Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Technologies

Bachelor Thesis

A comparison of old and new software testing methods

Yelaman Nurdauletov

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

BACHELOR THESIS ASSIGNMENT

Yelaman Nurdauletov

Informatics

Thesis title

A comparison of old and new software testing methods

Objectives of thesis

This objective of this thesis is to describe how new methods of software testing differ from older ones and why the latest methods for testing should be used. The thesis considers in detail the process of new testing methods as a component of the software development quality assurance process, as well as theoretically substantiating the main advantages of new testing methods for a particular software and testing them practically, on the basis of new tested methods.

Methodology

The thesis will be composed of a theoretical and practical part. The first, based on a review of the literature, will define 'software testing' and exactly what the different methods involve. Various methods and strategies of software testing will be examined in detail and a comparative analysis of new and old methods presented. The practical part will demonstrate the main problems preventing the full maximization of some new testing methods, and ways of solving and improving them will be shown.

The proposed extent of the thesis

30-40 pages

Keywords

software testing, testing methods, latest methods, testing improvements

---

Recommended information sources

BADGETT, Tom, MYERS, G, J, SANDLER, Corey, 2011. The Art of Software Testing 3rd Edition, Wiley. 256 p. ISBN-13 : 978-1118031964.

CERN, K, FALK, Jack, NGUYEN, Hung Q, 1999. Testing Computer Software, 2nd Edition. Wiley. 480 p. ISBN-13 : 978-0471358466.

IBM(International Business Machines).Find software errors and verify that an application or system is fit for use [online) 19 August 2020. https://www.ibm.com/topics/software-testingne], Accessed 24 May 2022.

Louise Tamres, "Introduction to Software Testing". Addison-Wesley (April 1, 2002) ,304 pages, ISBN-13: 978-0201719741.

MCGREGOR, J, D, 2001. A Practical Guide to Testing Object-Oriented Software (Addison-wesley Object Technology Series) . Addison Wesley. 416 p. ISBN-13 : 978-0201325645.

PATTON, Ron, 2005. Software Testing, 2nd edition. 408 p. ISBN-13 : 978-0672327988.

Expected date of thesis defence

2022/23 SS – FEM

The Bachelor Thesis Supervisor

John McKeown

Supervising department

Department of Languages

Electronic approval: 13. 6. 2022

PhDr. Mgr. Lenka Kučírková, Ph.D.

Head of department

Electronic approval: 27. 10. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Dean

Prague on 15. 03. 2024

Declaration

I declare that I have worked on my bachelor thesis titled " A comparison of old and new software testing methods" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the thesis does not break any copyrights.

In Prague on 15.03.2024 _____

Acknowledgement

I would like to thank John McKeown for his advice and support during the work on this thesis. I also would like to thank my friends and my family who have been supporting me my whole life.

A comparison of old and new software testing methods

Abstract

The paper discusses software testing methods.

An analysis of the literature devoted to the consideration of software testing methods was carried out.

A historical overview of old or "classical" testing methods is provided. The features of old methods, their advantages and disadvantages are considered. New software testing methods that have emerged in the last two decades are explored. The positive and negative aspects of the new methods are discussed in detail.

The theoretical foundations on which software testing methods are based are considered, and a classification of these methods is also made.

A comparative analysis of old and new software testing methods was performed. Practical examples are provided to illustrate the strengths and weaknesses of old and new methods. Improvements to existing software testing methods are proposed. These improvements will increase efficiency and allow you to find all the problem areas of the software under test. Recommendations are provided to improve existing software testing methods. The proposed improvements were tested in practice.

Keywords: SOFTWARE, SOFTWARE TESTING METHOD, OLD TESTING METHOD, MODERN TESTING METHOD, COMPARATIVE ANALYSIS, UNIT TESTING, INTEGRATION TESTING, SYSTEM TESTING, INCREMENTAL MODEL, SPIRAL MODEL. VERIFICATION, VALIDATION

Srovnání starých a nových metod testování softwaru

Abstrakt

Článek pojednává o metodách testování softwaru.

Byla provedena analýza literatury věnované úvahám o metodách testování softwaru.

Je uveden historický přehled starých nebo "klasických" testovacích metod. Zvažují se vlastnosti starých metod, jejich výhody a nevýhody.

Jsou zkoumány nové metody testování softwaru, které se objevily v posledních dvou desetiletích. Podrobně jsou diskutovány pozitivní a negativní aspekty nových metod.

Jsou zvažovány teoretické základy, na kterých jsou založeny metody testování softwaru, a je také provedena klasifikace těchto metod. Byla provedena srovnávací analýza starých a nových metod testování softwaru. Jsou uvedeny praktické příklady, které ilustrují silné a slabé stránky starých a nových metod.

Jsou navržena vylepšení stávajících metod testování softwaru. Tato vylepšení zvýší efektivitu a umožní vám najít všechny problémové oblasti testovaného softwaru. Jsou poskytnuta doporučení ke zlepšení stávajících metod testování softwaru. Navržená vylepšení byla ověřena v praxi.

Klíčová slova: SOFTWARE, METODA TESTOVÁNÍ SOFTWARU, STARÁ TESTOVACÍ METODA, MODERNÍ TESTOVACÍ METODA, SROVNÁVACÍ ANALÝZA, TESTOVÁNÍ JEDNOTEK, TESTOVÁNÍ INTEGRACE, TESTOVÁNÍ SYSTÉMU, PŘÍSTUPOVÝ MODEL, SPIRÁLOVÝ MODEL. OVĚŘENÍ, VALIDACE

Table of content

1 Introduction

Software testing arose along with the programs themselves, because no one needs a program that works incorrectly.

Every day in our work we come across the rather abstract concept of “software quality” and if you ask a tester or programmer “what is quality?”, then everyone will have their own interpretation. Let's consider the definition of "software quality" in the context of international standards [1]:

- Software quality is the degree to which the software has the required combination of properties.
- Software quality is the collection of software characteristics related to its ability to satisfy stated and intended needs.

Software quality characteristics

Functionality [2] is determined by the ability of the software to solve problems that correspond to the recorded and expected needs of the user, under given conditions for using the software. This characteristic means that the software operates properly and accurately, is interoperable, meets industry standards, and is protected from unauthorized access.

Reliability is the ability of software to perform required tasks under specified conditions over a specified period of time or a specified number of operations. The attributes of this characteristic are the completeness and integrity of the entire system, the ability to independently and correctly recover from failures, and fault tolerance.

Usability is the ability of software to be easily understood, learned, used, and attractive to the user.

Efficiency is the ability of the software to provide the required level of performance, in accordance with the allocated resources, time and other specified conditions.

Maintainability is the ease with which software can be analyzed, tested, and changed to correct defects to implement new requirements, to facilitate further maintenance and adaptation to the existing environment.

Portability [4] – characterizes software in terms of the ease of its transfer from one environment (software/hardware) to another.

Reasons for errors in programs

Why does it happen that programs do not work correctly? It's very simple - they are created and used by people. If the user makes a mistake, this can lead to a problem in the operation of the program - it is used incorrectly, which means it may not behave as expected.

Error is a human action that produces an incorrect result.

However, programs are designed and created by people, who can (and do) make mistakes. This means that there are shortcomings in the software itself. They are called defects or bugs (both designations are equivalent). The important thing to remember here is that software is more than just code.

Defect, Bug [5] - a defect in a component or system that can lead to the failure of certain functionality. A defect discovered during program execution can cause failure of a single component or the entire system.

When executing program code, defects that were inherent during its writing may appear: the program may not do what it should or, on the contrary, do what it should not do - a failure occurs.

Failure [5] - discrepancy between the actual result of the operation of a component or system and the expected result.

A program failure may be an indicator that it contains a defect.

Thus, the bug exists when three conditions are met simultaneously:

- the expected result is known;
- the actual result is known;
- the actual result differs from the expected result.

It is important to understand that not all bugs cause crashes - some of them may not manifest themselves at all and go unnoticed (or appear only under very specific circumstances).

Failures can be caused not only by defects, but also by environmental conditions: for example, radiation, electromagnetic fields or pollution can also affect the operation of both software and hardware.

There are several sources of defects and, accordingly, failures:

errors in the specification, design or implementation of the software system;

- system usage errors;
- unfavorable environmental conditions;

- intentional causing harm;
- potential consequences of previous errors, conditions or intentional actions.

Defects can occur at different levels, and whether and when they are corrected will directly determine the quality of the system.

Conventionally, we can identify five reasons for the appearance of defects in program code.

Lack or lack of communication within the team. Often, business requirements simply do not reach the development team. The customer has an idea of what he wants the finished product to look like, but if his idea is not properly explained to developers and testers, the result may not be as expected. Requirements must be accessible and understandable to all participants in the software development process.

- Software complexity. Modern software consists of many components that are combined into complex software systems. Multi-threaded applications, client-server and distributed architecture, multi-level databases - programs are becoming more and more difficult to write and support, and the more difficult the work of programmers becomes. And the more difficult the work, the more mistakes the person performing it can make.
- Changes in requirements. Even minor changes to requirements late in development require a large amount of work to implement changes to the system. The design and architecture of the application changes, which, in turn, requires changes to the source code and the principles of interaction of program modules. Such ongoing changes often become the source of subtle defects. However, frequently changing requirements in modern business are the rule rather than the exception, so continuous testing and risk control in such conditions is the direct responsibility of the quality assurance department.
- Poorly documented code. Poorly written and poorly documented code is difficult to maintain and change. Many companies have special rules for how programmers write and document code. Although in practice it often happens that developers are forced to write programs quickly in the first place, and this affects the quality of the product.
- Software development tools [6]. Visualization tools, script generators and other development tools are also often poorly functioning and poorly documented programs that can become a source of defects in the finished product.

2 Objectives and Methodology

2.1 Objectives

The objectives of the work are to compare old and new software testing methods.

This section discusses the objectives of this work, namely methods that provide a comparison of old and new software testing methods.

Old or classic software testing methods include the following:

1. Functional testing [3] is the testing of software manually (without the use of automated tools). The specialist performs testing and evaluates the quality of the product from the end user's point of view. The tester looks for defects and shortcomings at various levels. In this case, the process is regulated using a pre-developed plan.
2. Black Box Testing [3]: A testing technique without in-depth knowledge of the internal workings of the software is called black box testing. The specialist does not take into account the system architecture and does not have access to the source code. Typically, when performing a black box test, a specialist works with the system's user interface, entering data and analyzing the result. At the same time, the tester does not know how and where this data is processed.
3. White Box Testing [3]: White box testing is a detailed examination of the internal logic and structure of the code. Testing using this method is also called glass testing or open testing. Knowing how the code works, the expert studies it from the inside and finds out which device/block of code is behaving incorrectly.
4. Gray Box Testing [3]: This method is a cross between the previous two. When testing a gray box, a specialist should have an understanding of the internal structure of the software - but not too deep. He puts himself in the place of the end user, but checks the functionality of the program based on an understanding of its internal structure.

Old or classic testing models include the following:

Waterfall Model: This is a structured software development model that is quite applicable in the testing process. The sequential nature of the model requires the tester to adhere to a clear algorithm. The process is divided into several stages: requirements definition - design - coding - implementation - verification - installation - maintenance.

In this process, no stage can overlap or overtake another. It is a simple model that makes software testing easy and efficient.

Iterative Development: Here, each component is tested multiple times. This model works in three sequential cycles: form - test - evaluate. Immediately after the iteration of each part, a new, improved model is developed and submitted for testing. Thanks to prompt feedback from testing results, necessary changes in design/functionality/utility can be added to the new model.

New software testing methods are listed in the following list:

1. Automated testing [7]. When using this method, the QA engineer writes scripts and uses specialized applications in his work. Automation saves time on checking the quality of IT systems and is suitable for repeating standard operations - from simulating user work to creating test reports. This approach is used in situations where manual control cannot be done. Automated tests are developed individually, taking into account the relevant characteristics of the software product.

New or modern testing models include the following:

Agile Methodology [8] is a more complex software development model with a step-by-step approach to testing. Sometimes product requirements change—for example, if you are developing a startup and testing several business hypotheses. In this case, the previous two models will not be suitable. But in the flexible model, each component is tested immediately, which reduces the risk factor during the software operation.

Currently the following levels of software testing are also used:

- A unit test tests components at the module level. The tester checks each source code and compares it with the expected result.
- Integration test is designed to check the communication between modules. This level helps identify errors that prevent integrated components from communicating. To carry it out, several approaches are used, such as “top-down”, “bottom-up” and “sandwich”.
- System testing is also known as end-to-end testing because it tests the entire software. This technique provides a complete report on the system's performance and compliance with specified business requirements. In addition, the entire system is tested for performance at the individual component level.

The following methods are used to compare old and new types of testing:

1)Assessing the speed of software testing using certain methods.

2)Evaluating the effectiveness of various types of software testing.

3) Comparing the degree of software functionality coverage by tests in old and new methods.

4) Analyzing how quickly and effectively various testing methods detect defects and issues in the software.

5) Comparing how accurately old and new testing methods identify different types of defects.

Evaluating the quality of testing is an essential component of software development, especially when comparing old and new testing techniques. Because it includes multiple essential components that enhance the overall efficacy and dependability of the testing procedure, this review process is essential.

It mostly entails evaluating the system's capacity to provide quality testing using both traditional and novel techniques. This is a real problem that practitioners in the field are dealing with, not just a theoretical exercise. The evaluation takes into account a number of factors, such as the coverage of checks and the testing's development level. It basically aims to ascertain how exhaustively and fully the testing procedure investigates the software that is being evaluated.

Furthermore, the evaluation of quality explores the frequency of error detection. This is an important feature since it shows how quickly and effectively bugs and issues are found in the software. Gaining insight into the error detection frequency can help determine how reliable the testing strategy is.

Moreover, the quality representation of distinguishing flaws is part of quality assessment. Not only must errors be correctly identified, but they must also be thoroughly documented and described. In order to ensure that faults are addressed appropriately and that development teams communicate effectively, this phase is crucial.

2.2 Methodology

The methodology used to compare different testing methods is to compare the effectiveness of one method or another.

The following metrics are used to determine the quality of software tests:

- The number of defects found and corrected is estimated. The number of defects found during the testing process and the number of defects successfully corrected can serve as an indicator of the quality of testing.
- The percentage of tests completed is determined. This is the ratio of the number of tests performed to the total number of tests scheduled. A high percentage of tests completed indicate good testing quality.
- Estimate the time required to correct defects. Average time between detection of a defect and its correction. The shorter this time, the faster and more efficient the testing process.
- The number of missed defects is determined. The number of defects that were discovered after the product was released. Fewer missed defects indicate high quality testing.

For the practical implementation of this methodology, selected tests are run for the selected program code, and then the metric indicators are evaluated.

3 Literature Review

3.1 Svyatoslav Kulikov “Software testing. Basic course".

For beginners, we primarily recommend the book by Svyatoslav Kulikov “Software Testing. Basic course".

This book covers the following topics:

1. Software testing and development processes

Software development model - a structure that systematizes various types of project activities, their interaction and sequence in the software development process. The choice of a particular model depends on the scale and complexity of the project, the subject area, available resources and many other factors. The following software development models exist:

- waterfall model – classic model;
- V-model - classic model;
- iterative model, incremental model – modern model;
- spiral model – special case of the iterative model, incremental model;
- agile model - a set of different approaches to software development and is based on the so-called. In the Agile Manifesto [1], people and collaboration are more important than processes and tools.

Testing life cycle

Stage 1 (general planning and analysis of requirements) is objectively necessary at least in order to have an answer to questions such as: what do we have to test; how much work there will be; what are the difficulties; do we have everything we need, etc. As a rule, it is impossible to get answers to these questions without analyzing the requirements, because it is the requirements that are the primary source of answers.

Stage 2 (clarification of acceptance criteria) allows you to formulate or clarify metrics and signs of the possibility or need to begin testing (entry criteria), suspension (suspension criteria) and resumption criteria (resumption criteria) of testing, completion or termination of testing (exit criteria).

Stage 3 (refinement of the testing strategy) is another appeal to planning, but at the local level: those parts of the test strategy (test strategy) that are relevant for the current iteration are considered and refined.

Stage 4 (test case development) is devoted to the development, revision, clarification, refinement, processing and other actions with test cases, sets of test cases, test scripts and other artifacts that will be used in the actual testing.

Stage 5 (execution of test cases) and stage 6 (fixation of found defects) are closely related to each other and are actually performed in parallel: defects are recorded immediately after they are detected during the execution of test cases.

However, often after all test cases have been completed and all defect reports have been written, an explicit refinement stage is carried out, in which all defect reports are reviewed again in order to form a common understanding of the problem and clarify characteristics of the defect such as importance and urgency. Stage 7 (analysis of test results) and stage 8 (reporting) are also closely related and are carried out almost in parallel. The conclusions formulated at the results analysis stage directly depend on the test plan, acceptance criteria and refined strategy obtained at stages 1, 2 and 3.

The findings are formalized at stage 8 and serve as the basis for stages 1, 2 and 3 of the next testing iteration. Thus the cycle is completed.

2. **Testing documentation and requirements**

   Requirement. A condition or capability needed by a user to solve a problem or achieve an objective that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.

3. **Types and directions of testing**
4. **Checklists, test cases, test case sets**
5. **Errors, defects, failures, failures, etc.**
6. **Defect reports**
7. **Examples of using various testing techniques**
8. **Test automation**

3.2 Beizer B. Black Box Testing

Dr. Beiser's book, Black Box Testing, has long been recognized as a classic work in the field of behavioral testing of a variety of systems. It deeply examines the main issues of software testing, allowing you to find a maximum of errors with a minimum of time. The basic testing techniques covering all spectrums of aspects of software systems development are described in great detail. The methodical nature and breadth of presentation make this book an indispensable assistant when checking the correct functioning of software solutions.

The book is intended for software testers and programmers seeking to improve the quality of their work.

3.3 Sam Kaner, Jack Faulk, Yong Kek Nguyen. Software testing. Fundamental concepts of business application management

The book by eminent experts in the field of software development is devoted to one of the most important and non-trivial aspects within the process of creating complex software systems. The book is distinguished, first of all, by its connection to real world conditions using examples of well-known development companies located in Silicon Valley. A wide range of issues are discussed in detail: from organizing the testing process to the actual texting of the project, code, documentation, etc.

4 Practical Part

This part compares two testing methods - the “old” or classic method, waterfall testing, and the modern method, agile testing. The labor costs and efficiency of these testing methods are compared.

4.1 Waterfall testing model

The waterfall model [9] is a model of the software development process in which the development process looks like a flow, successively passing through the phases of requirements analysis, design, implementation, testing, integration and support.

Following the cascade model, the developer moves from one stage to another strictly sequentially. First, the “requirements definition” stage is completely completed, resulting in a list of software requirements. Once the requirements are fully defined, the transition to design occurs, during which documents are created that detail for programmers how and how to implement the specified requirements. After the design is completely completed, programmers implement the resulting project. The next stage of the process involves the integration of individual components developed by different teams of programmers. After implementation and integration are completed, the product is tested and debugged. At this stage, all shortcomings that appeared at previous stages of development are eliminated. After this, the software product is implemented and its support is provided - introducing new functionality and eliminating errors.

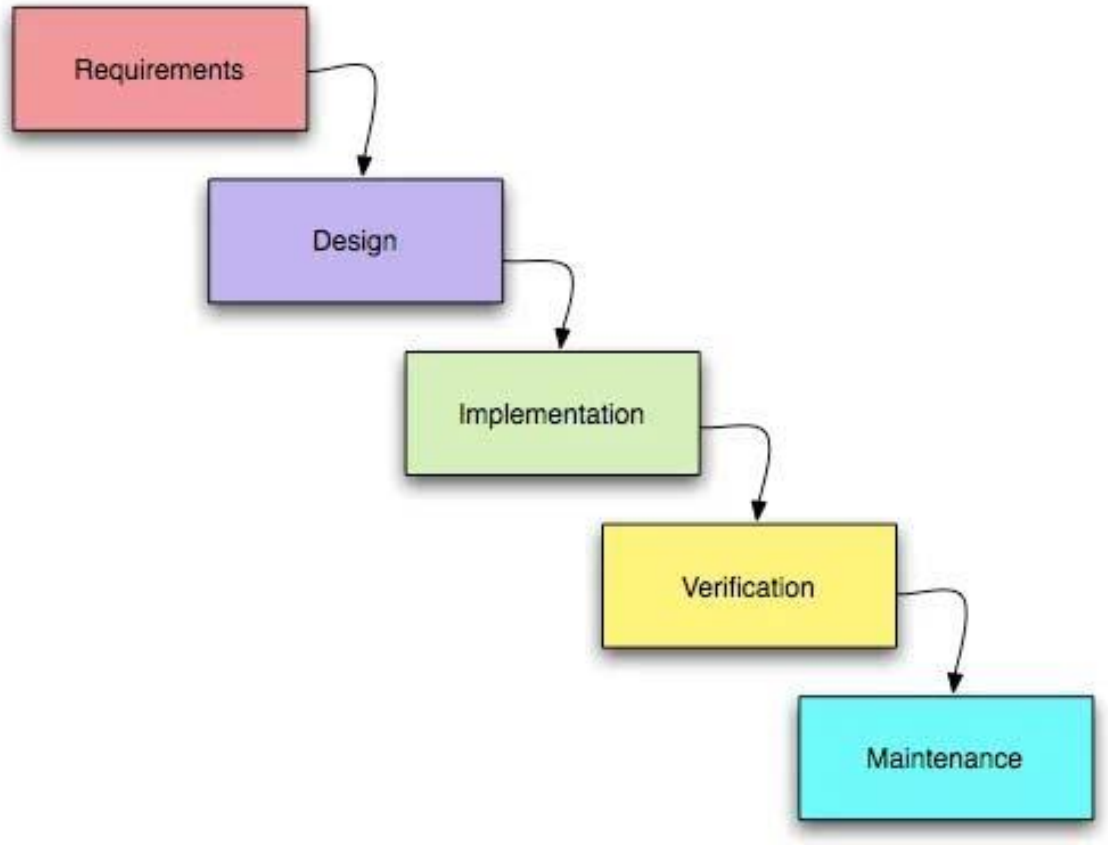

Fig. 4.1. Waterfall model

This model implies strictly sequential and one-time execution of each phase of the project. The transition from one phase to another is possible only after the successful completion of the previous stage. Each stage implies detailed planning and complete correctness of the result of the stage.

Such strict sequence restrictions allow us to build a development process that is as transparent and convenient as possible for the Customer.

The other side of this method is the need to support and constantly update product development documentation. Any change must be agreed upon with the Customer. And an insufficient level of elaboration of requirements entails an increase in the budget and project timescales, which are quite difficult to estimate.

Today, the waterfall model of software development is practically not used due to the low flexibility of the model. However, it continues to be used due to the high transparency of development. Thanks to the high level of formalization, managing such a project is much easier. It is generally accepted that the waterfall development

model reduces risks and brings clarity to the development process when several dozen people are working on a project.

The waterfall model is suitable for developing complex and large projects and systems with strictly defined functionality. Use when developing large government orders or scientific developments. It is highly undesirable to use this methodology for developing business applications.

In the process of comparative testing, the following advantages and disadvantages of this technique were discovered.

4.1.1 Advantages

- High transparency of development and project phases
- Clear sequence
- Stability of requirements
- Strict control of project management
- Facilitates the work of drawing up a project plan and assembling a project team
- Well defines the quality control procedure

4.1.2 Disadvantages

- A Waterfall project must have up-to-date documentation at all times. Mandatory updating of project documentation. Redundant documentation
- Very non-agile methodology
- May create a false impression of work on the project (for example, the phrase “45% completed” does not carry any useful information, but is just a tool for the project manager)
- The Customer does not have the opportunity to familiarize himself with the system in advance and even with the “Pilot” of the system
- The User does not have the opportunity to get used to the product gradually
- All requirements must be known at the beginning of the project life cycle
- There is a need for strict management and regular monitoring, otherwise the project will quickly go behind schedule
- There is no possibility to take into account rework, the entire project is done at one time

4.2 Agile testing model

Agile development methodology (Agile software development, agile methods) [10] is a series of approaches to software development focused on the use of interactive development, dynamic formation of requirements and ensuring their implementation as a result of constant interaction within self-organizing working groups consisting of specialists in various fields. There are several techniques that belong to the class of flexible development methodologies, in particular extreme programming.

Most agile methodologies aim to minimize risk by condensing development into a series of short cycles called iterations, which typically last two to three weeks. Each iteration itself looks like a miniature software project and includes all the tasks necessary to deliver a mini-increment in functionality: planning, requirements analysis, design, programming, testing and documentation. Although a single iteration is generally not sufficient to release a new version of a product, the assumption is that an agile software project is ready for release at the end of each iteration. At the end of each iteration, the team re-evaluates development priorities.

Agile methods are such flexible methodologies as Lean Development, Scrum, etc. They were developed in the early 2000s as an alternative to ineffective traditional IT methods. Almost all agile teams are concentrated in one office (bullpen). The office includes the product owner - the customer, who determines the requirements for the product. The customer can be a business analyst, a project manager, or a client. In addition, the office may include interface designers, testers, and technical writers. That is, agile methods are aimed primarily at direct communication.

The main metric of agile methods is work product. By prioritizing direct communication, agile methods reduce the amount of written documentation compared to other methods.

Key ideas:

- people and interaction are more important than processes and tools;
- a working product is more important than comprehensive documentation;
- cooperation with the customer is more important than agreeing on the terms of the contract;
- willingness to change is more important than sticking to the original plan.

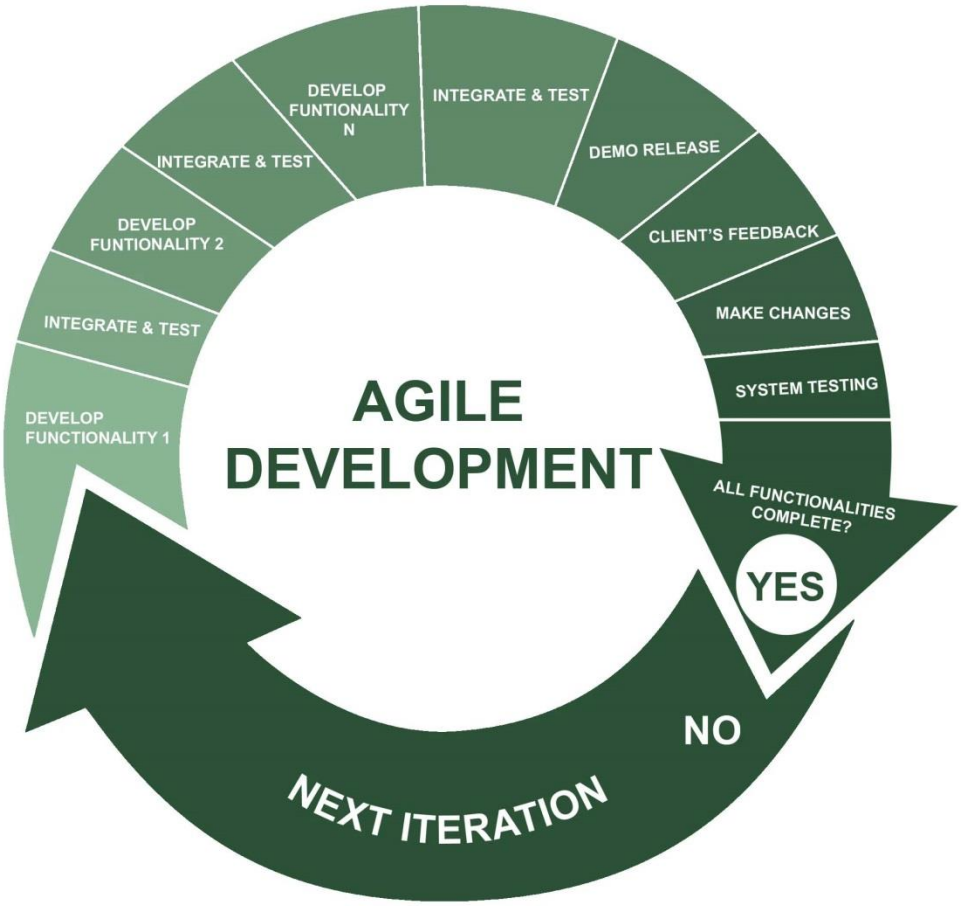

Fig. 4.2. Agile testing model

Agile principles:

1. customer satisfaction through early and uninterrupted delivery of valuable software;
2. welcoming changes in requirements even at the end of development (this can increase the competitiveness of the resulting product);
3. frequent delivery of working software (every month or week or even more often);
4. close, daily communication between the customer and the developers throughout the entire project;
5. the project is carried out by motivated individuals who are provided with the necessary working conditions, support and trust;
6. the recommended method of transmitting information is personal conversation (face to face);

7. working software is the best measure of progress;
   sponsors, developers and users must be able to maintain a constant pace indefinitely;
8. constant attention to improving technical excellence and user-friendly design;
9. simplicity - the art of not doing unnecessary work;
10. the best technical requirements, design and architecture are obtained from a self-organized team;
11. constant adaptation to changing circumstances.

4.2.1 Advantages

During the testing process, the following advantages of agile testing were revealed.

- Product quality

Involving the customer in the process of each iteration makes it possible to adjust the process, which invariably improves quality.

- High development speed

The iteration lasts no more than 3 weeks, by the end of this period there will definitely be a result.

- Minimizing risks
- A large project allows the customer to pay for several iterations and, during the work, understand that he will receive exactly what he wants on time and at an affordable price. Waterfall models (using specifications and technical specifications) do not provide such opportunities.
- The customer always has the opportunity to monitor the progress of development, adjust the functionality of the project, test or launch it, and can even stop it at any time.

In the process of comparative testing, the following disadvantages of this technique were discovered.

4.2.2 Disadvantages

- Difficult to adapt

When changing the management style to the agile methodology, the team needs time to adapt. This concerns new responsibilities and approach to project development. Often, managers encounter problems when they first try Agile.

However, constant practice and training can help everyone on the team get comfortable. Agile supports autonomy in achieving goals. Therefore, to begin with, you can divide the team into groups. This will facilitate consultation on emerging questions or problems.

- Changing Goals

Agile pays attention to several goals at the same time, which does not work as a plus, it is rather a minus. Sometimes some of them may be forgotten. This can lead to timing and cost uncertainty. To prevent unwanted costs and missed deadlines, regular meetings can be held to discuss goals. Another way is to implement a cost policy to ensure that the project budget is adhered to.

- Insufficient documentation

In Agile, documentation is not a priority, compared to planning and progress. This can lead to problems with documentation, such as keeping records and plans. To fix this, you should get used to regular documentation, at least once a month. If documentation is falling behind, emphasizing its importance in the schedule can help bring it back to the attention it deserves.

- Lack of documented improvements

Agile focuses on quickly introducing improvements, leaving documentation in the background. This may slow down progress tracking. While rapid implementation of improvements works to the benefit of the process, it is important not to forget the role of documentation. Regularly reviewing documents and tracking progress will provide the team with transparency about goals.

- Shifting focus away from goals

Agile encourages shifting focus depending on the urgency of tasks, which can make it difficult for a team to work towards a common goal. Closer to the end of the project, the team could focus on one task. To coordinate efforts, it is useful to distribute tasks among team members in the final phase of the project.

- Reduced predictability

Agile relies on continuous improvement and feedback, which can make it difficult to predict profits and costs. Agile aims to ship a quality product quickly, which

sometimes causes problems with forecasting. However, active use of feedback and attention to documentation can help predict possible defects.

4.3 Automated testing

The most modern method of testing is automated testing. This type of testing can be used in the following areas:

- execution of test cases that are impossible for a person;
- solving routine tasks;
- acceleration of testing;
- release of human resources for intellectual work;
- increase in test coverage;
- improvement of the code due to the increase in test coverage and the use of special automation techniques.

4.3.1 Advantages

- The speed of execution of test cases can be times and orders of magnitude higher than human capabilities. If you imagine that a person will have to manually check several files with a size of several tens of megabytes each, the estimate of the time of manual execution becomes frightening: months or even years. At the same time, 36 checks implemented as part of smoke testing with command scripts {284} are performed in less than five seconds and require only one action from the tester - to run the script.
- There is no influence of the human factor in the process of performing test cases (fatigue, carelessness, etc.). Let's continue the example from the previous point: what is the probability that a person will make a mistake comparing (symbolically!) even two ordinary texts of 100 pages each? And if there are 10 such texts? 20? And do the checks need to be repeated over and over again? It is safe to say that a person is guaranteed to make a mistake. Automation will not make a mistake.
- Automation tools are capable of performing test cases that are, in principle, impossible for a human due to their complexity, speed, or other factors.

- And again, our example of comparing large texts is relevant: we cannot afford to spend years repeatedly performing an extremely complex routine operation in which we are also guaranteed to make mistakes. The second excellent example of test cases that are overwhelming for a person is the study of performance {91}, in the framework of which it is necessary to perform certain actions at high speed, as well as to fix the values of a wide set of parameters. Will a person, for example, be able to measure and record the amount of RAM occupied by an application a hundred times per second? No. Automation will be able to.
- Automation tools are capable of collecting, storing, analyzing, aggregating and presenting colossal volumes of data in a form convenient for human perception.
- Automation tools are able to perform low-level actions with the application, operating system, data transmission channels, etc. In one of the previous paragraphs, we mentioned such a task as "one hundred times per second, measure and record the amount of RAM occupied by the application." A similar task of collecting information about the resources used by the application is a classic example. However, automation tools can not only collect such information, but also affect the application execution environment or the application itself, emulating typical events (for example, lack of RAM or processor time) and fixing the reaction of the application.

4.3.2 Disadvantages

- The need for highly qualified personnel is due to the fact that automation is a "project within a project" (with its own requirements, plans, code, etc.). Even if you forget for a moment about the "project within a project", the technical qualification of employees engaged in automation, as a rule, should be significantly higher than that of their colleagues engaged in manual testing.
- The development and maintenance of both the automated test cases themselves and the entire necessary infrastructure takes a lot of time. The situation is aggravated by the fact that in some cases (in the case of serious changes in the project or in the case of errors in the strategy), all the relevant work has to be performed anew from scratch: in the case of a significant change in

requirements, a change in the technological domain, reworking of interfaces (both user and software) many test cases become hopelessly outdated and need to be created anew.

- Automation requires more careful planning and risk management, because otherwise, serious damage may be caused to the project (sees the previous paragraph about remaking all the works from scratch).
- Commercial means of automation are significantly expensive, and the available free analogues do not always allow you to effectively solve the tasks. And here we are again forced to return to the issue of errors in planning: if initially the set of technologies and automation tools was chosen incorrectly, it will be necessary not only to redo all the work, but also to buy new automation tools.
- There are too many automation tools, which complicates the problem of choosing one or another tool, makes it difficult to plan and define a testing strategy, may entail additional time and financial costs, as well as the need to train personnel or hire relevant specialists.

5 Results and Discussion

In this work, modern and outdated methods of software testing were considered. In particular, testing using the waterfall model of software development, the flexible model of software development, and testing automation were considered. The following results were obtained.

5.1 Waterfall method

This development model [9] implies testing at a separate stage, which is associated with high costs, especially in the case of detection of serious errors. As established in the work, in the case of developing a new software project, the costs of error elimination will be many times higher than the costs of error elimination in the case of using other development models.

- Since testing starts at the stage of completion of development, if a bug is discovered, its correction will cost more than at the initial stage. After all, testers will find an error only when the developer has already finished writing the code, and the copywriters - the documentation.
- The customer gets acquainted with the finished product after the development is completed. Accordingly, he can evaluate the product only when it is almost completely ready. If he does not like the result, the cost of the project budget will increase significantly due to the need for correction.
- The more technical documentation, the longer it takes to complete the work. Such documentation requires more changes and is agreed upon.

5.2 Agile testing

In the work, flexible testing methods were studied, as a result of which the following results were obtained.

Cooperation and communication

- Everyone in the team works together and exchange ideas.
- Testers and developers work closely together to achieve common goals.
- Communication takes place throughout the development process to avoid misunderstandings and respond to changes in a timely manner.

Flexibility and adaptability

- Agile allows you to quickly react to changes in requirements.
- Plans and priorities may change during development, and the team must be ready to adapt.
- This implies a flexible approach to planning and execution of tests.

Continuous feedback

- Interaction and feedback between team members are considered an integral part of agile testing.
- Fast and effective transmission of information about errors and problems allows the team to react quickly and improve the product.
- Feedback also contributes to the improvement of the testing process and improvement of methods.

Iterativeness and incrementality

- Work on the project is divided into small iterations or sprints.
- Each iteration has specific goals and priorities for testing.
- The team strives to gradually improve the product and software quality by consistently adding new functions and eliminating errors.

Testing automation

- Automatic tests allow you to quickly perform repeated checks and provide a wider coverage of testing.
- This helps speed up the process and provides more reliable testing of the product's functionality.

Customer involvement

- Agile assumes the active participation of the customer in the development and testing process.
- The customer provides feedback, clarifies requirements and helps determine testing priorities.
- Close cooperation with the customer contributes to the achievement of higher product quality.

6 Conclusion

The following results were obtained in the work:

1. The goals and methodology used in the research process are formulated.
2. Classical ("old") software testing methods are examined and studied in detail.
3. Modern methods of software testing are also studied.

In the process of comparative analysis of old and new testing methods, the following was determined:

- Classic (old) testing methods are used in the development of large and medium-sized software projects, and as a rule, new versions of previously developed projects. In this case, there is a minimal risk of detecting critical (serious) errors, the elimination of which requires large labor and financial costs.
- Flexible (new) testing methods allow testing at each stage. In this case, error detection is ensured at each stage of program development, which, in turn, allows minimizing the costs of error elimination and speeds up the process of releasing new versions of programs.
- Also, in case of application of flexible testing methods, the possibility of automation of testing is provided. Thanks to the automation of the testing process, the work of specialists in the field of testing is facilitated by the use of the following solutions:
- The use of command files to perform sequences of operations — from copying several files from different directories to deploying a test environment.
- Data generation and processing using the capabilities of office applications, databases, small programs in high-level programming languages. There is no sadder picture than a tester who manually numbers two hundred lines in a table.
- Preparation and design of technical sections for reports. You can spend hours scrupulously reading logs of the work of some automation tool, or you can write a script once, which will prepare a document with neat tables and graphs in an instant, and all that remains is to run this script and attach the results of its work to the report.
- Management of the tester's workplace: creation and verification of backup copies, installation of updates, cleaning of disks from outdated data, etc. etc. The computer can (and should!) do all this by itself, without human involvement.
- Mail sorting and processing. Even sorting incoming correspondence into subfolders is guaranteed to take you a few minutes a day. If you assume that setting up special

rules in your email client will save you half an hour a week, the savings will amount to approximately 24 hours per hour.

- Virtualization as a way of getting rid of the need to install and configure the necessary set of programs every time. If you have several pre-prepared virtual machines, it will take seconds to start them. And if it is necessary to eliminate failures, deploying a virtual machine from a backup copy replaces the entire process of installing and configuring the operating system and all necessary software from scratch.

7 References

1. Agile-manifesto. [http://agilemanifesto.org/iso/ru/manifesto.html].
2. Agile System Development Life Cycle. [http://www.ambysoft.com/essays/agileLifecycle.html].
3. Boris Beizer. Black-Box Testing: Techniques for Functional Testing of Software and Systems 1st Edition, Wiley & Sons, 320 pages, 1995.
4. David Gelperin, Bill Hetzel. Growth of Software Testing. [https://www.researchgate.net/publication/234808293_The_growth_of_software_testing].
5. International Software Testing Qualifications Board Glossary. [http://www.istqb.org/downloads/glossary.html].
6. History of Software Testing. [http://www.testingreferences.com/testinghistory.php].
7. James Whittaker. The Plague of Aimlessness. [https://testing.googleblog.com/2009/06/7-plagues-of-software-testing.html].
8. Sam Kaner, Jack Faulk, Yong Kek Nguyen. Testing Computer Software, 496 pages, 1999.
9. Svyatoslav Kulikov. Software testing. Basic course. 301 pages. EPAM Systems.
10. What are the Software Development Models? [http://istqbexamcertification.com/what-are-the-software-development-models/].

8 List of pictures, tables, graphs and abbreviations

8.1 List of pictures

The following figures are used in this work:

9 Appendix

9.1 List of basic definitions used in testing

Alpha testing. Testing that is performed within the developer's organization with the possible partial involvement of end users. It can be a form of internal acceptance testing.

Automated testing. A set of techniques, approaches, and tools that allow you to exclude a person from performing some tasks in the testing process.

Beta testing. Testing that is performed outside the developer's organization with the active involvement of end users/customers.

Black box testing. A testing method in which the tester either does not have access to the internal structure and code of the application, or is not sufficiently familiar with them to understand them, or he does not consciously refer to these data during the testing process.

Border condition, boundary condition. A value located on the boundary of the equivalent class.

Code review, code inspection. A family of techniques for improving code quality due to the fact that several people participate in the process of creating or improving code. In contrast to the techniques of static code analysis (by control flow and data flow), code audit also improves such characteristics as comprehensibility, maintainability, conformance to design conventions, etc. Code auditing is performed mainly by the programmers themselves.

Defect, anomaly. Deviation of the actual result from the observer's expectation, formed on the basis of requirements, specifications, other documentation or experience and common sense.

Dynamic testing. Testing with running the code for execution.

Integration testing. Testing, which is aimed at checking the interaction between several parts of the application (each of which, in turn, is checked separately at the stage of module testing).

Gray box testing. A combination of white box and black box methods, consisting in the fact that the tester has access to part of the code and architecture, but not to part.

Smoke test. Testing that is aimed at checking the most important, most important, most key functionality, the inoperability of which makes the very idea of using the application (or other object subject to smoke testing) pointless.

Software Development Model. A structure that systematizes various types of project activity, their interaction and sequence in the software development process. The choice of one or another model depends on the scale and complexity of the project, subject area, available resources and many other factors.

Root cause analysis. The process of research and classification of the root causes of the occurrence of events that negatively affect safety, health, the environment, quality, reliability and production process.

White box testing. A testing method in which the tester has access to the internal structure and code of the application, and is also familiar enough to understand what he saw.