Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Technologies

Diploma Thesis

Automated testing of web application

Almira Kelgenbayeva

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

Bc. et Bc. Almira Kelgenbayeva

Informatics

Thesis title

Automated testing of a web application

Objectives of thesis

The main objective of this work is a comparison of the automated and manual testing methods according to the selected criteria.

Partial objectives:
1) Describe and analyze the application that will be tested for the company
2) Analyze functional requirements and create test cases
3) The created test cases will be tested manually and automatically

Methodology

The theoretical part is based on the analysis of professional information sources about test automation. Test groups, different levels and types of testing, and different tools used for automated testing will be described in this part of the thesis.

Knowledge gained in the theoretical part will be used in the practical part. The methodology of the practical part consists of the analysis and preparing test cases and testing reference data application for shares trading company.

The proposed extent of the thesis

80 pages

Keywords

Testing, automation, web application, behavior-driven testing, Python.

Recommended information sources

GOERICKE, Stephan. The future of software quality assurance. Springer Nature, 2020.

GRAHAM, Dorothy; FEWSTER, Mark. Experiences of test automation: case studies of software test automation. Addison-Wesley Professional, 2012.

KHAN, Rijwan; AMJAD, Mohd. Performance testing (load) of web applications based on test case management. Perspectives in Science, 2016, 8: 355-357

KUMAR, Divya; MISHRA, Krishn Kumar. The impacts of test automation on software's cost, quality and time to market. Procedia Computer Science, 2016, 79: 8-15.

SHETH, Tarik; SINGH, Santosh Kumar. Software Test Automation-Approach on evaluating test automation tools. International Journal of Scientific and Research Publications, 2015, 5.8: 1-4

Expected date of thesis defence

2022/23 SS – FEM

The Diploma Thesis Supervisor

Ing. Miloš Ulman, Ph.D.

Supervising department

Department of Information Technologies

Electronic approval: 14. 7. 2022

doc. Ing. Jiří Vaněk, Ph.D.

Head of department

Electronic approval: 28. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Dean

Prague on 17. 02. 2023

Declaration

I declare that I have worked on my diploma thesis titled "Automated testing of web application" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the diploma thesis, I declare that the thesis does not break copyrights of any person.

In Prague on 15 March 2023   _____

Automated testing of web application

Abstract

The purpose of the theoretical part of this work is to study the main aspects of software testing and identify the different testing types, describe tools used for automated testing consider the history of the development.

The practical task includes:

- Analyse the web application which should be tested
- Create test cases for automation testing
- Execute created test cases

Keywords: Testing, automation, web application, behaviour-driven testing, Python.

Automatizované testování webové aplikace

Abstrakt

Cílem teoretické části této práce je prostudovat hlavní aspekty testování softwaru a identifikovat různé typy testování, popsat nástroje používané pro automatizované testování s ohledem na historii vývoje.

Praktický úkol zahrnuje:

- Analýza webové aplikaci, která by měla být testována
- Vytváření testovacích případů pro testování automatizace
- Provedení testovaní

Klíčová slova: Testování, automatizace, webová aplikace, behavior-driven testing, Python.

Table of content

List of pictures

List of tables

1. Introduction

The rapid development of automation of software development systems and network technologies has led to an increase in production in the software market. Increasing competition between software manufacturers required high attention to product quality. Consumers are starting to pay more attention to the software's quality because of the product range's significant expansion and lower prices.

Now almost every aspect of modern life is computerized. Computers are not just used in daily life for routine tasks; they are also crucial in a number of considerably more important fields, including security, construction, transportation, and medicine. Thus, the issue of software quality becomes especially important since it is not only a matter of comfort but also of safety.

After becoming aware of the above, a significant number of companies around the world started to invest in raising the quality of software. They started to establish quality control departments and use new technologies, which enabled businesses to enhance their competitive advantage by raising the scope of their software products.

Testing is required to determine whether the program operates as expected and whether it satisfies the program's requirements. In the process of creating a software product, it is crucial to quickly identify and fix defects and flaws as it lowers the risk and, consequently, lowers the price of software development. The process of software testing may often be automated, which in some situations can increase the speed and efficiency of testing. The importance of the development of automated testing explains why the topic of this diploma thesis is relevant.

2. Objectives and methodology

Objectives

The main objective of this work is a comparison of the automated and manual testing methods according to selected criteria.

Partial objectives:

1) Describe and analyse the application that will be tested for the company
2) Analyse functional requirements and create test cases
3) The created test cases will be tested manually and automatically

Methodology

The thesis will have two parts: theoretical and practical.

The theoretical part is based on the analysis of professional information sources about test automation. Test groups, different levels and types of testing, and different tools used for automated testing will be described in this part of the thesis.

Knowledge gained in the theoretical part will be used in the practical part. The methodology of the practical part consists of the analysis and preparing test cases and testing reference data application for shares trading company. Once testing was completed the comparison of two testing methods manual and automated was done.

3. Literature review

3.1 Concept of software testing

Software testing is the process of analyzing a software tool and related documentation to identify defects and improve the quality of the product. It is a verification of the correspondence between the actual and expected behavior of the program, carried out on the final set of tests, selected in a certain way. Testing is one of the quality control techniques which involves activities like creating test cases, executing them, and evaluating the results. According to the ANSI/IEEE 1059 the standard definition of testing can be the process of analyzing a software item to detect the differences between existing and required conditions (that is defects or errors or bugs) to evaluate the features of the software item. (Dhir, 2016) Software testing is more than just error detection, it is running the software under specific controlled conditions, which means that first verify that it behaves "as specified", second it detects errors, and third it validates what has been specified is what the user needs.

- Verification confirms that the software meets its technical specification
- The validation process confirms that the software meets the system product requirements
- Error variance between the expected and actual result. (Singh, 2019)

Figure 1 shows the general testing scheme.

Figure 1: General testing scheme (Kelgenbayeva, 2023)

The tester receives the requirements and a program that must be tested at the input. Observing the program under certain conditions, the tester receives information about the compliance or non-compliance of the program with the requirements at the output. This information is used to fix bugs in an existing product, or to change requirements for a product that is still under development.

3.2 History of software testing

The history of the evolution of software testing can be separated into distinct eras since testing and quality issues have been managed in different ways during the decades of software development.

The first software systems were created as a part of projects for scientific research or projects for the requirements of the military sectors of the country. Such products had completely standardized testing, with a record of all test procedures, test data, and results. Testing was divided into a separate process that started after coding was finished but was typically done by the same team.

The history of software testing can be divided into several phases.

- Debugging-orientated phase
- Demonstration-oriented phase
- Destruction-orientated phase
- Evaluation-oriented phase
- Prevention-oriented phase

Debugging–orientated phase of testing took place mainly between 1950 and 1957, when there was no clear agreement on what to call testing and what to call debugging. At this time, more attention was paid to the hardware of the computer rather than the software. The term "bug" is believed to be used for the first time and then coined by Thomas Edison in 1878. Although Edison worked mostly with hardware, he wrote a letter to his associate, in which he coined the word "bug" (Prytulenets, 2022) Software bug means an error in the program or in the system, due to which the program produces unexpected behavior. Further, in 1949, Alan Turing, an English mathematician, wrote a scientific paper called Checking a large routine". In this paper, he noted that in order to prove the correctness of a program, the developer must make a set of simple comparisons or statements that can be checked individually, and which easily prove the correct behavior of the entire program. (F.L.Morris, 1984)

The next phase lasted from 1957 - 1978 and is called the Demonstration-oriented Period.
In connection with the development of programming and with the expansion of the tasks of the programs being created, it became necessary to introduce the concept of testing.

Computers began to be used not only for mathematical calculations but also for programs of vital importance, this is what contributed to the separation of the concepts of testing and debugging. The main event of this testing period is that the debugging and testing were clearly distinguishable by including efforts to detect, locate, identify, and correct faults. Charles Baker emphasizes program checkout with two goals:

- to make sure the program runs, and

- the program solves the problem. (Manpreet Kaur)

The years 1979-1982 have been named in testing history as the Destruction-oriented period. This period was so called because in the book The Art of software testing" released in 1979 Glenford J. Myers calls testing the process of executing a program with the intention of finding errors. Testers during this period began to create test cases to break the program. Mayer states that it's more efficient to show that a program has bugs than to show that there aren't any. "A successful test case is one that detects an as-yet-undiscovered error. (Myers, 2004) The destruction-oriented method also came up short since the software would never be released because testers could keep finding bugs. Additionally, resolving one bug might create another.

From 1983-1987 there was an era of Evaluation-oriented testing. During this period, methodologies were developed to ensure program evaluation throughout the software development life cycle. In 1983, the Institute for Computer Sciences and Technology of the National Bureau of Standards published Guideline for Lifecycle Validation, Verification, and Testing of Computer Software. This publication describes testing as a methodology that includes analysis, review, and testing activities. (Panthers, 2016)
It is important to note that from this period the tester starts working from the very beginning of the software life cycle, and not after the development of the program under test. This helped in the detection of errors in the description of the functionality and in requirements.

The Prevention oriented period (1988-1996) focused on finding defects as early as possible. In the early 1990s, the term "testing" began to be used to refer to the organization, creation, maintenance, and execution of tests and test environments. This marked a shift from testing to quality assurance, which now encompasses the entire software development cycle. In

1993, the SCRUM methodology was developed, which made it possible to work effectively on software development in a constantly changing reality and requirements.

Since 1996, the period of Process-oriented testing has begun. During this period, testing has grown from a stage of the software life cycle into an independent activity that is carried out in parallel with the rest of the software life cycle processes from the very beginning. In 2001, a new Agile methodology appears where the following points were noted:

- individuals and interactions over processes and tools
- working software over comprehensive documentation
- customer collaboration over contract negotiation
- responding to change over following a plan (Kent Beck, 2001)

3. 3 Software development life cycle

Any software product, like physical goods, goes through a variety of stages of development, beginning with the idea of production and finishing with the end product.

A software process is a set of work activities, actions, and tasks that are required to build software. The aim of a software process is to produce high-quality software within budget and time. The process can be seen as a road map which guides project participants about the necessities to successfully complete the project. (Ritu Jain, 2015) The workflow of creation can be guided in the right direction with the aid of an appropriate framework. The creation of a software product would not be systematic and disciplined without the use of a precise life cycle model. There needs to be agreement among team members regarding when and what to do when producing a software product.

SDLC refers to software development life cycle, i.e. the various stages used in the life cycle of software development. There are various software development approaches defined and designed which are used during the development process of software, these approaches are also referred to as "Software Development Process Models". Software development life cycle is basically a systematic way of developing software. (Maneela Tuteja, 2012) To put it another way, this refers to the period from the start of the development and implementation of a software product. The eight primary steps of the SDLC are illustrated in the diagram below.

Figure 2: SDLC (BCT, 2023)

Generally, the SDLC is a closed loop in which each stage influences the activities in the following stages and offers encouraging signs for the future. All eight processes try to influence one another effectively and consistently to provide answers to specific issues and guarantee consistency in the development process.

3.3.1 Planning

What do you want to do? is the question that needs to be answered at this point. The team may be motivated by this question to understand the unit economics (prices and benefits), risk management strategies, and expected costs of future development.

A good example of the importance of planning is the Instagram story, which's planning phase took an incredibly long time. Due to the fast expansion of social media during this time, little was known about how users will interact with the product. The developers made careful planning and spent a lot of effort on a design since they recognized that a powerful primary experience (collecting, editing, and sharing photographs) would guarantee growth, success, and high conversion. They constantly considered the future of social media and kept an eye out for emerging trends. (Naor, 2020)

3.3.2   Design requirements

At this stage, the SDLC should determine which issues require solutions and collect input and support from the appropriate internal and external parties. Consideration must be given to all possible customers for the newly developed product. Clients, designers, managers, or other technical team members may offer different ideas. It is important to properly identify and record product needs and get them accepted by the client or market analysts after completing the requirement study. Software Requirement Specification is used to outline all the product's design and development needs over the course of the Software Development Life Cycle. (BCT, 2021) This stage of software development is one of the most difficult stages. At this stage, various types of challenges may arise. For example,

- Low client involvement and insufficiently detailed requirement specifications result in incomplete knowledge and delay in development
- A slowdown of requirements negotiation and reduction in transparency due to the continuous passing of information back and forth between different sites involved in the development
- The client's requirements were not fully absorbed by the vendor team due to the restricted flow of contextual or open-ended information. Therefore, a part of the required documentation was based on assumptions
- Clients do not share responsibility for the delay with vendors even when the delay is due to late inputs from the client side (Ritu Jain, 2015)

3.3.3   Design

The functionality of the software program is simulated during the design process. Some characteristics of the design stage include the following:

- Programming language, industry standards, and general design of applications are all specified in the architecture.
- The user interface of software determines how users interact with it and how it responds to input.
- Platforms - Describes the types of operating systems that will be used to operate the software.
- An application's ability to communicate with other assets, such as a central server or other instances of the program, is described in the communications section.

- Security refers to the steps taken to keep the program safe and may include data encryption, SSL traffic encryption, and secure user credential storage.

3.3.4 Build

The actual development phase of the SDLC starts here, and software is created. Coding represents the start of implementation. Programming tools including compilers, interpreters, debuggers, and other similar tools are used to generate and implement the code, and developers must respect the coding standards outlined by their management. "Developers must keep on communicating with their colleagues for sharing ideas, knowledge, and artifacts, and resolving confusion. Communication to vendor team only via project manager creates communication bottlenecks and leads to frustration in client organization and delay of the project. In addition, at this stage may arise other challenges, such as unplanned communication that distract developers from their regular work. Developers often have to resolve requirement issues that were not resolved during analysis and design. All above-mentioned issues can be solved by:

1. Development should follow clearly established processes. For quick delivery, good technical compatibility and support are required.
2. Clear, objective decisions that are focused on the project's success are required.
3. When necessary, specialists can be contacted using knowledge management tools." (Ritu Jain, 2015)

3.3.5 Document

In the context of software development, "technical documentation" covers all textual materials relating to the development of software. No matter the size or complexity of the software development project, documentation is always necessary. Additionally, other types of papers are produced during the entire software development lifecycle (SDLC). Clarifying product functionality, collecting project-related data, and simplifying information exchange between stakeholders and developers are the objectives of documentation.

3.3.6 Testing

An app should undergo extensive testing before being made accessible to the public. The testing can be executed either automatically or manually, depending on technical specifications. Complex apps might need a simulated production environment, which can

only be built in a specialized testing environment. The testing procedure will cause fewer errors and malfunctions for users.

3.3.7 Deployment

The main goal at this stage of the procedure is to deploy the program into a live environment. Because of this, many companies decide to first introduce their goods in a testing or staging environment. This permits quick product prototyping and testing while ensuring that any last-minute mistakes are found and fixed before the product is launched to the market.

3.3.8 Maintenance

At this point, the development cycle is almost finished. The software has been finished and is in use right now. The operating and maintenance phase is nevertheless essential. End users uncover bugs in this situation that developers missed during testing. These problems must be resolved immediately because they could trigger new development cycles.

3. 4 SDLC models

During the existence of project management, many effective approaches, methods, and standards have been created that can be adopted nowadays. A software development life cycle model (also termed process model) is a pictorial and diagrammatic representation of the software life cycle. A life cycle model represents all the methods required to make a software product transit through its life cycle stages. It also captures the structure in which these methods are to be undertaken. (JavaTpoint)

The testing process, which determines the choice of approach, timetable, necessary resources, etc., is significantly impacted by the software development model that is selected. Most system developers presently employ one of two SDLC methodologies: traditional development or agile development.

The following are different models for software development:

- waterfall,
- v-shaped,
- iterative incremental,
- spiral

Software methodologies like the waterfall method and V-shaped are classified as traditional software development methodologies. Traditional software development methodologies require defining and documenting a stable set of requirements at the beginning of a project. (Yu Beng Leau, 2012) The success of this method of software development depends on a number of factors, first of all, it is necessary to know all the requirements for the product before starting its development. The introduction of any changes during development will be problematic, this can be considered the main disadvantage of this method. But at the same time, the limitation of the cost of products and useful resources is reduced.

Agile development is based on the idea of incremental and iterative development, in which the phases within a development life cycle are revisited repeatedly. It iteratively improves software by using customer feedback to converge on solutions. (Yu Beng Leau, 2012) In the agile model, there is no comprehensive planning and only clear future tasks that are associated with the characteristics that must be developed. The team adjusts to sudden changes in the needs of the product. Regular testing reduces the possibility of a serious flaw in the product. Interaction with the clients is the strong point of agile methodology and open communication and minimal documentation are typical characteristics of the agile development environment. Teams collaborate closely and often are in the same geographical space. (Marian STOICA, 2013)

3.4.1   Waterfall

The waterfall model was defined by Winston W. Royce in 1970. It is also known as the linear- sequential life cycle model. This model is easy to understand and use. (Marian STOICA, 2013) This model assumes that each project phase will be carried out strictly sequentially and only once. Only when the previous stage has been successfully completed is it feasible to move on to the next. Each stage presumes careful planning and total correctness of the stage's output. A waterfall model is a good option if the project conditions are met:

- The project is short and has zero risk
- Requirements are fixed
- Technology is stable
- All necessary resources are available

Below the diagram of the waterfall model is represented.
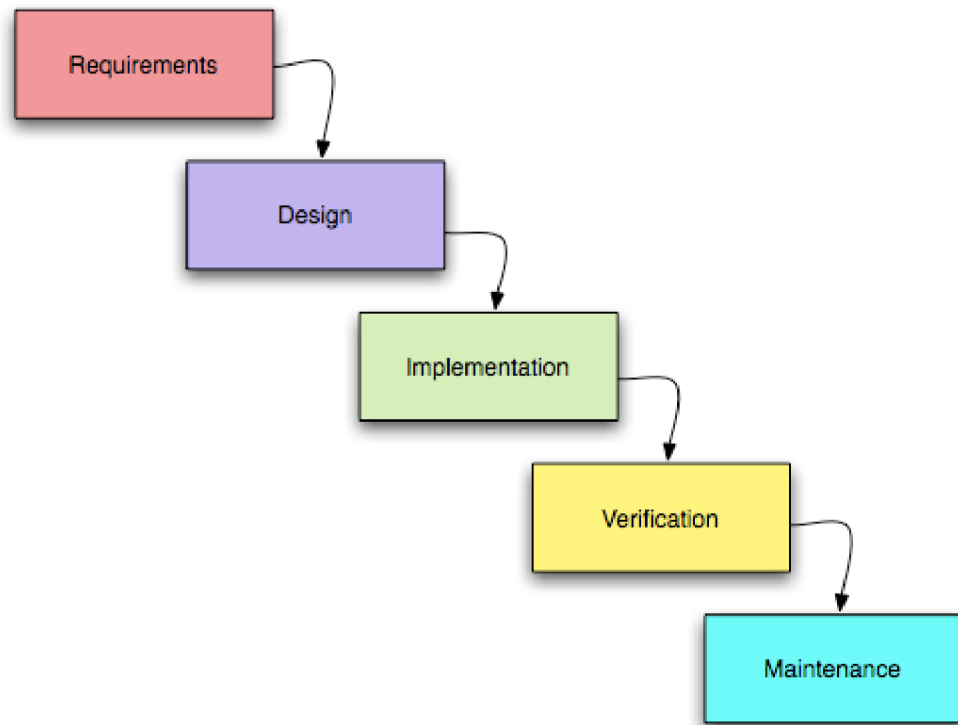
Figure 3: Waterfall model (Evolution, 2023)

3.4.2 V-shaped

V-model means Verification and Validation model. Just like the waterfall model, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. Testing of the product is planned in parallel with a corresponding phase of development. (Prof. Seema Suresh Kute, 2014) As well as the waterfall model, is used for small and medium projects with clear requirements and resources. The diagram is presented below.
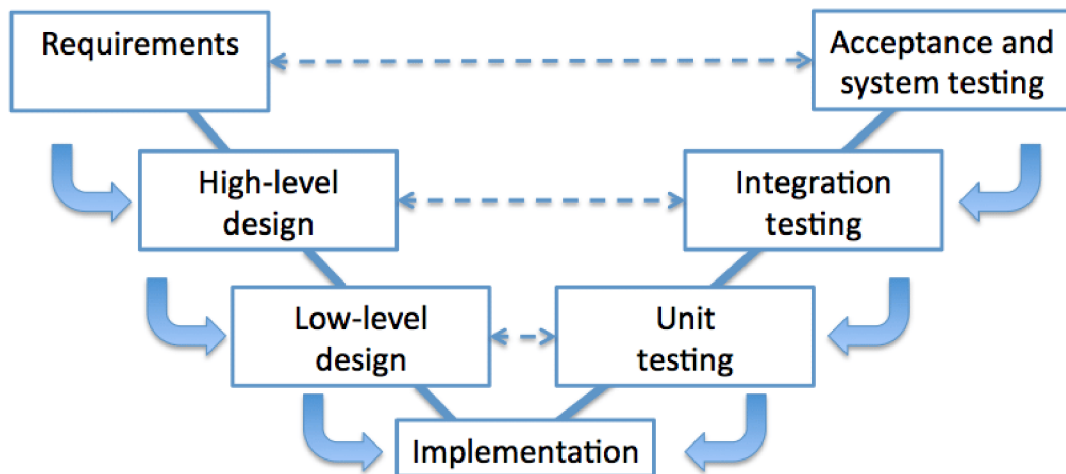
Figure 4: V-shaped (Fronza, 2016)

3.4.3 Iterative incremental model

This model combines elements of the waterfall model in an iterative fashion. The basic requirements are addressed in the first increment, and it is the core product, however, many supplementary features (some known, others unknown) remain undeliverable at this increment. This model constructs a partial implementation of a total system. Then, it slowly adds increased functionality. Therefore, each subsequent release will add a function to the previous one until all designed functionalities are implemented. (Adel Alshamrani, 2015) The following situations will make the model useful:

- if the system is divided into different segments with specific requirements
- there are few resources available for the project
- for startups who have completed investment rounds
- large-scale projects
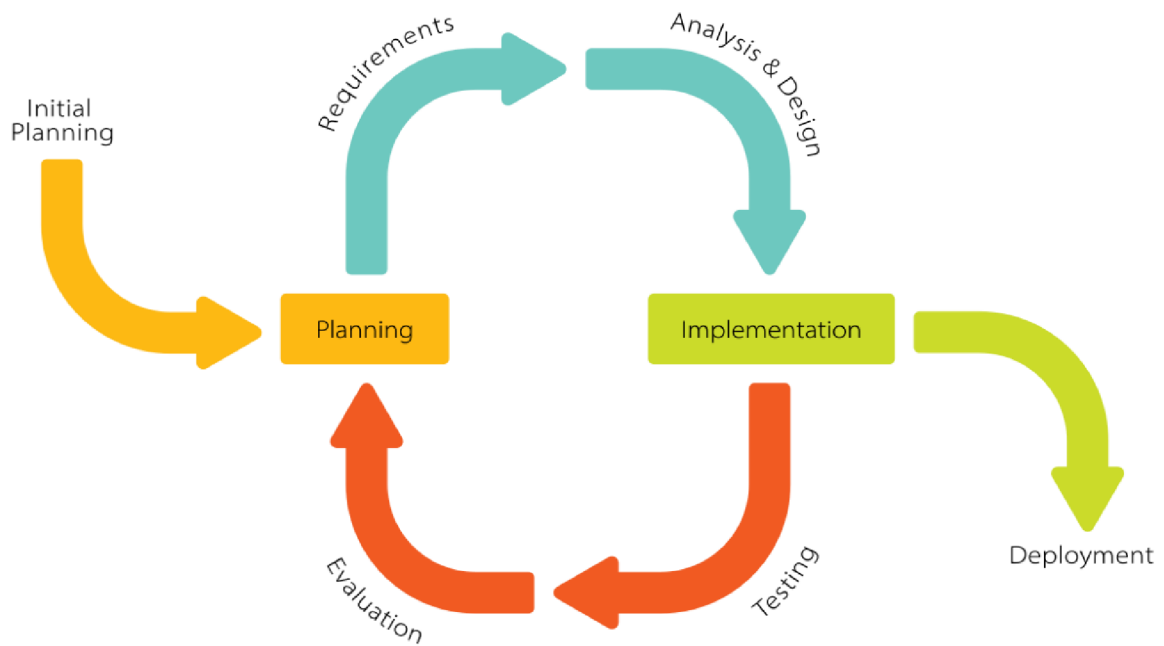
Figure 5: Iterative incremental model (Aprika.com, 2023)

3.4.4 Spiral

Based on risk assessment, this method combines the advantages of waterfall, prototyping, incremental, and iterative models. The design looks like a spiral with numerous circles. Four key phases are clearly highlighted there:

1. Objectives determination and identification of alternative solutions
2. Identification and risk resolving
3. Developing the next version of product
4. Review and plan of next phase

This model focuses on risk assessment and minimizing project risk. This can be achieved by breaking a project into smaller segments, which then provide more ease of change during the development process, as well as providing the opportunity to evaluate risks and weigh consideration of project continuation throughout the life cycle. In this model, the development team starts with a small set of requirements and then goes through each development phase (except Installation and Maintenance) for those sets of requirements. (Adel Alshamrani, 2015)   Below the diagram of the model is represented.

24
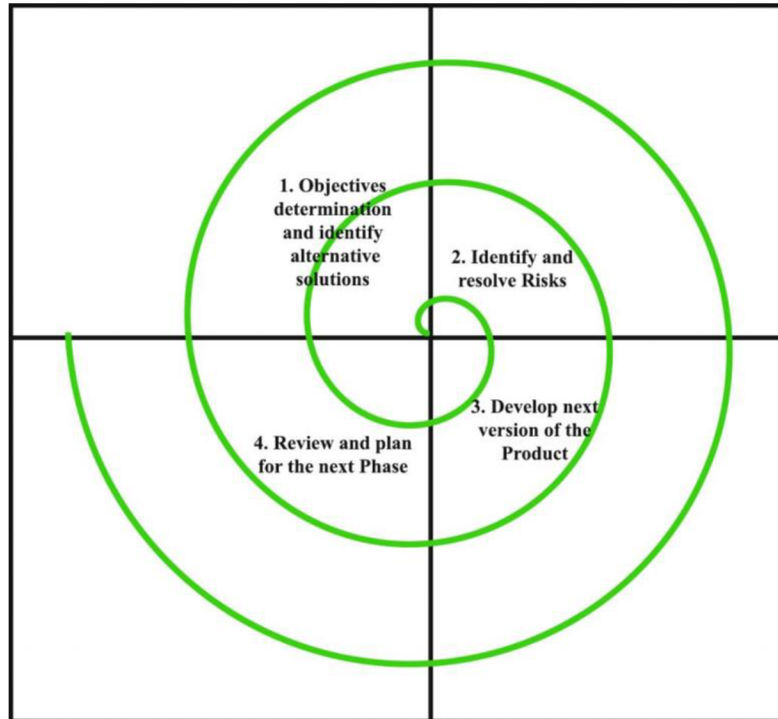
Figure 6: Spiral model (Geeksforgeeks.org, 2023)

3.4.5   Advantages and disadvantages

Model name	Advantages	Disadvantages
Waterfall	1. easy to use model 2. each step is documented 3. result of the project is predictable 4. minimum client intervention	1. difficult and expensive to adapt to changing requirements 2. documenting every phase of a project takes a lot of time 3. can't provide anything to the customer until the completion of the entire project
V-shaped	1. easy to implement 2. test cases are pre-created 3. project budget and duration are predictable 4. structured approach with clearly defined roles and functions	1. making changes in the middle of a project is extremely difficult 2. with many test procedures, there is less time to code 3. requires more specialists

25

Iterative incremental	1. ensures fast and regular "delivery" of working software to customers 2. easier and cheaper to accommodate changes in project requirements 3. small pieces of software are easier to test and fix	1. architectural problems may arise 2. the system should be planned from the very beginning, otherwise, it cannot be divided into modules 3. total system costs will increase as new modules are integrated
Spiral	1. risk analysis at each iteration increases the project's chances of success 2. stable and reliable systems as they are thoroughly tested 3. can change requirements between cycles	1. risk management experience required 2. involves working with a large amount of documentation 3. can't change requirements in the middle of a cycle

Table 1: Table 1: SDLC Models advantages and disadvantages (Anywhere, 2021)

3. 5 Test classification

There are many different types of tests that can be performed while testing a software program. Roman Savin, the author of "Testing Dot com" established the largest and most comprehensive classification. He combined different testing methods based on factors including the test's object, subject, level, positivity, and degree of automation.

Testing must be classified in order to organize information, greatly speed up the design and development of test cases and allow to reduce labor expenses by eliminating the need for the tester to create new models.

The following diagram shows the classification of testing and the types of tests.

26
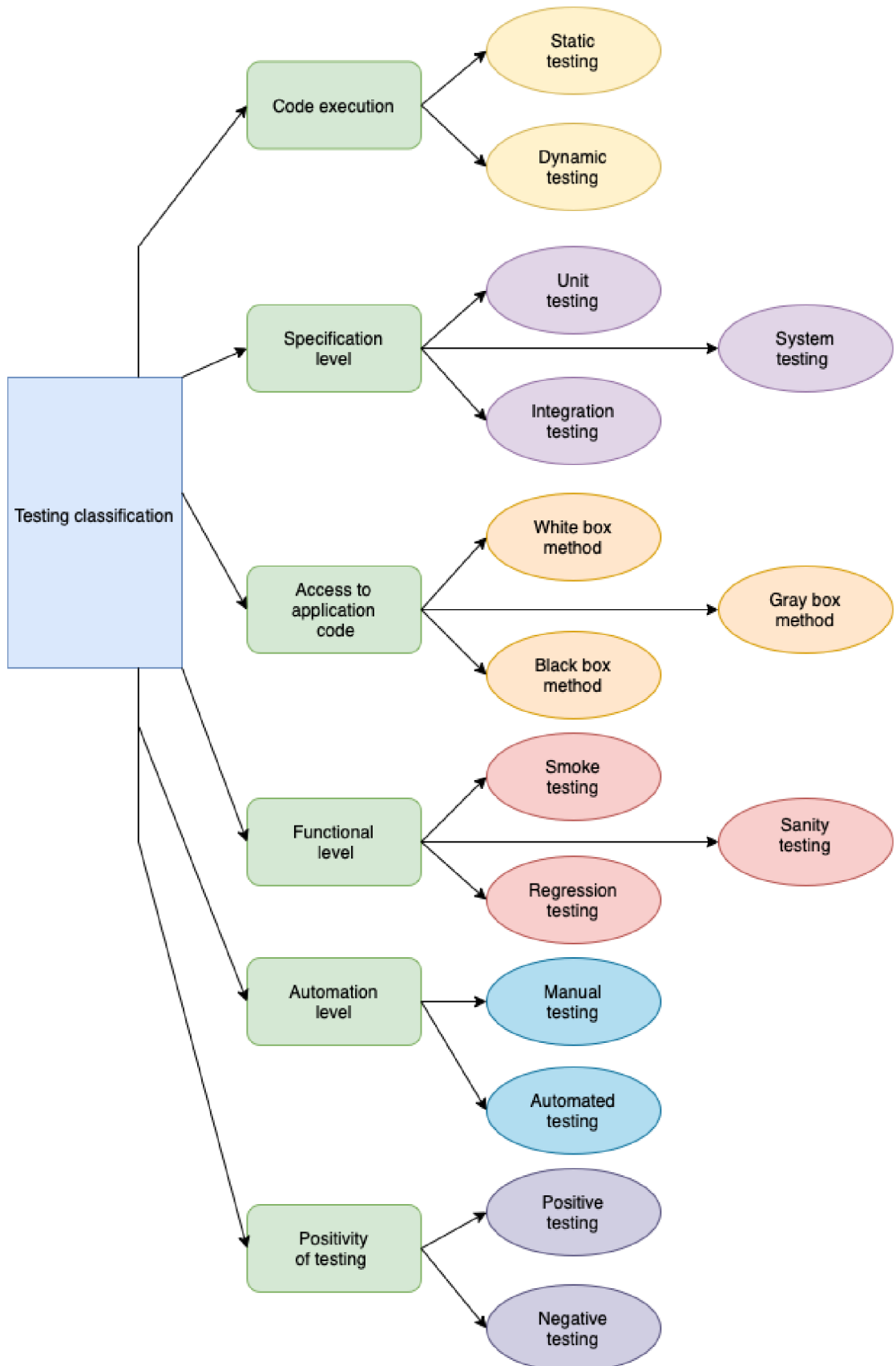
Figure 7: Testing classification. (Adapted from Software Testing. Kulikov, 2020)

3.5.1 Static testing

The static testing method is a type of software testing where software is tested without running code and is a process or tool aimed at detecting possible bugs in software. In addition, it finds and eliminates errors in various accompanying documents, for example, the specifics of software requirements.

There are two forms of static testing:

- Reviews
- Static analysis

Reviews are tests that look for errors in the documentation (requirements, design, test cases, etc.).

Additionally, reviews are divided into:

1. Software inspection. It implies, in most cases, verification of documentation by higher authorities, for example, the study of software requirements.

2. Informal. The final version is presented to the audience for informal discussion when everyone offers their opinions. This helps in finding defects in products at an early stage of development.

3. Expert evaluation. A team of specialists checks the documentation to identify and eliminate errors.

4. Through views. Performed by an experienced specialist (expert) to check defects. The major objective is to avoid having issues throughout the development or testing phase.

Code written by programmers can be examined for structural defects that could result in bugs using static analysis. Static analysis's "composition" also provides an assessment of the quality of developer-written code. The code combination is analyzed using a variety of technologies, and it is then compared to compliance standards. Static analysis can detect the following defects:

- Dead code
- Variables that are not used
- Wrong syntax
- Variables with values that cannot be determined
- Endless cycles (Testengineer, 2022)

Advantages of doing static testing:

1) Discovers issues in the early stages of software development, which helps to lower the cost of resolving problems that are found

2) The statements made throughout this testing process helps in improving the functional aspect of the process, which also aims to prevent similar bugs

3) Provides extensive information on software quality issues

4) Helps to improve the exchange of important information between employees

5) Fixing bugs requires minimal effort, which increases the efficiency of development (TestMatick, 2019)

The disadvantages of doing static testing:

1) The procedure takes a long time because static testing is typically done manually

2) It also prevents the discovery of runtime vulnerabilities. (TestMatick, 2019)

3.5.2 Dynamic testing

Dynamic testing is a method for evaluating a program's functionality during code execution. This type of testing involves the actual operation of the program and the determination of how its functionality works, in accordance with expectations or not.

By providing input data and seeing how the program responds, the dynamic style of testing involves testing the software directly in real time. Dynamic analysis is the study of how a system physically reacts to external variables that vary over time. Dynamic testing requires the actual compilation and execution of the software. It involves interacting with the software, providing input data, and executing certain test cases—either manually or automatically—to see if the output matches the intended outcomes.

Advantages of dynamic testing

1) The runtime environment's vulnerabilities can be found through dynamic testing.

2) Even if the tester is working without access to the actual code, dynamic testing provides application analysis.

3) Some vulnerabilities that are challenging to identify with static testing can be exposed via dynamic testing.

4) The accuracy of the results of static testing can also be verified by dynamic testing.

5) Dynamic testing can be applied to any application.

Disadvantages of dynamic testing

1) Automated security measures, like checking everything, may not be appropriate.
2) Both false positives and false negatives can be produced by automated tools.
3) It can be difficult to find qualified Dynamic Testers.
4) Finding code vulnerabilities is challenging with dynamic testing, and fixing the issue takes longer. Error correction consequently gets expensive.

3.5.3 Unit testing

Unit testing is a process in programming that allows checking of the correctness of individual modules of the source code of the program. The goal is to create tests for each non-trivial method and function. This makes it easier to immediately determine whether the next change in the code has caused a regression, or the appearance of mistakes in the tested sections of the program, and it also makes it easier to find and fix such errors.

Unit testing serves as the foundation of the test pyramid. It has a limited scope and ensures that isolated code units function as expected. Unit tests should assess a single variable and not rely on external dependencies (Singh, 2022)
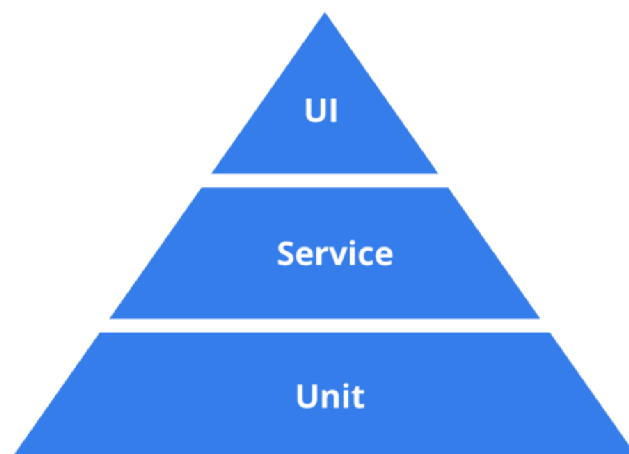

Figure 8: Unit testing (BotPlay, 2023)

The more software components a test affects, the higher it is on the pyramid. High-level UI tests check user interfaces and business logic, putting them closer to the business. And those located at the base of the pyramid assist in identifying issues with specific sections of the code.

Unit tests are needed in the following cases:

- if the code is incomprehensible
- if the code changes often
- if updates in one part of the code may break something in another part.

3.5.4 Integration testing

Integration testing evaluates how various components of a program interact with one another (each of which, in turn, is tested separately at the stage of unit testing). Unfortunately, issues frequently occur at the "junction" of two components' interactions, even when we deal with very high-quality individual components. Integration testing makes these issues clear. (Kulikov, 2020)

3.5.5 System testing

System testing's primary goal is to test the system's functionality during the development of each product version as well as during the alpha and beta testing phases of the software release process. It is s aimed at checking the entire application as a whole, assembled from the parts tested in the previous two stages. It not only identifies defects at the junctions between components but also provides an opportunity to fully interact with the application from the end user's point of view, applying many of the other types of testing listed in this chapter. (Kulikov, 2020)

3.5.6 White box testing

White Box Testing Technique: It is a detailed investigation of the internal logic and structure of the code. In white box testing, it is necessary for a tester to have full knowledge of the source code. (Khan, 2012) White box testing investigates the internal workings and design of a software program to find implementation flaws, such as poor key management. The integration, module, and system levels of the software testing process are all suitable for white box testing. When performing white box testing, the tester must review the source code to identify the problematic code.

Advantages of White box testing:

- Helps in diagnosis by exposing hidden issues
- During the earliest phases of project development, it is possible to automate test cases and their execution in a very easy manner.

- It has a well-developed metrics system, its collection and analysis can be easily automated.
- Encourages developers to produce high-quality code.

Disadvantages of white box testing

- Cannot be carried out by testers without sufficient programming knowledge
- Testing focuses on functionality that has already been developed, which raises the risk of missing requirements that haven't been carried out
- The impact of the execution environment is not taken into consideration while analysing the application's behaviour in isolation from it.
- Application behaviour is studied separately from situations involving actual users.

3.5.7   Black box testing

Black box testing treats the software as a "Black Box" – without any knowledge of internal working and it only examines the fundamental aspects of the system. While performing a black box test, a tester must know the system architecture and will not have access to the source code. (Khan, 2012) The primary source of information for developing test cases in black box testing is documentation.

Advantages of Black box testing:

- It is not necessary for the tester to have strong programming experience
- The impact of the application's behaviour is considered as it relates to the actual runtime environment
- The behaviour of the application is studied within the context of actual user scenarios.
- Test cases can be created as soon as stable requirements start to arise.
- The process of developing test cases enables the identification of requirements defects.
- Allows for the creation of reusable test cases

Disadvantages of white box testing

- A part of the test cases that the developers have previously finished can be repeated.
- It is very likely that some of the application's potential behaviours will go unexplored.
- The creation of very efficient test cases requires appropriate documentation.
- Detected error diagnosis is more challenging than with white box methods.
- Because there are so many different methods and procedures, it is challenging to plan for and predict labour expenses.
- In the event of automation, sophisticated, expensive tools might be necessary.

3.5.8 Grey box testing

White box + Black box = Grey box, it is a technique to test the application with limited knowledge of the internal working of an application and also has the knowledge of fundamental aspects of the system. Grey box testing technique will increase the testing coverage by allowing to focus on all the layers of any complex system through the combination of all existing white box and black box testing Examples of grey box testing techniques are: architectural model, Unified Modelling Language (UML), State Model (Finite State Machine). (Khan, 2012) Grey box testing combines advantages and disadvantages of the White box and Black box testings.

3.5.9 Smoke testing

Smoke testing is preliminary testing to reveal simple failures severe enough to reject a prospective software release. This is the first testing performed on the build and all other kinds of testing follow it. The purpose of smoke testing is to determine whether the new software build is stable or not so that the build could be used for detailed testing by the QA team and further work by the development team. If the build is stable i.e. the smoke test passes then the build could be used by the QA and development team. (Chauhan, 2014) A smoke test saves resources and time. It can be performed by a small group of QA engineers or by the developers themselves because they are not particularly complex. At the same time, they can show that the current assembly does not even fulfill its main tasks, and therefore it simply does not make sense to conduct deeper and more costly testing until the main functionality of the product works stably.

3.5.10 Sanity testing

Sanitary testing is a very specific verification of the performance of individual functional elements, systems, web architectures, and calculations. This kind of testing is frequently carried out after the conclusion of smoke testing but before the whole cycle of regression tests. It is carried out when QA does not have time to perform a detailed check of the entire assembly.

Sanity and smoke types of testing have "vectors of movement" or different directions. Sanity testing is focused deeply on the tested function, as opposed to smoke testing, which is focused broadly to test as many functionalities as possible in the shortest amount of time.

3.5.11 Regression testing

Regression testing is the process of testing a previously tested program to ensure that any changes made did not cause bugs in the parts of the program that remained the same. Regression testing is used to ensure that additions are made correctly and that the software will continue to successfully meet the defined criteria and interact with other systems after the changes have been made. On Scrum projects, regression checks are especially important because they help teams focus on new functionality. QA experts can be confident that the updates had no impact on the functionality that was already in place.

Regression testing does not always mean a complete check of all functionalities. To guarantee the high quality of the released software while maintaining optimal timing and cost, a thoughtful approach is needed to determine the strategy and sufficient regression coverage.

3.5.12 Manual testing

Manual testing is a type of software testing where testers manually execute test cases without using any automation. Manual testing is the most primitive of all types of testing and helps to find bugs in a software system. This type of testing requires more effort but is necessary to test the feasibility of automation. Software testing was based on the concept that "100% automation is impossible." This makes manual testing mandatory. The need for manual testing lies in the fact that with manual testing of the functionality, we can much faster get information about the state of the product we are analyzing, and about the quality of development. In addition, during automation, pre-developed cases often have to be changed

and updated, and it takes a certain time to write auto tests. The significant disadvantage of manual testing is the possibility that a tester will miss a defect due to human factors.

3.5.13 Automated testing

Automated testing is a testing technique in which special programs are used to execute test cases. Automation programs compare the results with the actual ones and generate detailed test reports. With special tools, it is possible to create test scenarios and run them when needed. Once test scenarios are written there is no need for human interaction. Automation of testing aims to minimize the number of tests that must be performed manually. It is important to understand that automated testing will not completely replace manual testing. Automation will make the manual work of test engineers easier and more efficient, allowing them to focus more on problem areas. Usually, large projects use an automated method of testing. If the system is stable and the requirements for the tested areas don't change frequently, tests can be automated. It is recommended to automate only those tests that will be run many times during the development of the software; if tests need to be executed only once, it is unprofitable to automate them.

3.5.14 Positive testing

Positive testing is a type of testing that verifies that the application under test works correctly for a positive set of inputs. Positive testing examines the application in a situation where all activities are carried out strictly as instructed, with no errors, deviations, incorrect data input, etc. If positive test cases end with errors, this is a red flag — the application is not working properly even under ideal conditions. (Kulikov, 2020)

3.5.15 Negative testing

Negative testing should guarantee that the application will work even if a user makes a mistake or behaves in an unexpected way. Negative testing is often referred to as failure testing. It requires maximum creativity, as its intended purpose is to test how errors are displayed and what the user sees. It helps to evaluate the functional reliability of an application or software.

3.6 Automated testing

By improving the entire testing process, continuous testing speeds up the delivery of software. It also ensures that development teams create high-quality, reliable applications by providing quick feedback that enables them to find bugs and other issues in applications at an early stage. Additionally, the ability to plan and carry out effective testing can significantly lower costs in an organization, both by saving developers' time and by creating a strong delivery pipeline that allows them to quickly make changes to the code with little risk of breaking the application in a production environment. Continuous testing's primary component is automation, which offers several advantages:

- quick response
- comprehensive and precise testing
- high test coverage
- detecting errors quickly
- test reuse
- quick deliveries
- adaptation to DevOps
- saving both money and time.

Despite the advantages mentioned above, test automation can have a significant initial cost. It takes a lot of time and money to acquire software, pay for training to use it, design, and create automated tests. However, when it's added more and more new features to a product, manual testing becomes more expensive and automated testing becomes more affordable. (Hayes, 2004)

The chart below shows the long-term difference between automated testing and manual testing in relation to the price and number of test cases conducted.
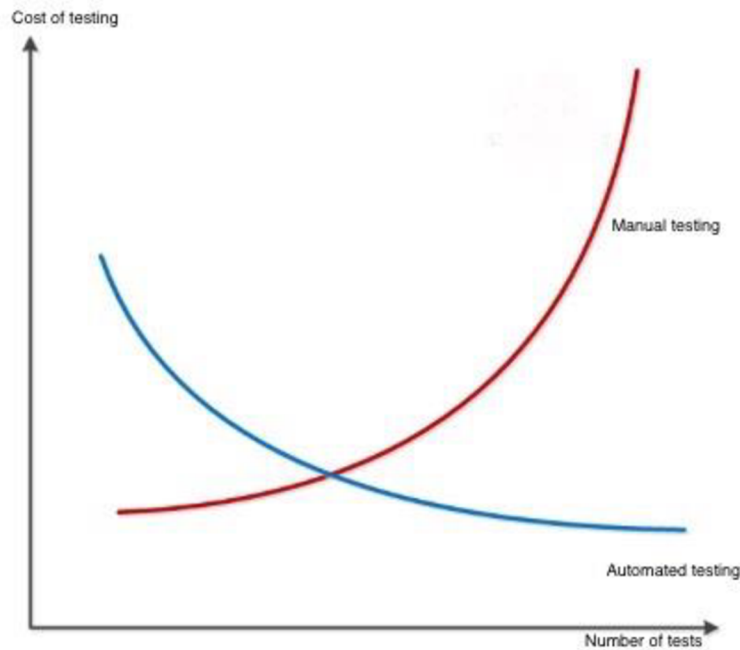
Figure 9: : Manual testing vs Automated testing Adopted from (Enterprise, 2021)

It's important to realize that not everything can be automated and that not everything needs to be. Therefore, before selecting how to best structure test automation, it is crucial to properly assess, research, and analyze the needs of the product. Almost all development teams work on projects that are extremely time-sensitive, thus there is never enough time to implement all the best practices. The same stands true for test strategy, given that testing itself is not often a top priority for development teams. The type of application being created, the deadline, the testing software being used, and the resources available all need to be considered to strike a balance and make the best decision. (Enterprise, 2021) The following are significant test types that can be automated.

3.6.1 Unit testing

Because unit tests focus purely on the part of the code that verifies its functionality and is independent of other components of the application, this is a perfect method to begin using test automation. Thus, developers get more information about the work of the created functionality. Many teams are using the test-driven development (TDD) methodology, where tests are written before code, as a result of the testing culture that exists today. This ensures the quality of both the code and the tests.

37

3.6.2 Regression and integration testing

Regression tests confirm that application features operate as intended, while integration tests check whether separate modules in an application function together. The two tests mentioned above are frequently performed following program modifications or enhancements by testers. By automating these tests, a significant amount of time can be saved and used to perform different kinds of tests.

3.6.3 Performance tests and load tests

Since it is necessary to simulate hundreds or thousands of users working under various conditions from using different browsers, being in different time zones, using different operating systems, etc. automated testing is the only option for performance and load tests.

3.6.4 Consistent Test Scenarios

Development teams must perform these important tests almost regularly. For instance, the functionality of the login page, which allows users to access the application, has an impact on its accessibility. To save developers and testers a ton of time, it is advisable to automate testing.

3.6.5 Basic functionality (smoke tests)

Smoke tests are not as complicated and relatively simple to implement as other tests. But passing these scenarios is important. They provide development teams with information on whether the program's fundamental features—such as if the application's login window opens, whether users can log in, whether the API is available, whether the application is reachable from various locations, etc.—are operating as intended.

3.7 Software development approaches

TDD and BDD are examples of Agile Infrastructure Software Development established by XP (Extreme Programming). It is a software development approaches that uses unit tests to incrementally deliver small functionality. (Moe, June 2019)

3.7.1 Test Driven Development

TDD (Test Driven Development) is a software development technique that relies on repeating very short development cycles: first, a test is written to cover the desired change, then code is written that will allow the test to pass, and finally, the new code is refactored to the appropriate standards. TDD focuses on the "inside-out" perspective and creates tests from a developer's perspective. The methodology focuses specifically on unit tests. The primary goal of TDD is to accomplish the code clearer, simple, and bug-free. (Moe, June 2019)

3.7.2 Behaviour Driven Development

BDD (Behavior-driven development) is a development technique that considers not the result of the execution of any module, but the work that it performs. This principle can be seen as an extension of TDD. BDD focuses on user behavior to define the requirements for the program that is being developed. BDD introduces more understandable testing that makes use of natural language so that all stakeholders can comprehend the testing process better. All acceptance test cases in BDD must cover all file features from a variety of user scenarios. Based on the Given-When-Then structure, each scenario is viewed as a single test case. (Barus, 2019) Cucumber is one tool for automating testing using the BDD methodology. Cucumber will read the feature-containing file specification when used, after which it will observe. In order for Cucumber to process the scenario, some fundamental syntactic rules must be adhered to.

3. 8 Summary of Literature review

In the theoretical part of the work, the first was studied the concept of software testing and learned about the development of testing as an independent discipline at different times in history. There were also studied what SDLC is and learned about its stages in more detail. After that, a review of SDLC models was done and their advantages and disadvantages were identified.

In the last chapters, the different classifications of tests used today in software development have been described in more detail. Since in the practical part, it will be necessary to test the application not only manually, but in the way of automation, this topic was also considered in the theoretical part.

The following chapters will describe how the selected application was tested and what steps were taken to do so, and will reveal the results to compare the two testing methods.

4. Practical part

The main goal of the practical part of this work is a comparison of two methods of testing manual and automated according to three criteria:

- estimated speed
- labor cost
- usability rating

Testing will be demonstrated on a web application used internally by the company. The company is an international exchange organization that manages the work of various securities markets. The following research questions will need to be answered based on the comparison's findings:

a. Can the company fully migrate from manual to automated testing of the chosen application?

b. How can this application benefit from automated testing?

c. Is automated testing suite less labor cost than manual testing?

The obtained results and conclusions will help the company in further planning the development and support of this web application in terms of testing.

In order to achieve the main goal of the practical part the following works will be carried out:

- Describing and analyzing the web application used for testing
- Selection of tools used for automated testing
- Analyzing the test requirements
- Creation of test cases
- Manual testing of the web application
- Automated testing of the web application
- Get and conclude results
- Compare obtained results based on the selected criteria
- Make a conclusion and answer the research questions

4.1 Web application

The application that will be tested in this thesis is a web application used internally by the company and is called the Reference Data Factory, RDF for short. This application is used to create financial instruments as well as to create institutions that issue these instruments. It also helps to provide services for fund management, settlement and custody of financial instruments, as well as collateral and liquidity management.

Internal teams that issue and register securities and institutions utilize the application to prepare them for later acquisition and selling on the stock exchange.

Since the company is engaged in the regulation and admission to trading of securities not only locally but also on the international market, this application is often subject to various changes so that the issued instruments comply with all legal norms.

The testing of the web application will focus on its front-end functions. Three testers will assist to get real results. Each of the three testers has a unique background working with this application. They will be referred to as tester A, B, and C in the later paragraphs of the practical work.

Tester A has been working with this application as a tester for more than 6 years. Has in-depth knowledge of the functionality of the application. Tester B is a beginner and has been working as a tester for 1 year. Before that, he had no experience and did not use this application. Tester C has been working in the field of testing for about 1 year but before that used this application as an end user. In total, has more than 4 years of experience with this application.

4.2 Tools used for testing

In this chapter of the practical part the software and tools used for the test automation of the web application will be chosen and described. Then a testing setup will be described in more detail.

4.2.1 Selenium

First, it is necessary to choose the appropriate tool for test automation. Since the application under test is a web application, Selenium is the best tool for automation. Selenium is a family of drivers for different browsers (Firefox, Edge, Google Chrome, Internet Explorer, Safari,

Opera) and a set of client libraries in different programming languages for working with drivers. WebDriver supports Java, .Net (C#), Python, Ruby, JavaScript. These libraries are used to communicate with the driver via HTTP requests that specify the actions that the browser should take at the moment. Such actions might include commands for using a locator to find elements, for clicking buttons or links on a website page, or for parsing text on a page or in an element.

4.2.2 Jenkins

Jenkins is an open-source automation server built on the Java Virtual Machine (JVM) that supports thousands of plugins for developing, deploying, and automating software projects. Jenkins allows to organize the process of continuous integration (CI or Continuous Integration) and delivery (CD or Continuous Delivery) of a software product (permanent merging of working copies into the main development branch).

It is used to build projects from source code, run tests, and send build reports to the appropriate members of the development team.

4.2.3 Gherkin

Gherkin is a human-readable language for describing system behavior that uses indentation to define document structure. Each line starts with one of the keywords and describes one of the steps. Using Gherkin is convenient for several reasons:

- The scenarios that determine the behavior of the system are described in a simple way and can be understood by all project participants.
- Having a dictionary of available steps allows scenario variability and allows testers to write new auto-tests without resorting to code.

4.3 Test requirements

First and foremost, the IT team's major objective is to fulfill the demands and requests of its clients. Additional objectives include the implementation of software and the enhancement of working circumstances. The analysis of the requirements specified in the provided documentation and the interpretation of those requirements by programmers and testers are significant here. The quantity and complexity of customer requirements may vary based on the project's size. However, the primary purposes for using any software are usually identified and acknowledged when it is being developed. The primary objective of the

software under the test in this thesis is the creation and issuing of financial instruments that will afterward be sold on global markets. To illustrate how to write test cases and conduct subsequent manual and automated testing there were chosen the most systemic and functional requirements which are listed in the bellow table.

No.	Name	Description
1	Creation of financial instrument	Client requests to create a financial instrument
2	Edit created instrument	Client requests to have a possibility to edit the created instrument
3	Create institutions	Create an institution issuing the financial instrument

Table 2: Client requirements (Kelgenbayeva, 2023)

4.3.1 Creation of instrument

The first test requirement is the creation of different types of financial instruments. There are 5 types of it are available in the system: DEBT, EQUITY, WARRANT, UNIT, and RIGHT. Users should be able to have the possibility from the Menu to choose the link creation of an instrument. After clicking on the link system opens the tab where the user must choose "Instrument category", "Instrument group" and "Process purpose" from the dropdown menus. Once the fields were selected user can press the "Continue" button to complete the full setup of security. After pressing the "Continue" button the new window with instrument details should be visible. On this window are present different tabs, for the creation of an instrument, are used tabs "Main", "Main 2", "Agents/Issues/EB", "TEFRA/Tax", and "Income".

On the first tab "Main" the user must select provided options from the dropdown menus:

- Physical Entry
- Legal Form
- Market category
- Market group
- Physical form
- Closing date
- Maturity date
- Distribution date

44

On the second tab, "Main 2" the user must enter:

- Initial amount and currency

- Minimum and multiple settlement amounts

- Denomination values

- Issue price and currency

On the tab "Agents / Issues / EB" the user must enter the issuing agency name (institution) or its specified codes:

- Short code, Short code 2, Short code 3

- Delivery code

On the tab "TEFRA/Tax" user must select provided options from dropdown menus:

- TEFRA flag

- Taxability

On the tab "Income" the user can choose one of the "Interest type" values.

After all fields were selected and entered the system must check and validate their correctness. In order to do it the button "Default and Validate" should be enabled.

If the entered and selected values are correct system will give a message "Validation successful". If the system detects errors corresponding error message should appear. After successful validation user can save the created security by pressing the button "Save and Close". If saving was processed system will show the message "Instrument created" and will generate the ISIN code.

4.3.2   Edit created instrument

Since the user could make mistakes in the creation of instruments, the system should allow adjustments to be made to it. To do this, the user needs to find the necessary instrument in the search and go to edit mode by pressing the Edit button. Any value from a prior test request that was used to create an instrument may be changed by the system. Therefore, it was decided to change the following fields for the purpose of this thesis:

- Initial Closing date

- Initial Maturity date

- Distribution date

Once the dates will be changed it is required to check the edited instrument for validation via the button "Default and Validate" and save the performed changes.

4.3.3 Creation of institution

An institution is an issuer that creates, registers and sells securities. Corporations, investment trusts, as well as domestic or foreign governments, may be institutions. Securities like EQUITY, WARRANTS, and DEBT are made available by issuers. Users should be able to have the possibility from the Menu to choose the link creation of an institution. The user should see a new window after clicking the link, where he must enter the information listed below:

- Short code
- Institution name
- Institution short name
- Country code
- Industry
- Institution type
- Institution role

The system needs to verify that all fields were filled out correctly after they had all been chosen and entered. The "Default and Validate" button should be enabled. A message stating "Validation successful" will be displayed if the values entered and chosen are accurate. The created institution can be saved by the user by clicking "Save and Close" button after successful validation and the system will display "Institution successfully created" message.

4.4 Test cases

The following test cases were written for the above requirements, which will need to be tested manually as well as automated. Test cases are written in such a way as to cover all mentioned requirements. Of course, in practice, it is possible to create more cases, since the web application is complex and extensive and it would be difficult to add them in three test cases.

Creation of instrument – TC1
Precondition
Open the web application - Go to RDF site

No.	Test requirement	Test execution	Expected result
1	Possibility from the Menu to choose the link creation of an instrument.	Click on Menu Click on Instrument and click on the link Create	Link Create is functional and the new tab is open
2	Users must choose "Instrument category", "Instrument group" and "Process purpose" from the dropdown menus.	On the new tab select: Instrument category = DEBT, EQUITY, RIGHT, WARRANT, UNIT Instrument group = BOND, SHARES, RIGHT, CALL WARRANT, UNIT Process purpose = Settlement and Custody	Dropdown menus have proposed values and are functional
3	Once the fields were selected user can press the "Continue" button to complete the full setup of security	Press Continue button	The button is functional, and a new window is open
4	On the first tab "Main" the user must select provided options from the dropdown menus: • Physical Entry • Legal Form • Market category • Market group • Physical form • Closing date • Maturity date • Distribution date	Open the tab "Main" and select: Physical entry = Materialized Legal form = Bearer Market Category = Domestic Market group = CBF Physical form = CGN Closing date = any date Maturity date = any date Distribution date = any date	All values are present in dropdown menus and entering the date into system is possible
5	On the second tab, "Main 2" the user must enter: • Initial amount and currency • Minimum and multiple settlement amounts • Denomination values • Issue price and currency	Open the tab "Main 2" and select: Initial amount = 1 and currency = EUR Minimum and multiple amounts = 1 Denomination value =1 Issue price = 100 and currency = EUR	All values are present in dropdown menus and entering the amounts into system is possible

6	On the tab "Agents/Issues" the user must enter the issuing agency name (institution) or its specified codes: • Short code, Short code 2, Short code 3 • Delivery code	Open tab "Agents/Issues" and enter: Short code = BAYMU Short code 2 = LM Short code 3 = BOAHK Delivery code = 0	Entering the short codes into system is possible And dropdown menu for delivery code has correct value
7	On the tab "Tax" user must select provided options from dropdown menus: • TEFRA flag • Taxability	Open the tab "Tax" and select: TEFRA Flag = No Taxability = Non-taxable	All values are present in dropdown menus
8	On the tab "Income" the user can choose one of the "Interest type" values.	Open the tab "Income" and select Interest type = ZERO Coupons	Value is present in dropdown menus
9	After all fields were selected and entered the system must check and validate their correctness.	Press on "Default and Validate button"	Message "Validation is successful"
10	If saving was processed system will show the message "Instrument created" and will generate the ISIN code.	Press on "Save and close" button	Message "Instrument created", ISIN is generated

Table 3: TC1 - Creation of instrument (Kelgenbayeva, 2023)

Edit created instrument – TC2			
Precondition			
Open the web application - Go to RDF site			
No.	Test requirement	Test execution	Expected result
1	The user needs to find the necessary instrument in the search	Click on Menu Click on Instrument and click on the link View	Link View is functional, created security/instrument is found

		Search created instrument by generated ISIN	
2	Go to edit mode by pressing the Edit button.	Press "Edit" button next to found instrument	The button is functional, and a window with detailed information is open
3	Edit • Closing date • Maturity date • Distribution date	Open the tab "Main" and select: Closing date = any date Maturity date = any date Distribution date = any date	Entering the date into system is possible
4	Once the dates will be changed it is required to check the edited instrument for validation via the button "Default and Validate"	Press on "Default and Validate button"	Message "Validation is successful"
5	Save the performed changes.	Press on "Save and close" button	Message "Instrument changes saved"

Table 4: TC2 - Edit instrument (Kelgenbayeva, 2023)

Creation of institution – TC3			
Precondition			
Open the web application - Go to RDF site			
No.	Test requirement	Test execution	Expected result
1	Users should be able to have the possibility from the Menu to choose the link creation of an institution.	Click on Menu Click on Institution and click on the link Create	Link Create is functional and the new window is open
2	User must enter the information listed below: • Short code • Institution name • Institution short name • Country code • Industry • Institution type • Institution role	On the new tab enter: Short code = TEST01FR Institution name = CREATED FOR TESTING PURPOSE Institution short name = TESTING1 Country code = FR Industry code = EE	Entering the values into the system is possible

		Institution type = CORPORATE Institution role = ISSUER	
3	The system needs to verify that all fields were filled out correctly after they had all been chosen and entered. The "Default and Validate" button should be enabled	Press on "Default and Validate" button	Message "Validation is successful"
4	The created institution can be saved by the user by clicking "Save and Close" button	Press on "Save and close" button	Message "Institution successfully created"

Table 5: TC3 - Create institution (Kelgenbayeva, 2023)

4.5 Manual testing

In this part the manual testing of selected requirements were described. Three testers took part in the manual testing of the web application, which they did in accordance with the previously written test cases. Testing occurred in three test environments, or in three phases, and the time it took for the test cases to pass was recorded. The time recording procedure is as follows. The test cases were tested sequentially, and then the time of the entire cycle was summed up, i.e., the tester sequentially tests the first, second, and third test cases separately, and then these three values are added. A table was made after the recoding results are received, and the arithmetic mean was determined based on the it.

Test case	Tester A	Tester B	Tester C
TC1	147 sec	338 sec	118 sec
TC2	88 sec	91 sec	126 sec
TC3	76 sec	89 sec	52 sec
Sum for 1 cycle	311 sec	518 sec	178 sec
TC1	77 sec	111 sec	105 sec
TC2	75 sec	126 sec	83 sec
TC3	60 sec	53 sec	42 sec
Sum for cycle 2	212 sec	290 sec	230 sec
TC1	84 sec	182 sec	83 sec

TC2	59 sec	93 sec	66 sec
TC3	47 sec	56 sec	43 sec
Sum for cycle 3	190 sec	331 sec	192 sec
Arithmetic mean	**237 sec**	**379 sec**	**300 sec**

Table 6: Manual testing execution (Kelgenbayeva, 2023)

4.6 Automated testing

Since the tests that were written in previous chapter are tests focused on end-user experience it is better to utilize the Behavior Driven Development (BDD) extension rather than Test Driven Development (TDD) for automated front-end testing. TDD extension is ideal when it is needed to test one unit and no need to perform regression testing. The best thing about behavioral testing is that, unlike other testing approaches where technical specifications form the basis of the test code, it bases its tests on features and business standards. Effective dialogue and communication are the cornerstones of BDD.
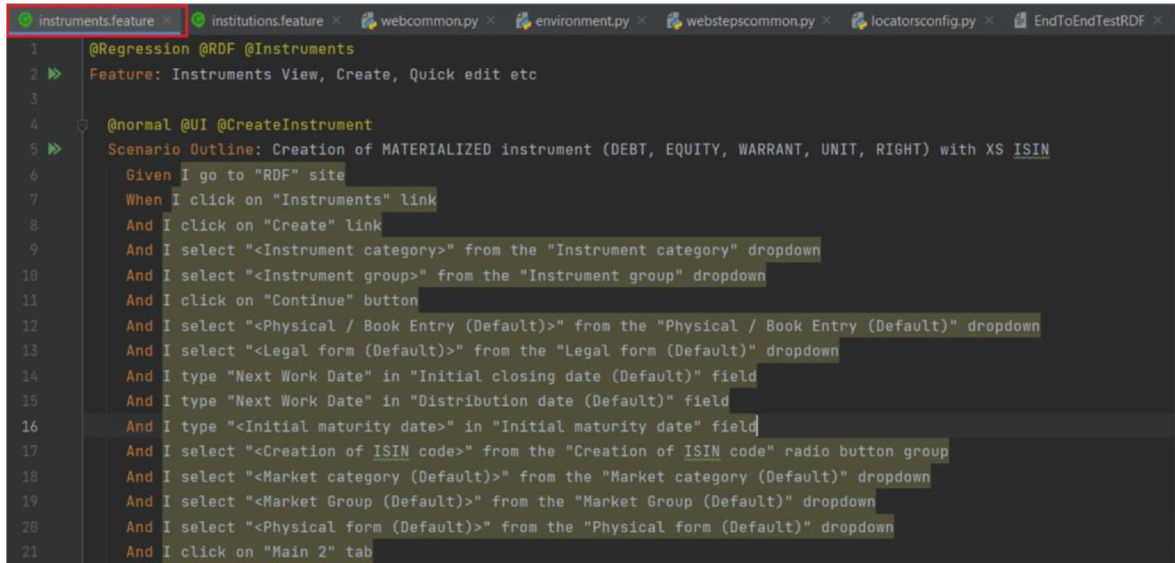
First is needed to install the following software and prerequisites on a machine running a supported version of Windows 10 in order to create automated test cases:

1) Python 3.0
2) IDE for writing the code – PyCharm
3) Gherkin plugins
3) Selenium WebDriver
4) Python Behave Framework
5) Jenkins

Using the File and New Project menu item in PyCharm, a new project must first be created. In this project, a feature file will be created after completing the installation of all prerequisites. The creation of BDD test cases can benefit greatly from the use of plain text files with the .*feature* extension known as feature files in Gherkin. The simplified Gherkin language syntaxes "Given," "When," "Then," and "And" constitute the foundation of BDD tests. Using this syntax, the steps that were described in the previous chapter about test cases will be written. One feature file can contain one or more test scenarios. That is why two .*feature* files will be created. One for Instruments named *instruments.feature* where the steps for the TC1 and TC2 scenarios will be written. And the next one is named *institutions.feature* for TC3.

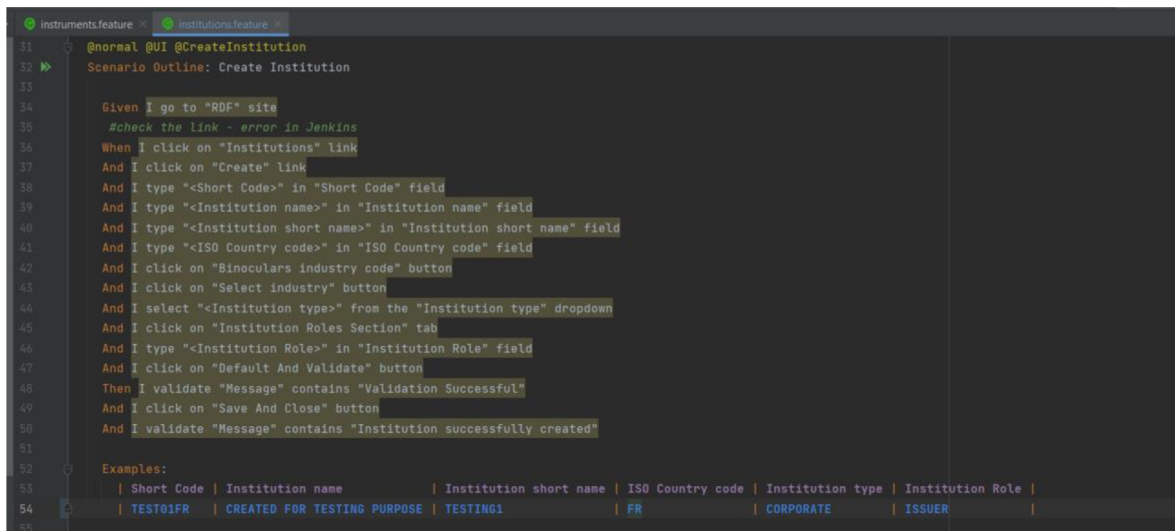### 4.6.1 Creation of Scenario steps in the Gherkin language

This code snippet contains the steps for the first test case TC1. The code starts with the "Feature" keyword which provides a brief description of the software feature. The test case's title is indicated by the @CreateInstrument and by the keyword "Scenario Outline" the description is indicated. The "Given" keyword describes the precondition "Go to RDF site". "When" and "And" keywords are used for describing steps. "Then" is used for indicating the outcome. The full code of created steps used for testing will be stored in the annex.

Figure 10: Steps in Gherkin for TC1 (Kelgenbayeva, 2023)

The same structure was applied for test case TC3 to create an institution, which is visible in the bellow figure.

Figure 11:Steps in Gherkin for TC3 (Kelgenbayeva, 2023)

4.6.2   Creation of common functions

A file with the extension *webcommon.py* must be created when all the steps from the created test cases have been written in the Gherkin language. It will have common functions that are used to test any test case. There were various Selenium WebDriver, Jenkins, and Allure libraries loaded. Checking whether a field in a browser is visible and enabled is an example of a created function. In addition to the wrapper for Selenium WebDriverWait, wrapper methods were developed for carrying out operations like opening a Selenium WebDriver, closing a Selenium WebDriver, and identifying components by XPath. An example would be the following functions to find an element and check it's visibility in the app.

```
158   def find_element(context, locator_attribute, locator_text):
159       possible_locators = ["id", "xpath", "link text", "partial link text", "name", "tag name", "class name",
160                            "css selector"]
161
162       if locator_attribute not in possible_locators:
163           raise Exception(
164               'The locator attribute {} provided is not in the approved attributes. The approved attributes are : %s '.format(
165                   locator_attribute) % possible_locators)
166
167       if (isElementPresent(context, locator_attribute, locator_text) and isElementVisible(context, locator_attribute,
168                                                                                            locator_text)):
169           try:
170               element = context.driver.find_element(locator_attribute, locator_text)
171               return element
172           except Exception as e:
173               ReportFailure(context, str(e))
174       else:
175           ReportFailure(context, "The element -" + locator_text + " not present/ visible")
```

Figure 12: Find an element with xpath (Kelgenbayeva, 2023)

```
245   def isElementVisible(context, locator_att, locator_text, wait_time=20):
246       wait = WebDriverWait(context.driver, wait_time)
247       elementList = []
248       waitForPageToBeLoaded(context)
249       try:
250           elementList = wait.until(EC.visibility_of_all_elements_located((locator_att, locator_text)))
251       except Exception as e:
252           return False
253       return True
254
```

Figure 13: Is Element visible? (Kelgenbayeva, 2023)

4.6.3   Creating Step Definitions for each Scenario Step

After the common functions such as finding elements in the application have been developed, it is needed to create another file with the python extension, which was called *webstepscommon.py*. This file will store definitions for the created steps in the Gherkin language. It is important to notice that each scenario step is assigned to a step definition, a

decorated Python function. For instance, Figure 10 demonstrates that the step "*And I select Instrument category from the Instrument Category dropdown*" was written at the eighth line of code in Gherkin; now, it is important to define this step in order to reproduce it without human interaction. To perform the action "*And I select Instrument Category from the Instrument Category dropdown*" the bellow code was used.

```python
78   @step('I select "{value}" from the "{name}" dropdown')
79   def i_select_as(context, name, value):
80       with allure.step("Selecting " + value + " from " + name + " dropdown"):
81           webcommon.get_locator(context, name)
82           webcommon.select(context, value, context.locator_type, context.locator_value)
83
```

Figure 14: Select from the dropdown menu (Kelgenbayeva, 2023)

Another example for step *"And I type Next Work date in Initial closing date field"*

```python
112  @step('I type "{text:NullableString}" in "{name}" field if "{enable_check}"')
113  @step('I type "{text:NullableString}" in "{name}" field')
114  def i_type_in_field(context, text, name, enable_check="enabled"):
115      with allure.step("Typing " + text + " in " + name + " field"):
116          if text.lower() == 'current date':
117              text = context.current_date
118          elif text.lower() == 'last work date':
119              text = context.last_workdate
120          elif text.lower() == 'next work date':
121              text = context.next_workdate
122          webcommon.get_locator(context, name)
123
124          if enable_check.lower() in ("visible") and not webcommon.isElementVisible(context, context.locator_type,
125                                                  context.locator_value, wait_time=2):
126              pass
127          elif webcommon.isElementEnabled(context, context.locator_type, context.locator_value, enable_check):
128              webcommon.type_into_element(context, text, context.locator_type, context.locator_value)
129          else:
130              pass
```

Figure 15: Type in a field (Kelgenbayeva, 2023)

4.6.4 Execution

For the execution of the newly created BDD test, Jenkins will be used. Jenkins required the setup of two systems: a Jenkins master node and a helper node to automate testing. A big advantage of Jenkins is that it can provide the tester with reports from Allure. The next Allure Report can show how the first test case TC1 for the creation of the instrument was passed. On the Behaviours tab, a list of tests that have been passed is visible. Also here the execution time is visible, it will be used later for comparison of results with manual testing.

54
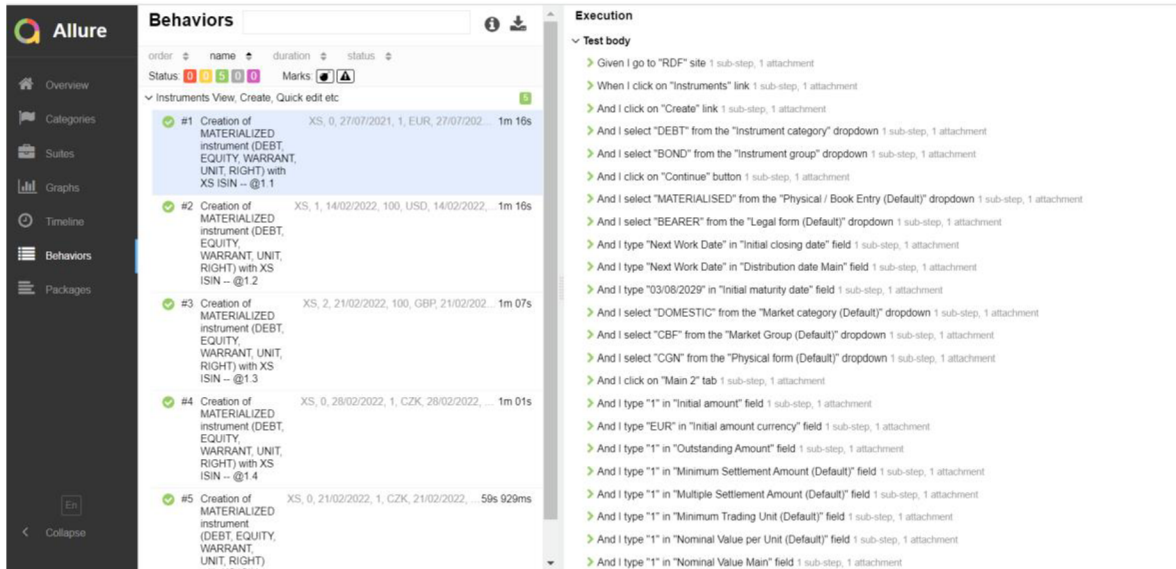
Figure 16: Allure report for TC1 (Kelgenbayeva, 2023)

The steps for TC3 that the machine carried out are shown in the following figure. Here it is possible to view more specific execution details on the right side, including the history and retires, the performed commands in the selected test, and any screenshots that were produced.
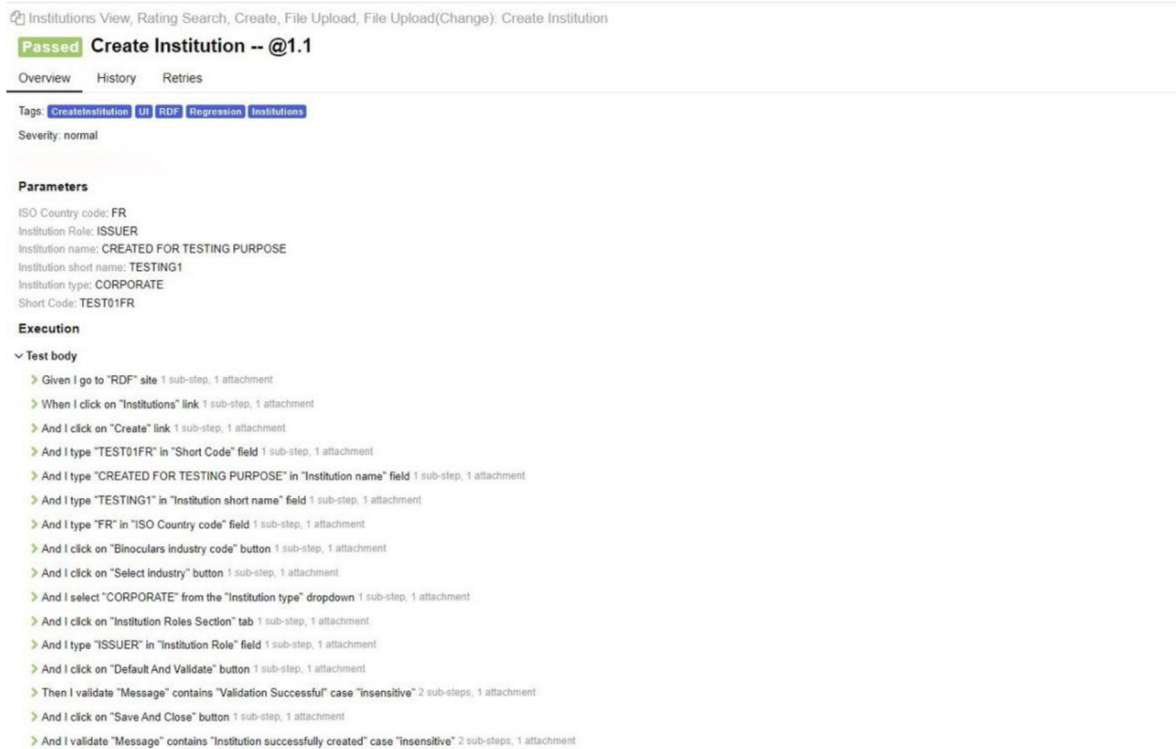
Figure 17: Allure report for TC3 (Kelgenbayeva, 2023)

The execution of the test cases was done in a similar way as for manual testing which was described in chapter 4.5 in three cycles. The bellow table is demonstrating the time range of execution and the arithmetic mean.

Test case	Automation
TC1	76 sec
TC2	81 sec
TC3	137 sec
Sum for cycle 1	291 sec
TC1	67 sec
TC2	87 sec
TC3	81 sec
Sum for cycle 2	235 sec
TC1	61 sec
TC2	96 sec
TC3	88 sec
Sum for cycle 3	245 sec
Arithmetic mean	**257 sec**

Table 7: Automated testing execution (Kelgenbayeva, 2023)

5. Results and Discussion

After completing testing and getting results it is possible to evaluate and compare the differences between manual and automated testing of this web application. The comparison will take place according to the following key criteria for all stakeholders:

- Estimated speed
- Labor cost
- Usability rating

5.1 Estimated speed

Since the execution speed of each test case was measured and obtained in the previous chapters, it will only be necessary to compare them. Three average values for manual testing and one average time value for automated testing were both achieved in the previous chapters of this thesis. It will also be important to determine one average time for manual testing for the purposes of comparison and calculate the ratio of test execution time for two testing methods.

Test execution			
Tester A	237 sec	Automated test	257 sec
Tester B	379 sec		
Tester C	300 sec		
Avg	**305 sec**	**Avg**	**257 sec**
Ratio	**1: 0,8426**		

Table 8: Speed results (Kelgenbayeva, 2023)

From the above table is visible that automated testing has a higher speed in the execution of test cases. The ratio is 1 to 0.8426 and automated test executed the test cases faster.

5.2 Labor cost

The next important comparison criteria is labor cost. This affects everyone responsible for a project. Therefore, for comparison, it is necessary to find out how much the work of individual employees in the field of manual testing and automation costs. To estimate salaries can be used sites where people can find jobs in this field. One of the most popular websites is Jobs.cz. Open positions for testers, functional or IT analysts and test automation

engineers were found on this site. Based on this information, the most often proposed salaries were revealed, which can be seen in the following table.

Position name	Month salary	Salary per hour
Manual tester	45000 kc	282 kc /h
Test automation engineer	70000 kc	437 kc /h
Functional / Test Analyst	60000 kc	375 kc /h

Table 9: Salaries in 2023 (Kelgenbayeva, 2023)

The above table also shows how much an employee earns per hour, these prices were calculated based on standard working hours, i.e. 160 hours per month. Nevertheless, it should be noted that the individual salary for each employee is different depending on the company, work experience, and place of work. The value of labor was measured by the time spent for testing. Only human resources are considered in the overall cost of manual and automated testing. The time required to analyze, create, run, validate results, and report on the test suite was estimated by the three testers who carried out manual testing in the earlier chapters. Each test analyst separately offered cost estimates that were quite similar. These three gave an average estimate of time of two hours. The creation of automated tests takes five hours average for three testers. As an outcome for manual test preparation the company will pay for test analysts 750 kc (350 kc * 2h). For automated test script preparation, the company will pay 2185 kc (437kc *5 h). The cost of the automation test won't vary because the machine will run the tests once the script is complete. However, the business will keep paying a manual tester to test execution. The cost of human testing and automated testing will consequently be equivalent if a manual tester works for an additional 3 hours and 48 minutes.

The cost difference between a manual test and an automated test is roughly represented in the graph below. The inflection point, which occurs at 3h 48 minutes of test repetitions, demonstrates when automated testing will become more affordable than manual testing. This graph is valid under the following conditions:

1. An analyst writes and prepares test cases by analyzing test requirements
2. An automated tester writes and sets up a script
3. The application under test does not change

If we convert 3 hours and 48 minutes of running tests into the number of repetitions, the manual test's 44th cycle will indicate the breaking point. This means that a manual test is

more profitable if the number of cycles does not exceed 44. More repetitions will mean higher costs compared to an automatic scenario. In 3h 48', the automatic test will pass 53 cycles.
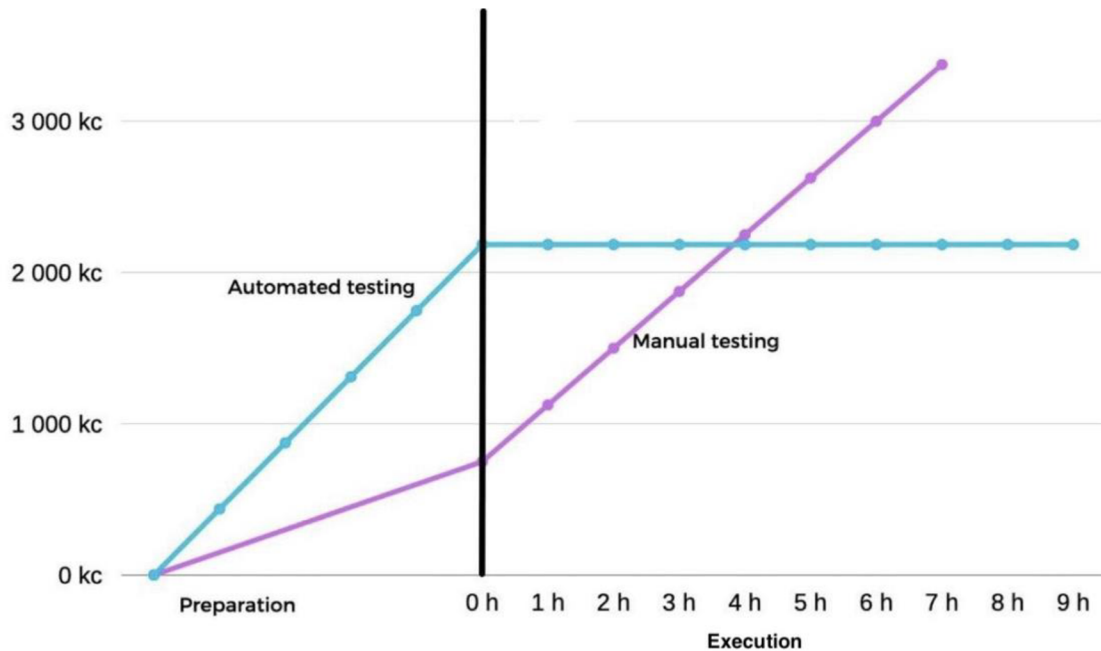
Figure 18: Labor cost (Kelgenbayeva, 2023)

The following table shows the costs of testing for the number of cycles for TC1, TC2 and TC3.

Cycle	Manual	Automated
1- 11 cycles	1125 kc	2185 kc
12-23 cycles	1500 kc	2185 kc
24-35 cycles	1875 kc	2185 kc
36 – 47 cycles	2250 kc	2185 kc

Table 10: Cost per cycles (Kelgenbayeva, 2023)

5.3 Usability rating

The convenience of using an automated test depends on how the given requirements will be tested. The machine is faster than a manual test if testing speed is the primary factor, as was already mentioned before, but the difference is not very significant. The time ratio was 1:0.84. The number of repeats will depend on the parameter's definition of cost. However, because the automated tests were created using BDD technologies and in the Gherkin language, it would be possible to create a basis of frequently used steps and then use them

without having to change the Python code for common functions. It could significantly reduce the time and facilitate the work of the tester.

The next important factor is that usually the scope of automation is limited to testing user behavior, which is easy to imitate. For backend testing, where application databases are tested, the BDD method still works well and is still useful, but not for server testing. Applying such a backend test would require effort from people with more in-depth programming knowledge and another testing tools and approaches.

Next important aspect of the usability of test automation is the stage of software development. If the web application is not yet fully developed or frequent changes are made to the main functions of the application, then the prepared tests would have to be modified. Test automation produces more open and visible test results for all parties involved. This is made possible by the report tool Allure, which provides results not only in the form of a log but also in the form of a report, which is more visually attractive. Also, Allure produces test-result statistics which can be used by test managers for further planning and organization of testing processes. Fixing the results with manual testing takes more time and afford than with automation.

5.4 Discussion

RQ1: Can the company fully migrate from manual to automated testing of the chosen application?

Since was mentioned that this application is often subject to various changes so the issued instruments comply with all legal norms the full migration is not feasible. It will likely take more time to automated testing than manual testing to test new requirements because prepared scripts will need to be amended or new ones created. Additionally, if the test cycle count does not surpass 44 cycles, automated testing will be more expensive. This means that the number of test cases for code change should be more than 132 tests of similar complexity as described above. Furthermore, it should be mentioned that if new fields or tabs are created and added to the application, the Gherkin language script can only be created after the development is complete, which will prolong the process as a whole development. This will occur as a result of front-end testing depending on XPath. It is impossible to complete the test scripts if there is no XPath accessible. If this application is also interconnected with others and it will be necessary to test the E2E flow, then it will be required to create more

complex test cases that cannot be automated. Or, when testing rare scenarios, manual testing will also be necessary.

The company is encouraged to combine manual and automated testing for this application in order to achieve the best quality of the software being created. Since with manual testing, complex and single tests can be performed, and with automation, scripts can be created for extensive projects.

RQ2: How can this application benefit from automated testing?

It should be mentioned that using test automation for regression testing would be the best option for this web-application. Regression testing evaluates all the application's primary features, and its outcomes will provide quick assessment of whether the application's core functionality has been harmed by the changes made. Regression testing can also be carried out as frequently as code updates are made without necessitating several manual tester evaluations. Automation will cost more and take longer to write a script to test new modifications to the application. For applications or parts of applications that rarely change, automated testing is more appropriate. In his study, the author Pekka Aho from the VTT Technical Research Center of Finland also came to this opinion and put manual testing in second place after automated testing in the list of the most suitable methods for conducting regression testing. (Aho, 2016) This can be explained by the fact that if the application changes frequently, it will become more necessary to do regression testing, which will consequently take more time for manual testers to complete. An organization can set up regression tests to run as often as needed with the help of automated testing. Since regression testing requires more repetitive steps, the tester may be tempted to skip some of them, and with automation there is no need to skip any.

RQ3: Is automated testing suite less labor cost than manual testing?

Automated testing costs more if the number of testing cycles does not exceed 44 or 3 hours 48 minutes. It makes sense to use automated testing rather than manual testing if a large project will have an impact on the application and it is obvious from a testing perspective that the number of test cases for front end functionalities would surpass 44 cycles or 132 test cases. Automated testing will be less expensive in this case. And company will be charged to approx. 2185 kc.

To compare the results obtained in this thesis, similar research by University of Azteca and University of Central Nicaragua was studied, where the authors of the scientific article also concluded that the amount of money saved when using an automated test case would be 155,95X for 158 hours of testing, where X is the amount of money paid to the tester. Based on this, it can be said that the use of automated scripts will help increase efficiency if the test cases are monotonous, often repeated and time taking, approx. 158h. (Asfaw, 2015) The inflection point of manual and automated testing will differ for each application, but researches show that in the long run, with reoccurred test cases, it is preferable to employ automated testing so the company may conserve time and money.

6. Conclusion

This thesis consists of two main parts: theoretical and practical. In the first part of this work, a literature review was written, in which the testing concept and the history of testing were touched upon. In this part of the thesis the advantages and disadvantages of several SDLC models were derived, and each stage in the model was explained in detail.

Since in the modern world there are many different types of tests, it was necessary to study them and understand their main classifications, which was also described in the theoretical part of the work. In the practice it was necessary to test the selected application by automated testing that is why this topic was also deeply explored in the literature review.

The knowledge gained in the theoretical part was used in the practical part of this work. The main goal of the thesis was a comparison of the automated and manual testing of a web application for shares trading company. The application that was tested and the test automation tools were both described in this section of the work. Following the selection of the tools, test cases were created based on the description and analyzing of the application's test requirements. Three testers manually tested these cases, and the time taken to complete the tests successfully was documented in a table.

After receiving the results of manual testing, the test steps were written in the Gherkin language, as well as the main functions in the Python language for test automation. The written scripts were extracted in the Jenkins application and a report was received from Allure. And the results of the automated test were saved as a table. After identifying all the necessary results, a comparison was made according to three parameters: estimated speed, labor cost and usability rating.

From the comparisons obtained, it was revealed that test automation is more suitable for applications that rarely change or for regression testing, where it is necessary to check the main functionality that should not be broken by new changes in the code. Comparing the labor cost, it was revealed that the automation test for the company will be beneficial if the number of test cycles exceeds 44. In discussion part the research questions of the practical part were answered, and it was determined that the chosen application could not fully migrate

from automated testing to manual testing without increasing costs to the business. However, it could still benefit from automated testing by using it for regression testing.

7. Bibliography

Adel Alshamrani, Abdullah Bahattab. 2015. *A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model.* s.l. : IJCSI International Journal of Computer Science Issues, 2015. SSN (Online): 1694-0784.

Anywhere, Epam. 2021. Proglib. [Online] May 15, 2021. [Cited: September 2, 2022.] https://proglib.io/p/sdlc-modeli-kak-vybrat-pravilnyy-podhod-k-razrabotke-i-ne-zavalit-proekt-2021-05-15.

Aho Pekka. 2016. *Evolution of Automated testing of Software System Through the Graphical User Interface.* s.l. :The First International Conference on Advances in Computation, 2016.

Barus, Arlinta Christy. 2019. *The implementation of ATDD and BDD from Testing Perspectives.* s.l. : Journal of Physics, 2019. 1175 012112.

Chauhan, Vinod Kumar. *Smoke testing .* 2014. s.l. : International Joirnal of Scientific and Research Publications , 2014. 2250-3153.

Daniel L Asfaw. January 2015. *Benefits of Automated Testing Over Manual Testing.*: International Journal of Innovative Research in Information Security.2015. 2349-7017.

Dhir, Poonam Priya. *Software Testing Strategies and Methodologies .* 2016. s.l. : International Journal of Advanced Research Trends in engineering and Technology, 2016. 2394-3777.

Enterprise, Digital. 2021. Cleverics. [Online] 2021. https://cleverics.ru/digital/2021/01/sw-testing-automation/.

F.L.Morris, C.B.Jones. 1984. *An Early Program Proof by Alan Turing.* s.l. : Annals of the History of Computing, 1984.

Hayes, Linda G. 2004. *The automation testimg handbook.* s.l. : Software Testing Institute, 2004. 0-9707465-0-4.

JavaTpoint. *Java Tpoint.* [Online] [Cited: August 14, 2021.] www.javatpoint.com. BCT. 2021. What is Software Development Life Cycle? *https://bcodestech.com.* [Online] November 10, 2021. [Cited: August 18, 2022.]

Kent Beck, Mike Beedle. 2001. Manifesto for Agile Software Development. [Online] 2001. [Cited: August 31, 2022.] https://agilemanifesto.org.

Khan, Mohd.Ehmer. *A Comparative Study of White Box, Black Box and Grey Box testing techniques.* 2012. s.l. : International Journal of Advanced Computer Science and Applications , 2012, Vols. Vol.3,No.6. 2156-5570.

Kulikov, Svyatoslav. 2020. *Sotware testing* . Minsk : Chetyre chetverti , 2020. 978-985-581-362-1.

Maneela Tuteja, Gaurav Dubey. 2012. *A Research Study on importance of Testing and Quality Assurance in Software Development Life Cycle (SDLC) Models.* s.l. : International Journal of Soft Computing and Engineering (IJSCE), 2012. ISSN: 2231-2307.

Manpreet Kaur, Rupinder Singh. *A Review of Software Testing Techniques.* s.l. : International Research Publication House. ISSN 0974-2174.

Marian STOICA, Marinela MIRCEA, Bogdan GHILIC-MICU. 2013. *Software Development: Agile vs. Traditional.* s.l. : Informatica Economică, 2013. ISSN 14531305/17.4.2013.06.

Moe, Myint Myint. June 2019. *Comparative Study of Test-Driven Development (TDD), Behavior-Driven Development (BDD) and Acceptance Test–Driven Development (ATDD).* s.l. : International Journal of Trend in Scientific Research and Development, June 2019. 2456- 6470.

Myers, Glenford J. 2004. *The Art of Software Testing.* Canada : John Wiley & Sons, Inc., Hoboken, New Jersey., 2004.

Naor, Adam. 2020. What is SDLC? Stages, methodology and processes of the software life cycle. *Habr.* [Online] October 20, 2020. [Cited: September 2, 2022.] https://habr.com/ru/company/dcmiran/blog/521718/.

Panthers, Asphalt. 2016. History of Software Testing. *ASPHALT PANTHERS Software Quality, Computer Scientist.* [Online] February 16, 2016. [Cited: August 31, 2022.] http://www.asphaltpanthers.com/2016/02/16/history-of-software-testing-2/.

Prof. Seema Suresh Kute, Prof. Surabhi Deependra Thorat Assistant Professor. 2014. *A Review on Various Software Development Life Cycle (SDLC) Models.* s.l. : International Journal of Research in Computer and Communication Technology, 2014. ISSN (Online) 2278- 5841.

Prytulenets, Alesya. 2022. A Brief History of Software Testing. *DoG QBlog* . [Online] February 22, 2022. [Cited: August 31, 2022.] https://dogq.io/blog/a-brief-history-of-software-testing/.

Ritu Jain, Ugrasen Suman. 2015. *A Systematic Literature Review on Global Software Development Life Cycle.* s.l. : ACM SIGSOFT Software Engineering Notes, 2015. DOI:10.1145/2735399.2735408.

Singh, Dr.Sanjay Kumar. 2019. *Software testing* . New Delhi : Vandana Publications Lucknow, 2019. 978-81-941110-6-1.

Singh, Rohan. 2022. The Testing Pyramid: Simplified for One and All. *www.headspin.io.* [Online] May 23, 2022. https://www.headspin.io/blog/the-testing-pyramid-simplified-for-one-and-all.

Testengineer. 2022. Testengineer. [Online] July 21, 2022. [Cited: August 25, 2022.] https://testengineer.ru/chto-takoe-staticheskoe-dinamicheskoe-testirovanie/.

TestMatick. 2019. TestMatick. [Online] December 24, 2019. [Cited: August 25, 2022.] https://testmatick.com/ru/staticheskoe-i-dinamicheskoe-testirovanie/.

Wolf Stanislav. 2019. *Selecting the type of testing the user interface based on the requirements for the interface design.* : Estonian Entrepreneurship University of Applied Sciences. 2019.

Yu Beng Leau, Wooi Khong Loo, Wai Yip Tham and Soo Fun Tan. 2012. *Software Development Life Cycle AGILE vs Traditional Approaches.* Singapore : IACSIT Press, 2012.

8. Annex

Gherkin steps for TC1

```gherkin
@Regression @RDF @Instruments
Feature: Instruments View, Create, Quick edit etc


  @normal @UI @CreateInstrument
  Scenario Outline: Creation of MATERIALIZED instrument (DEBT, EQUITY,
WARRANT, UNIT, RIGHT) with XS ISIN
    Given I go to "RDF" site
    When I click on "Instruments" link
    And I click on "Create" link
    And I select "<Instrument category>" from the "Instrument category"
dropdown
    And I select "<Instrument group>" from the "Instrument group"
dropdown
    And I click on "Continue" button
    And I select "<Physical / Book Entry (Default)>" from the "Physical /
Book Entry (Default)" dropdown
    And I select "<Legal form (Default)>" from the "Legal form (Default)"
dropdown
    And I type "Next Work Date" in "Initial closing date (Default)" field
    And I type "Next Work Date" in "Distribution date (Default)" field
    And I type "<Initial maturity date>" in "Initial maturity date" field
    And I select "<Creation of ISIN code>" from the "Creation of ISIN
code" radio button group
    And I select "<Market category (Default)>" from the "Market category
(Default)" dropdown
    And I select "<Market Group (Default)>" from the "Market Group
(Default)" dropdown
    And I select "<Physical form (Default)>" from the "Physical form
(Default)" dropdown
    And I click on "Main 2" tab
    And I type "<Initial amount>" in "Initial amount" field
    And I type "<Initial amount currency>" in "Initial amount currency"
field
    And I type "<Outstanding Amount>" in "Outstanding Amount" field
    And I type "<Minimum Settlement Amount (Default)>" in "Minimum
Settlement Amount (Default)" field
    And I type "<Multiple Settlement Amount (Default)>" in "Multiple
Settlement Amount (Default)" field
    And I type "<Minimum Trading Unit (Default)>" in "Minimum Trading
Unit (Default)" field
    And I type "<Nominal Value per Unit (Default)>" in "Nominal Value per
Unit (Default)" field
    And I type "<Nominal Value>" in "Nominal Value" field
    And I type "<Issue price>" in "Issue price" field
    And I click on "Agents" tab
    #Short code = ISSUER
    And I type "<Short Code>" in "Short Code" field
    #Short code 2 = LEADMGR
    And I type "<Short Code 2>" in "Short Code 2" field
    #Short code 3 = DEPOSITORY
    And I type "<Short Code 3>" in "Short Code 3" field
    And I select "<Delivery Code>" from the "Delivery Code" dropdown
    And I click on "TEFRA/Tax" tab
    And I select "<TEFRA Flag>" from the "TEFRA Flag" dropdown
    And I select "<Taxability (Default)>" from the "Taxability (Default)"
```

```
dropdown
    And I click on "Income" tab
    And I select "<Interest Type>" from the "Interest Type" dropdown
    And I click on "Default And Validate" button
    Then I validate "Message" contains "Validation Successful"
    And I click on "Save And Close" button
  # And I verify there is potential duplicate continue button
    # Then I click on "Continue" button #
    Then I validate "Message" contains "Instrument created"
    Examples:
      | Instrument category | Instrument group | Physical / Book Entry
(Default) | Legal form (Default)     | Initial closing date |
Distribution date (Default) | Initial maturity date |Creation of ISIN
code| Market category (Default) | Market Group (Default) | Physical form
(Default) | Initial amount | Initial amount currency | Outstanding Amount
| Minimum Settlement Amount (Default) | Multiple Settlement Amount
(Default) | Minimum Trading Unit (Default) | Nominal Value per Unit
(Default) | Nominal Value | Issue price | Short Code | Short Code 2 |
Short Code 3 | Delivery Code | TEFRA Flag | Taxability (Default) |
Interest Type |
| DEBT BOND| MATERIALISED| BEARER| 27/07/2021| 27/07/2021| 03/08/2029
| XS | DOMESTIC| CBF| CGN| 1 | EUR| 1 | 1 | 1| 1 | 1 | 1 | 100 | BAYMU
| LM | BOAHK | 0 | NO | NON-TAXABLE | ZERO COUPON|

| EQUITY | ORDINARY SHARE| MATERIALISED | BEARER/REGISTERED | 14/02/2022
| 14/02/2022 | 28/02/2029 | XS | FOREIGN | INTERNATIONAL | CGN | 100
| USD | 1 | 10 | 10 | 1 | 1 | 10 | 150 | BOSPRUS | LM | BOAHK| 1 | NO
| NON-TAXABLE | ZERO COUPON |

| WARRANT | CALL WARRANT | MATERIALISED | REGISTERED | 21/02/2022
| 21/02/2022 | 28/02/2029| XS| GLOBAL | CBL | CGN | 100 | GBP
| 1 | 1 | 1  | 1 | 1 | 1 | 150 | BONGRFR    | LM | 6E | 2 | NO | NON-
TAXABLE | ZERO COUPON |

| UNIT | UNIT | MATERIALISED | BEARER DEPOSITORY RECEIPT | 28/02/2022
| 28/02/2022 | 28/02/2029 | XS | EURO | CBM | GLOBAL | 1 | CZK
| 1 | 1 | 1  | 1 | 1 | 1  | 100  | ATLCOSE | LM | BOTMIGB | 0
| NO | NON-TAXABLE | ZERO COUPON |

| RIGHT | BONUS RIGHT | MATERIALISED | BEARER DEPOSITORY RECEIPT |
21/02/2022 | 21/02/2022 | 28/02/2029  | XS | INTERNATIONAL |
INTERNATIONAL | GLOBAL | 1 | CZK | 1 | 1  | 1 | 1 | 1 | 1 | 100 | ATLCOSE
| LM | BOTMIGB | 0 | NO | NON-TAXABLE  | ZERO COUPON    |
```

Gherkin steps for TC2

```
@normal @UI @EditInstrument @RDF-1055
Scenario Outline: Modify an instrument Initial closing date and
Distribution date

  Given I go to "RDF" site
  When I click on "Instruments" link
  And I click on "View" link
  And I type "<Isin Code>" in "Isin Code" field
  And I type "Next Work Date" in "Maturity date from" field
  And I type "<Maturity date to>" in "Maturity date to" field
  And I type "<Instrument status>" in "Instrument status" field
  And I click on "Search" button
  And I wait for "10" seconds
```

```
  And I click on "Edit Detail" button
  And I type "Next Work Date" in "Initial closing date" field
  And I type "Next Work Date" in "Distribution date (Default)" field
  And I type "Next Work Date" in "Actual closing date (Default)" field
  And I click on "Default And Validate" button
  Then I validate "Message" contains "Validation Successful"
  And I click on "Save And Close" button
  And I validate "Message" contains "Changes applied for instrument"

  Examples:
    | Isin Code    | Instrument status | Maturity date to |
    | %            | CREATED           | 31/12/9999       |
    | DE%          | ACCEPTED          | 31/12/9999       |
    | GB%          | ACTIVATED         | 31/12/9999       |
```

Gherkin steps for TC3

```
@normal @UI @CreateInstitution
  Scenario Outline: Create Institution

    Given I go to "RDF" site
     #check the link - error in Jenkins
    When I click on "Institutions" link
    And I click on "Create" link
    And I type "<Short Code>" in "Short Code" field
    And I type "<Institution name>" in "Institution name" field
    And I type "<Institution short name>" in "Institution short name"
field
    And I type "<ISO Country code>" in "ISO Country code" field
    And I click on "Binoculars industry code" button
    And I click on "Select industry" button
    And I select "<Institution type>" from the "Institution type"
dropdown
    And I click on "Institution Roles Section" tab
    And I type "<Institution Role>" in "Institution Role" field
    And I click on "Default And Validate" button
    Then I validate "Message" contains "Validation Successful"
    And I click on "Save And Close" button
    And I validate "Message" contains "Institution successfully created"

    Examples:
| Short Code | Institution name | Institution short name | ISO Country
code | Institution type | Institution Role |
| TEST01FR   | CREATED FOR TESTING PURPOSE | TESTING1 | FR
| CORPORATE | ISSUER |
```

Log file

Started by user Almira Kelgenbayeva

[Pipeline] Start of Pipeline

[Pipeline] node

Running on TestAutomation-sim-onprem in /svc/selenium/jenkins-

agent/workspace/testing/RDF/End_To_End_Testing

+ source /svc/selenium/jenkins-agent/workspace/testing/venv_rdh/bin/activate

++ deactivate nondestructive

++ '[' -n '' ']'

++ '[' -n '' ']'

++ '[' -n /usr/bin/sh -o -n '' ']'

++ hash -r

++ '[' -n '' ']'

++ unset VIRTUAL_ENV

++ '[' '!' nondestructive = nondestructive ']'

++ _OLD_VIRTUAL_PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin

++ PATH=/svc/selenium/jenkins-

+ cd TestAutomationBDDFramework

+ python3 -m runner --job_dir testing/RDF/End_To_End_Testing --run_allure true '--behave_options=-t Regression -t CreateInstrument -D environment=June2022-BAT -D browser=Firefox -D allstepexecution=No -D video=Yes -D screenshot=Yes'
/svc/selenium/jenkins-agent/workspace/testing/venv_rdf/lib64/python3.6/site-packages/azure/storage/blob/_shared/encryption.py:15: CryptographyDeprecationWarning: Python 3.6 is no longer supported by the Python core team. Therefore, support for it is deprecated in cryptography and will be removed in a future release.
 from cryptography.hazmat.backends import default_backend

1 feature passed, 0 failed, 23 skipped

5 scenarios passed, 0 failed, 17 skipped

190 steps passed, 0 failed, 295 skipped, 0 undefined

Took 5m14.900s

Allure report was successfully generated.

Creating artifact for the build.

Artifact was added to the build.

Finished: SUCCESS

Python code

```python
from tabulate import tabulate
import ast
from deepdiff import DeepDiff
from selenium import webdriver
from selenium.webdriver import DesiredCapabilities
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
```

```python
from BDDCommon.CommonConfigs import urlconfig
import time
from time import sleep
import logging as logger
import allure
from allure_commons.types import AttachmentType
from datetime import datetime
from selenium.webdriver.support.ui import Select
from BDDCommon.CommonConfigs import locatorsconfig
import os
import pathlib
import json
import pandas as pd
from selenium.webdriver.common.keys import Keys


def find_element(context, locator_attribute, locator_text):
    possible_locators = ["id", "xpath", "link text", "partial link text",
"name", "tag name", "class name",
                         "css selector"]

    if locator_attribute not in possible_locators:
        raise Exception(
            'The locator attribute {} provided is not in the approved
attributes. The approved attributes are : %s '.format(
                locator_attribute) % possible_locators)

    if (isElementPresent(context, locator_attribute, locator_text) and
isElementVisible(context, locator_attribute,

locator_text)):
        try:
            element = context.driver.find_element(locator_attribute,
locator_text)
            return element
        except Exception as e:
            ReportFailure(context, str(e))
    else:
        ReportFailure(context, "The element -" + locator_text + " not
present/ visible")


#
===============================================================================
=============#
def type_into_element(context_or_element, input_value, locator_att,
locator_text):
    if isinstance(context_or_element,
webdriver.remote.webelement.WebElement):
        element = context_or_element
    else:
        element = find_element(context_or_element, locator_att,
locator_text)

    if (element):
        element.send_keys(Keys.CONTROL + "a")
        element.send_keys(input_value)
        TakeScreenShot(context_or_element)
```

```python
def click(context_or_element, locator_att, locator_text, param=None):
    if isinstance(context_or_element,
webdriver.remote.webelement.WebElement):
        element = context_or_element
    else:
        element = find_element(context_or_element, locator_att,
locator_text)

    if (element.get_attribute('onclick') == 'return false;'):
        pass
    else:
        element.click()
        TakeScreenShot(context_or_element)


def select(context_or_element, input_value, locator_att, locator_text):
    if isinstance(context_or_element,
webdriver.remote.webelement.WebElement):
        element = context_or_element
    else:
        element = find_element(context_or_element, locator_att,
locator_text)
    if (element):
        select = Select(element)
        select.select_by_visible_text(input_value)
        TakeScreenShot(context_or_element)


def element_contains_text(context_or_element, expected_text, locator_att,
locator_text, case_sensitive=False):
    if isinstance(context_or_element,
webdriver.remote.webelement.WebElement):
        element = context_or_element
    else:
        element = find_element(context_or_element, locator_att,
locator_text)

    context_or_element.element_text = element.text
    if not case_sensitive:
        if expected_text.lower() in element.text.lower():
            # TakeScreenShot(context_or_element)
            return True
        else:
            return False
    else:
        return True if expected_text in element.text else False
```

Python code for common steps

```python
from behave import step, register_type
import parse
from BDDCommon.CommonFuncs import webcommon
import allure
import json
import time


@step('I go to "{page}" site')
def i_go_to_page(context, page):
```

```python
        webcommon.go_to(context, page)


@step('I am on new tab')
def i_am_on_new_tab(context):
    with allure.step("Switching to new tab"):
        webcommon.switch_tab(context)



@step('I click on "{name}" link')
@step('I click on "{name}" button')
def i_click_on(context, name, enable_check="enabled"):
    with allure.step("Clicking - " + name):
        webcommon.get_locator(context, name)
        if enable_check.lower() in ("visible") and not
webcommon.isElementVisible(context, context.locator_type,

context.locator_value, wait_time=2):
            pass
        else:
            webcommon.click(context, context.locator_type,
context.locator_value)


@step('I select "{value}" from the "{name}" dropdown')
def i_select_as(context, name, value):
    with allure.step("Selecting " + value + " from " + name + "
dropdown"):
        webcommon.get_locator(context, name)
        webcommon.select(context, value, context.locator_type,
context.locator_value)


@step('I type "{text:NullableString}" in "{name}" field')
def i_type_in_field(context, text, name, enable_check="enabled"):
    with allure.step("Typing " + text + " in " + name + " field"):
        if text.lower() == 'current date':
            text = context.current_date
        elif text.lower() == 'last work date':
            text = context.last_workdate
        elif text.lower() == 'next work date':
            text = context.next_workdate
        webcommon.get_locator(context, name)

        if enable_check.lower() in ("visible") and not
webcommon.isElementVisible(context, context.locator_type,

context.locator_value, wait_time=2):
            pass
        elif webcommon.isElementEnabled(context, context.locator_type,
context.locator_value, enable_check):
            webcommon.type_into_element(context, text,
context.locator_type, context.locator_value)
        else:
            pass
```

74