınız test stratejisi projeden projeye farklılık göstermekte midir yoksa hepsi için tek bir strateji ile mi ilerliyorsunuz? Eğer herbir proje için farklı strateji uyguluyorsanız bu stratejiye karar verme konusunda kullandığınız girdiler nelerdir?

2.2.3 Testleri geliştirme ekibinden bağımsız bir ekip mi gerçekleştirmektedir yoksa geliştirici ve testçi aynı kişi midir? Bu test seviyelerine göre değişiklik göstermekte midir?

APPENDIX D SURVEY QUESTIONS IN TURKISH

Bölüm-1.Temel Bilgiler

1. Hangi yaş grubundasınız?

- 18 - 25
- 26 - 33
- 34 - 41
- 42 - 49
- 50 ve üzeri

2. Cinsiyetiniz nedir?

- Erkek
- Kadın

3. En son edindiğiniz diploma derecesi nedir?

- Önlisans
- Lisans
- Yüksek lisans
- Doktora

4. Mezun olduğunuz programı/programları seçiniz.

- Bilgisayar Mühendisliği
- Yazılım Mühendisliği
- Bilişim Sistemleri Mühendisliği
- Bilgisayar Bilimleri
- Önlisans
- Diğer:

5. Çalıştığınız şirketteki pozisyonunuz nedir?

- Geliştirici (Software Developer)
- Test mühendisi (Test Engineer)
- Sistem mimarı (System Architect)
- Proje yöneticisi (Project Manager)
- Diğer:

6. Test alanında toplam kaç yıllık tecrübeniz bulunmaktadır?

- Test ile ilgili tecrübem bulunmamaktadır
- 5 yıldan az
- 6 - 10 yıl
- 11 - 15 yıl
- 15 yıldan fazla

7. Mesleğinizi kaç yıldır icra ediyorsunuz? *

- 0 - 3 yıl
- 3 - 6 yıl
- 7 - 10 yıl
- 10 yıldan fazla

8. Çalışmakta olduğunuz sektör nedir?

- Akademik
- Bilgi Teknolojileri
- Elektronik
- Enerji
- Güvenlik
- Otomotiv
- Sağlık
- Savunma Sanayi
- Diğer:........

Bölüm-2.1 Test Seviyeleri Hakkında Kişisel Değerlendirme

1. Aşağıdaki tabloda verilen test seviyelerine ne derecede hakim olduğunuzu değerlendiriyorsunuz işaretleyiniz (1: En Düşük, 5: En Yüksek).

Kalite Özniteliği:	**1**	**2**	**3**	**4**	**5**
Birim Testi (Unit Test)					
Entegrasyon Testi (Integration Test)					
Fonksiyonel Test (Functional Test)					
Sistem Testi (System Test)					
Kabul Testi (Acceptance Test)					

2. Aşağıdaki tabloda verilen test seviyelerinin ürün kalitesi açısından ne derece etkin olduğunu değerlendiriyorsunuz işaretleyiniz (1: En Düşük, 5: En Yüksek).

Kalite Özniteliği:	**1**	**2**	**3**	**4**	**5**
Birim Testi (Unit Test)					
Entegrasyon Testi (Integration Test)					
Fonksiyonel Test (Functional Test)					
Sistem Testi (System Test)					
Kabul Testi (Acceptance Test)					

Bölüm-2.2 Test Seviyelerinin Ürün Kalite Hedefleri Doğrultusunda Mevcut İş Deneyiminde Kullanılması

1. Aşağıdaki tabloda verilen test seviyelerinin ürün kalitesi açısından mevcut işinizde ne sıklıkla kullandığınızı değerlendirip işaretleyiniz (1: En Düşük, 5: En Yüksek).

Kalite Özniteliği:	**1**	**2**	**3**	**4**	**5**
Birim Testi (Unit Test)					
Entegrasyon Testi (Integration Test)					
Fonksiyonel Test (Functional Test)					
Sistem Testi (System Test)					
Kabul Testi (Acceptance Test)					

2. Aşağıdaki tabloda verilen test seviyelerinin mevcut projenizde ürün kalitesi açısından ne derece etkin olduğunu değerlendiriyorsunuz işaretleyiniz (1: En Düşük, 5: En Yüksek).

Kalite Özniteliği:	**1**	**2**	**3**	**4**	**5**
Birim Testi (Unit Test)					
Entegrasyon Testi (Integration Test)					
Fonksiyonel Test (Functional Test)					
Sistem Testi (System Test)					
Kabul Testi (Acceptance Test)					

3. Çalıştığınız kurumdaki tecrübeleriniz ışığında test seviyelerinin kalite hedeflerine ulaşılması konusunda ile ilgili varsa eklemek istedikleriniz / önerileriniz nelerdir?

Lütfen yanıtınızı buraya yazın.