# EFFECTIVE TESTING IN AGILE/SCRUM: METHODOLOGY AND STRATEGY



ANATOLII TYMOSHCHUK

**Introduction** **4**
Purpose of the book 4
Relevance of the testing topic in Agile/Scrum 4
A brief overview of the Agile/Scrum methodology 5
**Chapter 1. The role of testing in an Agile team** **6**
The main responsibilities of a tester in an Agile team 6
The importance of testing in the Agile process 8
Benefits and challenges of the tester role in Agile 9
**Chapter 2. Types of tests and their importance in Agile** **11**
Types of tests conducted by a tester 11
How to ensure quality at all stages of development 15
**Chapter 3. Collaboration with other roles in an Agile team** **18**
Collaboration with developers, analysts, and product owners 18
The impact of the tester's role on product quality 20
**Chapter 4. Testing strategy in Agile/Scrum conditions** **22**
Why do you need a testing strategy? 23
The difference between a test strategy and a test plan and types of test strategies 24
Tools and approaches for strategy implementation 26
**Chapter 5. Test automation in Agile** **30**
How to choose the right tools and frameworks 30
Prepare your infrastructure for automation 33
Integration of automation into CI/CD 36
**Chapter 6. Integrate testing into sprints** **52**
Sprint-level test planning 53
Automation priorities 55
Implementation of testing in Definition of Done (DoD) 57
**Chapter 7. Maintaining product quality in a fast-release environment** **61**
Technical debt management 62
Quality metrics: how to measure success 66
**Chapter 8. The future of testing in Agile** **68**
Advancing automation with AI/ML 69
Hybrid testing approaches 71
Continuous training and development of specialists 75
**Chapter 9. Conclusion** **77**
Summary of key ideas 78
The importance of a quality culture in an Agile team 79
**Application** **83**

List of Test Automation Tools 83
Useful links and resources for learning 83
Templates for Agile Test Planning 84

# Introduction

## Purpose of the book

The goal of this book is to help quality professionals, developers, and agile team leaders understand how to effectively integrate testing processes into agile and scrum projects. We'll look at key aspects of testing, including the roles of manual and automated testing, how to apply them, and how to ensure high product quality in a rapid release environment.

This book is designed to inspire readers to improve their testing processes and provide practical tools to do so. Whether you are a novice tester or an experienced specialist, here you will find useful recommendations for your work.

In recent years, Agile methodologies, including Scrum, have become the de facto standard for software development management. Their popularity is due to their ability to provide quick adaptation to changes, close interaction with the client, and regular deliveries of the work product. However, this flexibility creates new challenges for teams, especially in the area of quality assurance.

## Relevance of the testing topic in Agile/Scrum

A key problem with Agile is that the development cycle has become shorter and the pace of releases has significantly accelerated. In traditional approaches, testing was often seen as the final step after development was complete. In Agile, this approach loses its relevance as iterative processes require constant testing. This makes testing an integral part of development, not its final stage.

**Flexibility vs. Quality**

Agile emphasizes the importance of delivering product increments quickly, but this should not come at the expense of quality. In the face of frequent changes in requirements, there is a risk that some functionalities will not be sufficiently tested, which leads to defects at the operational stage. That is why the role of testing in Agile is becoming critical: testers do not just look for errors, but actively prevent their occurrence, starting from the first stages of development.

**The Rise of Automation**

One of the key trends in the context of Agile is the rise of test automation. Automated tests allow you to quickly validate each iteration, minimizing the time spent on regression testing. This is especially important in an environment where releases can occur weekly, daily, or even several times a day. However, automation does not replace manual testing, but complements it by providing verification of critical aspects of the product.

**The Role of the Team in Quality Assurance**

Agile has changed the very understanding of testing. It is no longer the task of one person or a group. In an agile team, responsibility for quality is shared among all team members. Developers write tests for their code, analysts help refine use cases, and testers build testing strategies and methodologies, including automation and risk analysis. This is a team approach to ensuring the quality of the product.

**Main challenges**

Working in Agile brings with it many benefits, such as flexibility, high speed, and close interaction with the client. However, all this also creates new challenges for testers:

- **Frequent changes in requirements.** This increases the likelihood that test cases will have to be reviewed and adapted.
- **Tight deadlines.** Sometimes only a few hours are allotted for the execution of tests.
- **Teamwork.** The tester no longer works in isolation, but becomes part of a cross-functional team, where his contribution directly affects the success of the project.

**Why is it important now?**

Test automation has become the standard without which it is impossible to imagine a successful Agile team. It allows you to cope with the volume of work, maintain product stability, and free up time for more complex tasks. But automation is not a panacea. It requires competent implementation, support, and integration into the CI/CD process. In addition, there are aspects of the product that cannot be tested automatically, and this is where manual testing comes to the fore.

Testing is more than just looking for defects. This is active participation in creating a quality product, preventing errors, analyzing risks, and helping the team achieve a common goal. That is why the topic of testing in Agile is so relevant.

This book is designed to help teams build effective testing processes in an Agile/Scrum environment, cope with challenges, and use them as opportunities for growth. You will learn how to build testing into sprints, improve interaction between roles, implement automation, and, most importantly, how to ensure high product quality even in the face of rapid releases.

## A brief overview of the Agile/Scrum methodology

Agile is a project management philosophy based on flexibility, adaptability, and constant feedback. The basic principles of Agile include:

- Incremental development: Creating a product in small chunks that add value to users.

- Collaboration: Close interaction between team members, stakeholders and customers.
- Quick onboarding: Your team's ability to respond to changing requirements.

Scrum is one of the popular Agile implementations that includes:

- Team roles: Scrum Master, Product Owner, Development Team.
- Sprints: Short iterations (typically 1-4 weeks) in which work is completed.
- Artifacts: Product Backlog, Sprint Backlog, and Increment.
- Ceremonies: Sprint planning, daily stand-ups, showcasing results, and a retrospective.

In the context of Agile/Scrum, testing should be built into all of these processes. For example, Definition of Done (DoD) includes running all tests, both automated and manual, until the task is completed. In addition, testers are actively involved in planning and retrospective to improve the quality of processes and products.

Thus, testing in Agile is not only about finding bugs, but also about constantly improving the quality of the product and processes. In the following chapters, we will analyze in detail how testers can effectively interact with the team, what types of tests are used in Agile, and how to build a successful testing strategy.

# Chapter 1. The role of testing in an Agile team

## The main responsibilities of a tester in an Agile team

Testing plays a critical role in an agile team. It ensures product quality at every stage of development and helps to identify and eliminate defects in a timely manner. The tester performs many tasks aimed at improving the quality and stability of the software. In this section, we will look at the main responsibilities of a tester in an Agile team.

The main task of a tester in Agile is to create and maintain a high level of product quality. This requires close collaboration with other team members, including developers, analysts, and product owners. The tester must be ready to adapt to changes and respond quickly to new requirements and changes in the project.

A tester in an Agile team performs the following key responsibilities:

**Task planning and evaluation:**

During the sprint planning process, the tester works closely with the rest of the team to discuss user stories in detail. It analyzes requirements, identifies potential risks, and proposes solutions to minimize them. The tester also estimates the complexity and effort of testing each task, which helps the team plan the amount of work per sprint more accurately. This phase is critical to ensure that all aspects of testing are considered from the outset.

**Development of test cases:**

Test cases are the most important tool for a tester to systematically check the performance of the product. When developing test cases, the tester describes in detail the steps required to conduct each test, as well as the expected results. It takes into account different types of testing, such as functional, regression, integration, and end-to-end testing. It is also important that test cases are clear and accessible to all team members, which promotes transparency and collaboration.

**Test Execution:**

The tester conducts both manual and automated tests, checking the correct operation of all product functions. It uses a variety of testing techniques such as positive and negative testing to ensure that the product performs reliably in various scenarios. When defects are found, the tester logs them in the bug tracking system, detailing the problem and the steps to reproduce it. After the developers fix the defects, the tester retests to make sure the problem is fixed.

**Test Automation:**

Test automation can significantly improve the efficiency and reliability of the testing process. The tester chooses suitable tools and frameworks for automation, such as

Selenium, JUnit, TestNG, and others. It creates automated tests for repetitive and mission-critical scenarios, allowing you to quickly identify regression errors. Automated tests are integrated into the continuous integration and delivery (CI/CD) process, ensuring that the product is tested regularly and in a timely manner whenever the code changes.

**Collaboration with the team:**

Effective collaboration is a key element of the success of an Agile team. The tester participates in daily stand-ups, where the current status of tasks, identified problems, and plans for the day are discussed. He works closely with developers, analysts, and product owners to ensure they understand requirements and expectations. In retrospectives, the tester shares his observations and suggestions for improving the testing process and the overall work of the team.

**Documentation maintenance:**

Documentation plays an important role in the testing process, ensuring transparency and consistency. The tester maintains up-to-date testing documentation, including test cases, test plans, and defect reports. It also collects and analyzes testing metrics such as the percentage of tests covered, the number of defects found, and the time it takes to run the tests. This data helps the team assess the quality of the product and identify areas for improvement.

**Quality control at all stages of development:**

The tester is actively involved in the code review process, which allows you to identify potential defects at an early stage of development. It conducts testing at the level of unit tests, integration tests, and regression tests, which ensures high standards of product quality. This multi-layered approach to testing helps minimize defects and ensures software stability and reliability throughout the development lifecycle.

## The importance of testing in the Agile process

Testing in the Agile process is central because it ensures high product quality and end-user satisfaction. In the face of constant change and a fast pace of development, testing becomes an integral part of the entire project life cycle.

**Early detection of defects**

One of the key benefits of Agile testing is the ability to identify and fix defects early. Frequent and short sprints allow testers to test early in the development process, identifying potential issues before they become critical. This allows you to reduce the cost of correcting defects and minimize technical debt.

**Continuous feedback**

Testing provides continuous feedback to the development team. Regular testing and integration of test results into the development process help developers quickly respond to changes and improve the quality of the product. Testers provide important information about the current state of the product, which allows the team to make timely adjustments and adapt to new requirements.

**Quality assurance at every stage**

Agile testing focuses on quality assurance at all stages of development. This includes requirements review, code review, manual and automated testing, and regression testing. Testers work closely with developers, analysts, and product owners to ensure that the product meets customer expectations and requirements.

**Support for frequent releases**

The agile approach requires frequent and rapid releases, making testing a critical element of the process. Testers help ensure the stability and quality of the product with each release by conducting regression testing and automating repetitive tasks. This minimizes the risks associated with frequent updates and changes to the product.

**Process and Method Improvement**

Testing in Agile contributes to the continuous improvement of development processes and methods. Retrospectives and regular discussions of test results help the team identify weaknesses and find ways to eliminate them. Testers play a key role in this process by suggesting improvements and optimizing test strategies and techniques.

**Risk mitigation**

Testing helps reduce risks related to product quality and end-user satisfaction. Regular testing and monitoring can identify and fix potential problems early, reducing the likelihood of critical defects in the late stages of development.

**Increased product credibility**

Quality testing increases the trust of customers and end users in the product. Testers ensure that the product works reliably and meets the stated requirements, which contributes to user satisfaction and strengthens the reputation of the team and the company.

**Supporting a culture of quality**

In an agile team, testing supports a culture of quality, where each team member is aware of the importance of ensuring a high level of quality at all stages of development.

Testers are actively involved in building this culture by providing valuable information and suggesting methods to improve processes.

Regular testing supports frequent and stable releases, reduces risk, and increases product confidence with customers and end users. Thanks to the close cooperation of testers with developers, analysts and product owners, as well as the continuous improvement of processes and methods of work, the team can quickly adapt to changes and meet market requirements. Testing in Agile contributes to building a culture of quality, where each team member is aware of his responsibility to create a reliable and high-quality product.

Thus, the importance of testing in the Agile process can hardly be overestimated. It not only ensures the quality and stability of the product, but also helps the team succeed in the face of constantly changing requirements and high expectations of end users.

## Benefits and challenges of the tester role in Agile

The role of a tester in an agile team has many advantages, but it also comes with its own challenges. In this section, we will consider what benefits and difficulties a tester can encounter in the process of working in an agile environment.

1. **Team integration:**

- In Agile, the tester works side by side with developers, analysts, and the product owner. This contributes to a faster exchange of information and the elimination of misunderstandings.
- Early integration allows you to identify and fix bugs at the requirements discussion stage.

2. **Enhanced product quality:**

- Constant testing at each stage of development reduces the likelihood of critical errors in the release.
- The shift-left approach allows you to detect defects at the design and planning level.

3. **Flexibility and adaptability:**

- Testers are actively involved in changing requirements, ensuring that they are correctly understood and testable.
- Rapid adaptation to changes minimizes delays and increases team efficiency.

4. **Growth of professional skills:**

- Working in an Agile team provides a tester with the opportunity to constantly learn and develop their skills. Testers face various tasks and problems that require learning new tools and approaches. This contributes to professional growth and professional development, making testers more valuable specialists in the labor market.

5. **Transparency of the process:**

- Thanks to daily meetings and close cooperation with the team, testers have a clear understanding of the goals and current state of the project.

6. **Influence on architecture and design:**

- Testers in Agile have the ability to influence architectural and design decisions. By participating in the design and code review phases, they can suggest improvements and identify potential problems early on. This contributes to the creation of more stable and easily testable code, which ultimately improves the quality of the product.

However, it's important to understand that these benefits come with challenges, such as the need to constantly learn new technologies, respond quickly to changes, and balance manual and automated testing.

**Challenges of the tester role in Agile**

1. **Short iterations and tight deadlines:**

- Sprints require fast task completion, which can limit the time for full-fledged testing.
- Testers need to find a balance between the depth of verification and meeting deadlines.

2. **Frequent changes in requirements:**

- Constant updates can lead to the need to redo test cases or create new tests.
- This increases the load on testers, especially when resources are limited.

3. **Need for a wide range of skills:**

- Testers must be versatile: they must be able to work with automation, understand CI/CD processes, and know the business logic of the product.
- The introduction of new technologies and tools requires constant learning.

4. **Shared responsibility for quality:**

- In Agile, testing is the responsibility of the entire team, not just the tester. This can lead to confusion in areas of responsibility.
- Testers should be able to delegate test-related tasks to other team members.

5. **Ensuring process continuity:**

- Frequent releases and integrations require a high degree of automation and test stability.
- Maintaining the relevance of test coverage in the face of dynamic changes requires a lot of effort.

The benefits and challenges of the Agile tester role show how diverse and important this role is. Successful testers in an agile team are able to effectively balance between benefits and challenges, maintaining a high level of product quality and contributing to the overall success of the project.

# Chapter 2. Types of tests and their importance in Agile

## Types of tests conducted by a tester

Testing is an integral part of the software development process, especially in an Agile environment. To ensure the high quality of the product, testers use various types of testing that help to identify and correct defects at different stages of development. In this section, we will consider the main types of tests conducted by testers: unit, integration, system, regression, and end-to-end testing. Each of these types of testing has its own characteristics and plays an important role in the overall quality assurance process.

**Unit testing**

Unit testing is aimed at checking the operation of individual modules or components of the system. Each module is tested in isolation from the others, which allows you to identify errors at the code level. Unit tests are usually developed and run by developers, but testers can also participate in the process of creating and reviewing tests.

**The main characteristics of unit testing:**

- Check the operation of individual functions and methods.
- Run tests quickly.
- High degree of isolation from other components.
- Assistance in identifying defects at the early stages of development.

Unit tests allow you to make sure that the code works correctly at the level of individual modules, which reduces the likelihood of errors at higher levels of the system.

**Integration testing**

Integration testing is aimed at checking the interaction between different modules or components of the system. During integration testing, modules are combined and tested as a single unit to ensure that they work correctly as part of the system.

**Main characteristics of integration testing:**

- Checking the interaction between modules.
- Identify defects related to component integration.
- Testing interfaces and interconnections between modules.
- Ensure that data is transferred correctly between components.

Integration testing helps to identify errors that occur during the interaction of modules, and guarantees the correct operation of the system as a whole.

**System Testing**

System testing is aimed at checking the operation of the entire system as a whole. It is carried out at the system level and includes a review of functional and non-functional requirements. System testing ensures that the product meets the stated requirements and expected behavior.

**The main characteristics of system testing are:**

- Checking the entire system as a whole.
- Functional and non-functional requirements testing.
- Run tests on real or near-real data.
- Evaluate system performance, security, and usability.

System testing is an important step in checking the quality of a product, as it allows you to make sure that it is ready for use by users.

**Regression testing**

Regression testing is aimed at verifying that changes in the code have not led to new defects in previously functional parts of the system. This testing is done after changes or fixes have been made to the code to ensure that the product remains stable and functional.

**The main characteristics of regression testing are:**

- Validation of previously tested features after code changes.
- Ensuring the stability and functionality of the system.
- Rerun tests to identify new defects.
- Automate regression tests to improve efficiency.

Regression testing minimizes the risks associated with making changes and ensures that the product remains stable and operational.

**End-to-end testing**

End-to-end testing is aimed at verifying the operation of the entire system from start to finish, including all its components and interactions. This testing simulates the real use of the product by users and allows you to make sure that all functions and scenarios work correctly.

**The main characteristics of end-to-end testing:**

- Verification of the operation of the entire system in real conditions.
- Testing of all features and use cases.
- Identify defects related to the interaction of different components.
- Measure usability and user satisfaction.

End-to-end testing allows you to make sure that the product works correctly and meets the needs of end users, ensuring high quality and stability of the system.

**Exploratory Testing**

Exploratory testing is an approach in which testers study and test a product without pre-written test cases. This allows you to detect unexpected defects and problems that may have been missed in traditional testing.

**The main characteristics of exploratory testing:**

- Flexibility and adaptability.
- Testers examine the product in real-world conditions.
- Identification of hidden and unexpected defects.
- Using the experience and intuition of the tester.

Exploratory testing allows testers to gain a deeper understanding of the product and identify issues that may be difficult to detect with other testing methods.

**Performance Testing**

Performance testing is aimed at assessing the speed, scalability, and stability of the system under various loads. This testing helps determine how well the system handles high load and identify bottlenecks that can lead to poor performance.

**Key features of performance testing:**

- Assessment of the speed and scalability of the system.
- Checking the operation of the system under high load.
- Identify bottlenecks and performance issues.
- Analysis of the stability and reliability of the system.

Performance testing is a key aspect of product quality assurance, especially for systems that need to handle a large number of users and data.

**Acceptance Testing**

Acceptance testing is carried out to verify that the product meets the requirements and expectations of the customer and end users. This testing is typically done in the final stages of development and before the product is released.

**The main characteristics of acceptance testing are:**

- Verification of product compliance with customer requirements.
- Evaluation of usability and functionality.
- Enable real-world use cases.
- Participation of the customer and end users in testing.

Acceptance testing is the final stage of product quality control, which helps to ensure that it is ready for release and meets customer requirements.

**Relationship between tests in Agile**

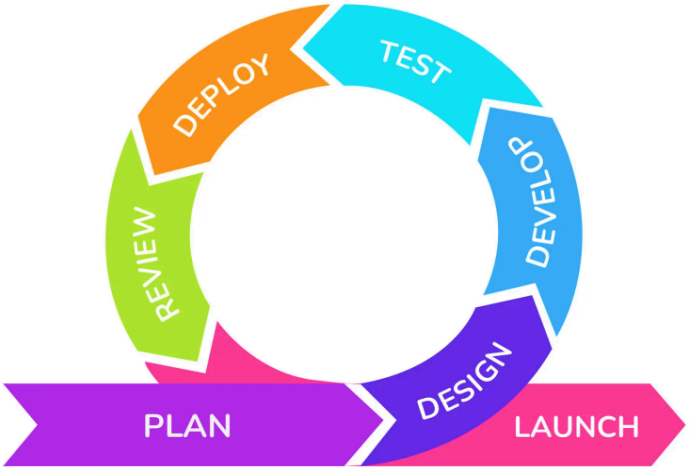
The use of all these types of tests in Agile allows you to effectively manage the quality of the product. Unit tests ensure the stability of the code, integration tests ensure the correct interaction of components, system testing checks compliance with requirements, regression tests prevent new errors, and end-to-end testing confirms the performance of the system in real conditions. This multi-layered approach allows you to identify problems at each stage of development and minimize risks in a rapid release environment.

Thus, the role of a tester in the Agile process includes conducting a variety of types of testing, each of which performs a unique function aimed at maintaining a high level of product quality. From thorough Unit tests to extensive End-to-End testing, each methodology plays a role in ensuring the stability and compliance of the product with the requirements of users and the business.

The key idea is to use a combined approach that combines automated and manual testing, helping to minimize risks and identify potential problems in a timely manner. Using adaptive testing methods allows agile teams to be ready for frequent changes and rapid releases without losing control over quality.

## How to ensure quality at all stages of development

The sprint cycle in Agile/Scrum is a key iteration in which all the main stages of work on the functionality, including testing, are performed. In Agile processes, quality control becomes a collective responsibility, covering all stages: from planning to release. Below we will take a closer look at the stages of the sprint cycle and the role of testing at each of them.




**Picture. Visualization of the Agile testing process**

1. **Planning**

In the Sprint Planning Phase, the team works with the Product Owner to select tasks from the product backlog to run during the sprint. These tasks form the sprint backlog.

Role of the tester:

- Participate in discussions about user stories and tasks.
- Checking the Acceptance Criteria for completeness and testability.
- Suggesting clarifications and additions to make the requirements more understandable.
- Estimating the amount of testing for each task and taking this time into account in planning.

2. **Design phase**

The design phase includes the development of the architecture and design of solutions for the tasks included in the sprint. The team discusses the technical details and decides how the features will be implemented.

Role of the tester:

- Participate in the discussion of the architecture and design of solutions proposed by the developers.
- Determine what types of testing will be needed to validate new features and changes.

- Develop test strategies and plans to ensure that all aspects of the product are fully covered by tests.
- Offer tools and approaches that can be useful for effective testing.

**3. Development phase**

During development, the team works on the implementation of tasks, including writing code, test cases, and automated tests.

Role of the tester:

- Development of test cases for tasks, including functional, non-functional and regression tests.
- Write automated tests for key scenarios.
- Collaborate with developers to conduct joint testing, such as unit tests.

**4. Testing phase**

Testing is not a separate stage, but an integrated part of the work throughout the sprint. However, in the middle and towards the end of the sprint, it takes up a large share of activity.

Types of testing:

- Functional testing: Validate new features for compliance.
- Regression testing: Confirm that changes have not affected existing functionality.
- Integration testing: Validates the interaction between modules.
- User Interface (UI) Testing: Verify that the interface works for the end user.

Role of the tester:

- Conduct manual and automated tests to verify the health and quality of new features.
- Perform functional, regression, integration, and end-to-end testing.
- Log identified defects and bugs, detailing the problem and the steps to reproduce it.
- Retest corrected defects to ensure that they are completely resolved.

**5. Deploy**

During the deployment phase, the team prepares the product for release. This includes deploying new versions to test or production servers, setting up the infrastructure, and performing final validations.

**Role of the tester:**

- Conduct final tests before deploying the product to test or production servers.
- Ensure that your infrastructure is ready to deploy new product versions.
- Check the stability and health of the system after deployment.
- Ensure that you are ready to respond quickly to potential issues during the deployment process.

**6. Sprint review**

After the sprint is over, the team holds a sprint review meeting to demonstrate the results achieved to the product owner and stakeholders. The review discusses the tasks completed, the goals achieved and the problems encountered.

**Role of the tester:**

- Participate in demonstrating the achieved results (product increments) to the product owner and stakeholders.
- Provide a report on the tests performed, the defects identified and the results achieved.
- Discuss successes and problems that arose during the sprint.
- Collect feedback to further improve the testing process and product quality.

**7. Launch**

The last stage of the sprint cycle is the launch of the product or its updates. The team officially releases a new version of the product to users, ensuring its availability and stability.

**Role of the tester:**

- Ensure that the product is ready for launch by conducting final quality checks.
- Monitor the performance and stability of the system after startup.
- Respond quickly to emerging problems and defects, eliminating them in the shortest possible time.
- Monitor quality metrics and report on the status of the product after launch.

Testing in Agile is not limited to finding defects. Quality assurance at all stages of development is a complex task that requires the involvement of the entire team. The tester plays a key role in achieving this goal by applying a variety of approaches and techniques to identify and eliminate defects at an early stage. This approach allows us to create products that not only meet user expectations, but also exceed them.

# Chapter 3. Collaboration with other roles in an Agile team

## Collaboration with developers, analysts, and product owners

One of the key principles of Agile is the close interaction of all team members, including developers, analysts, product owners, and testers. This collaboration allows not only to improve the quality of the product, but also to increase the efficiency of the development process. In this section, we'll look at how testers can successfully collaborate with different roles on a team to achieve common goals.

**Interaction with developers**

**Close work on tasks**

Testers and developers work closely together at every stage of development. A joint discussion of tasks, requirements and acceptance criteria helps to better understand the goals and expectations. Testers can offer their ideas for improving the quality of the code and identifying potential problems at an early stage.

**Code and test case reviews**

Conducting regular code reviews and test cases allows you to identify defects and improve the quality of the product. Testers can participate in the code review, offering their comments and recommendations. Collaborative review helps to share knowledge and experience, which contributes to more effective team work.

**Test automation**

Testers and developers can collaborate on test automation. Developing and maintaining automated tests requires close collaboration to identify and fix defects faster. Collaboration on automation also contributes to improving CI/CD processes and increasing overall team efficiency.

**Interaction with analysts**

Business analysts form requirements and help clarify development goals. Testers work closely with analysts to understand user expectations and correctly create acceptance criteria.

**Definition and clarification of requirements**

Analysts play a key role in defining and refining product requirements. Testers should work closely with analysts to understand the business requirements and goals of the project. Joint discussion of requirements helps to identify possible problems and shortcomings at an early stage.

**Creating Acceptance Criteria**

Acceptance criteria are an important element of testing in Agile. Testers and analysts can work together to develop acceptance criteria for each user story. This allows for a clear understanding of what needs to be tested and how, and ensures that the product meets customer requirements.

**Testing user scenarios**

Analysts can provide testers with valuable insights into real-world product use cases. This data helps to develop test cases that are as close as possible to real operating conditions. Joint discussion of user scenarios allows you to identify possible problems and improve the quality of the product.

**Interact with Product Owners**

**Prioritizing tasks and requirements**

Product owners play an important role in prioritizing tasks and requirements. Testers work with the product owner to ensure that the product being developed meets business goals. This helps to properly allocate efforts and resources to the most important and critical tasks.

**Getting feedback**

Regular feedback from product owners allows testers to better understand customer expectations and requirements. This helps to identify and fix problems in a timely manner, as well as make improvements to the testing process and product quality.

**Demonstration of results**

Testers can participate in demonstrating the results of the sprint to product owners. This allows you to show the results achieved, get feedback and discuss next steps. Working collaboratively with product owners helps ensure high product quality and customer satisfaction.

Effective collaboration with developers, analysts, and product owners is the foundation of success in an agile team. Close interaction, regular communication and knowledge sharing contribute to the creation of a high-quality product that meets the requirements and expectations of the customer. Testers play an important role in this process, ensuring quality control at all stages of development and improving the overall efficiency of the team.

## The impact of the tester's role on product quality

The role of a tester in an Agile team has a huge impact on the quality of the product being created. The tester not only identifies defects, but also helps to prevent them, maintains high standards of development and creates conditions for continuous improvement. This influence is evident at every stage of development, making the tester a key participant in the process.

**1. Early detection of defects**

Testers play an important role in identifying defects and problems early, which helps reduce the cost of fixing them. By participating in sprint planning and code reviews, testers can detect potential bugs and flaws at the earliest stages of development. This allows the team to quickly respond to identified problems and prevent their negative impact on the product.

*Example:* During sprint planning, a tester noticed that one of the user stories contained unclear requirements for the input data. He raised the issue in the meeting, and the team, along with the product owner, clarified the requirements. This helped to avoid errors during development and testing, which reduced the time to fix defects and improved the quality of the product.

**2. Reduction of technical debt**

Testers help reduce technical debt, which can accumulate due to unidentified defects and problems. By conducting regular regression testing, they ensure the stability of the product and prevent new errors when code changes are made. This allows us to maintain a high level of product quality and stability throughout the project life cycle.

*Example:* A tester performed regression testing after making changes to the code and found that one of the functions that worked before stopped functioning correctly. He registered the defect in the tracking system, and the developers promptly fixed the problem. This helped prevent the accumulation of technical debt and kept the product stable.

**3. Increase confidence in the quality of the product**

Regular testing allows the team to be confident in the quality of the product at each stage of development. Testers ensure that all functional and non-functional requirements are fully covered by tests, which allows you to identify and eliminate most defects before the release of the product. This increases the confidence of the team and the customer that the product will work reliably and meet user expectations.

*Example:* Testers regularly run automated tests and provide reports on test results to the team. With these reports, developers and product owners can be sure that the product is working correctly and is compliant. This increases confidence in the quality of the product and helps plan for future releases.

**4. Improving the user experience**

Testers play a key role in ensuring a high level of usability of the product. By conducting user scenario testing and exploratory testing, they help identify issues that may affect user satisfaction. This allows you to make the necessary changes and improvements to make the product as convenient and useful as possible for end users.

*Example:* During exploratory testing, a tester discovered that one of the app's features was inconvenient to use and caused confusion among users. He proposed several options for improving the interface, which were discussed and accepted by the team. As a result, the user experience has become more enjoyable and intuitive.

**5. Influence on architecture and design**

Testers participate in the discussion of the architecture and design of solutions, offering their recommendations and comments. Their experience and knowledge help create a more sustainable and easily testable architecture, which ultimately improves the quality of the product. Testers can identify potential problems at the design level and suggest solutions to prevent them.

*Example:* In the design phase, a tester suggested changing the data structure for one of the features to simplify testing and improve performance. The team reviewed and accepted the proposal, resulting in a more resilient and easily testable architecture, improving product quality and stability.

**6. Ensuring transparency and visibility of work**

Testers provide transparency and visibility into the team's work by providing reports on test results, quality metrics, and other important information. This helps the team and stakeholders stay on top of the current state of the product and make informed decisions. Transparency and visibility of work promote more effective communication and collaboration within the team.

*Example:* A tester regularly updates testing documentation, including test plans, test cases, and defect reports. These documents are shared with the entire team and stakeholders, ensuring transparency and visibility. The team can see the current state of the product, identified issues, and test results, which promotes more effective communication.

**7. Maintaining a Culture of Quality**

Testers play a key role in maintaining a culture of quality in an agile team. They promote the importance of quality at all stages of development, help implement best practices and testing methods, and participate in retrospectives and discussions to improve processes. This fosters an environment where quality is a priority for all team members.

*Example:* A tester initiates retrospectives after each sprint, where the team discusses the successes and challenges that arose during development and testing. He also suggested implementing regression testing automation, which helped improve processes and improve product quality. These initiatives contribute to building a culture of quality, where each team member is aware of their responsibility to create a reliable and high-quality product.

**8. Continuous improvement**

In Agile processes, the tester becomes a catalyst for continuous improvement. Through defect analysis, user feedback, and an iterative approach, the team gets the opportunity to improve the product at every stage.

*Example:* After each sprint, the tester helps the team analyze what worked well and what could be improved. Lessons learned from past mistakes are embedded in future processes, resulting in a more stable and high-quality product.

The role of a tester in an Agile team has a huge impact on the quality of the product. This role encompasses not only the search and elimination of defects, but also the implementation of practices that contribute to the creation of a product that meets high standards. A tester who actively works at all stages of development becomes a guarantor of quality, increasing the value of the product for users and business.

# Chapter 4. Testing strategy in Agile/Scrum conditions

In the context of the Agile/Scrum methodology, testing plays a key role in ensuring the quality and stability of the product. Frequent and short development cycles require a specific testing approach that is flexible, efficient, and adaptable to rapid change. The testing strategy in Agile/Scrum is aimed at integrating testing into all stages of development, close cooperation within the team, and ensuring continuous quality control.

The Agile/Scrum approach requires testers to be not just performers, but active team members who take part in planning, development, and retrospectives. Testing in Agile is not only about verifying the operation of the application, but also ensuring that the product meets the expectations of customers and users, as well as responds to changes in requirements.

The purpose of this chapter is to take a closer look at the key aspects of a testing strategy in an Agile/Scrum environment, including the importance of testing, the methods and tools needed to work effectively, and best practices that will help ensure high product quality at all stages of development. In this chapter, we will look at how to build an effective testing strategy that will be integrated into the development process, maintain the quality of the product, and contribute to the successful implementation of the project.

## Why do you need a testing strategy?

You can do without a testing strategy only in an ideal world where there are endless resources — a team, time, budget. Under these conditions, the product can be tested indefinitely, using different methods and techniques in a random order, until it reaches a state of readiness for release.

However, in real life, projects have tight deadlines, resources are limited, and requirements change frequently. In such conditions, working without a clear plan inevitably leads to chaos and a decrease in quality. This is where a testing strategy comes in. It allows you to efficiently allocate resources, minimize risks and achieve your goals in the face of restrictions.

To organize the testing process in conditions of limited resources, it is important to have a clear strategy. It helps to effectively use available opportunities and minimize risks. But first, you need to realize what resources the team has: the qualifications of specialists, time and budget.

A testing strategy provides a unified understanding of the role of testing for all project participants. It helps the team and stakeholders understand what testing can bring to the product, what results to expect, and what benefits it will bring. In addition, the strategy

creation process helps identify potential issues that may arise during the planning or development phase.

**Who is responsible for creating a strategy?**

Most often, a tester (QA) is responsible for developing a testing strategy together with a project manager. These two professionals combine their knowledge to create a plan that covers both technical aspects and organizational nuances.

In the process of creating a strategy, it is important to have a detailed discussion: gather a team, take paper, markers, laptops, and start planning. This will allow you to take into account all possible risks and provide optimal approaches to testing. Such preliminary preparation creates the basis for the successful implementation of the project.

## The difference between a test strategy and a test plan and types of test strategies

In the field of software quality assurance, the test plan and test strategy are the most important documents, each of which performs its own specific role. Understanding their differences and importance helps QA teams effectively structure the testing process and achieve high results. Let's take a closer look.

**A testing strategy** is a high-level document that formulates a general approach to testing in a company or on a specific project. It outlines the main methods, tools, and standards that will be applied to ensure the quality of the product.

The strategy sets the goals of testing, defines the types of testing (for example, functional, non-functional, regression, etc.), and describes the roles and responsibilities of team members. This document is developed early in the project and serves as the basis for all subsequent testing plans and activities.

**A test plan** is a more detailed document that describes how testing will be conducted for a specific project or phase. It includes a description of specific tasks: what features will be tested, who will run the tests, what resources will be required, and the timeline and criteria for successful completion of testing.

A test plan includes a work schedule, role assignments, defect tracking tools, and reporting methods. It adapts to the specifics of the project, taking into account its specific requirements and limitations.

Below is a table that shows the differences between a test strategy and a test plan:

| Criterion | Test strategy | Test plan |
|---|---|---|

| | | |
|---|---|---|
| **Scope** | A general document describing testing approaches and principles | Specifies testing for a specific project or version |
| **Level of detail** | High level, includes general principles and methods | Detailed, includes practical aspects of testing |
| **Purpose** | Determination of the general orientation of testing at all levels | Planning to run tests for a specific project |
| **Flexibility** | Stable throughout the project | Subject to change depending on changes in the project |

**Types of test strategies**

Below are the types of strategies adapted to different tasks:

1. **Analytical strategy**

   This approach uses risk- and demand-based testing methods that require different approaches. After analyzing factors such as risks and requirements, the QA team clarifies the testing conditions. In the case of requirements-based testing, these requirements are examined and tests are then developed and executed to verify that they are met. An analytical strategy tracks the results of meeting requirements: those that have been tested, those that have not, and those that have not been fully tested.

2. **Model-based strategy**

   The testing team evaluates real and assumed conditions and builds a model, taking into account the inputs, outputs, actions, and possible reactions of the product. For example, when testing the performance of a mobile application, you can create a model that simulates mobile network traffic, the number of active users, and predicted changes, and then develop a strategy based on this model.

3. **Methodical strategy**

   This approach involves strict adherence to formalized quality standards (for example, ISO25000), the use of checklists and other official methods. In some types of testing, for example, in the field of security or mobile applications, generally accepted and standardized checklists of checks are used. For example, for maintenance testing, a checklist that describes functions, their properties, and other details is often enough.

4. **Standards- or process-oriented strategy**

   An example is the testing of medical IT systems, which are required to meet certain government standards. In such cases, the company must follow the recommendations set by committees or groups of specialists, which applies not only to the testing

conditions, but also to the compilation of test cases and the work of the QA team. If the requirements are less stringent, for example, when developing according to an informal standard (for example, Agile), a test strategy can be created for each user story with a description of testing criteria, test cases, and scenarios.

5. **Reactive strategy**

Tests are created and run only after the product is released. The focus is on defects found in an already working system. For example, exploratory testing of the product is carried out, test cases are developed for already implemented functions, and then tests and approaches are updated based on the results obtained. This approach is often used in Agile development.

6. **Advisory strategy**

As with Agile, this approach is based on feedback from users and stakeholders. For example, when testing cross-browser compatibility of a web application, product owners can provide a list of browsers, versions, operating systems, and other requirements.

7. **Anti-regression testing strategy**

This type of strategy focuses on reducing the risk of regression of both the functional and non-functional aspects of the product. For example, if a web application needs to be tested for regression, the QA team can automate both positive and negative use cases, running tests after each application update.

All these strategies can be used depending on the features of the product or combined with each other. Choosing the right strategy will be discussed below.

**Choosing the right strategy**

The following factors influence the choice of testing strategy:

- The testing strategy depends on the product features.
- Requirements, such as security-focused applications, require more stringent approaches.
- The product development model also determines the choice of strategy.
- "Large projects" require a long-term strategy.
- The type and size of the company can also influence the choice of an appropriate strategy.

## Tools and approaches for strategy implementation

In Agile/Scrum approaches, testing is not an isolated activity, but an important part of the entire development process, integrated into each stage. In order for a testing strategy to be effective, it is necessary to use the appropriate tools and approaches that are appropriate for dynamic and flexible development conditions. In this section, we will look at the key tools and approaches that will help you successfully implement a test strategy in Agile/Scrum, improve team interaction, improve product quality, and speed up the development cycle.

**1. Test automation tools**

Test automation in Agile is a necessity. Frequent releases, rapid changes, and the need to quickly test new features require the use of tools that can run tests automatically and at high speed. Automated tests speed up the development cycle and increase confidence that new changes do not introduce defects in the parts of the application that are already in operation.

- **Selenium**: One of the most popular tools for automating web application testing. Selenium allows you to perform functional and regression tests in different browsers, which is especially useful when testing web applications where browser compatibility is critical.
- **JUnit and TestNG:** These frameworks are the foundation for testing Java applications. They allow you to create tests, organize them into groups, and integrate them with CI/CD systems to automatically validate changes with every commit.
- **Cucumber**: A behavior-based testing (BDD) tool that allows a team of developers and testers to write tests in a language that everyone involved in the process understands. This lowers the barrier between technical specialists and business stakeholders.
- **Postman**: For REST API testing, Postman provides a powerful set of tools for creating, running, and documenting tests. This tool allows you to test requests, validate server responses, and easily integrate API testing into your CI/CD process.
- **Appium**: This is a mobile app automation tool that supports both Android and iOS. With Appium, you can write tests that will work on a variety of devices and operating systems, which is especially important for mobile apps.

**2. Defect tracking and management tools**

Effective defect management is a key aspect of the testing process that helps the team respond quickly to identified problems, effectively capture them, and avoid them in the future.

- **JIRA**: One of the most popular tools for project management and defect tracking in Agile teams. JIRA helps you organize tasks, track defects, and integrates with other tools like Jenkins and GitLab to automatically update defect status.
- **Bugzilla**: A simple and easy-to-use defect tracking system that allows you to record detailed information about a bug, its status, and when it was resolved.
- **Trello**: Used to visualize tasks and defects. This is especially useful for Scrum or Kanban teams, where tasks and defects can be tracked using cards that move through the columns depending on the stage of execution.

**3. Continuous Integration and Delivery (CI/CD) Tools**

Continuous integration and delivery are processes that involve frequently merging changes into the primary repository and automatically verifying their quality through tests. CI/CD allows you to improve code quality and speed up the release cycle, which is a perfect fit for Agile principles.

- **Jenkins**: One of the most popular tools for automating the CI/CD process. Jenkins allows you to automate the build, testing, and deployment of an application by integrating with various version control systems, testing tools, and defect tracking systems.
- **GitLab CI:** Built-in CI/CD capabilities that provide a comprehensive solution for automating the entire process, from testing to deployment, with integration with GitLab repositories and other external services.
- **CircleCI:** A cloud-based service for CI/CD that allows you to automate testing and deployment, integrates with GitHub and GitLab, supports various programming languages and frameworks.
- **Travis CI:** An easy-to-use CI/CD service often used in open source projects. It allows you to easily set up pipelines for automated testing and building projects.

**4. Approaches to testing in Agile/Scrum**

In Agile/Scrum, testing is not limited to just running tests at the end of the sprint; It becomes part of every stage of development. Testing begins with planning and continues throughout the sprint. Let's look at a few key approaches that make the testing process in Agile effective.

- **Testing as part of every sprint**: Unlike the traditional approach, where testing is done after development is complete, in Agile, testing is an ongoing process that starts at the earliest stages of a project and continues throughout the sprint.
- **Behavioral Testing (BDD):** This is an approach in which tests are described in a language that is understandable to both developers and business stakeholders. In

BDD, test scenarios are written from the user's point of view, which improves understanding of functionality and allows all project participants to work in a single information field.

- **Testing using user stories**: In Scrum, user stories are the main artifact that describes the requirements for functionality. Testing is based on these stories, which allows you to quickly test and validate the requirements in practice.

**5. Use metrics to assess quality**

Evaluating the effectiveness of a test strategy in Agile requires the regular use of metrics. These metrics allow you to evaluate not only the quality of the product, but also the effectiveness of the team and testing as a whole.

- **Code Coverage**: This is a metric that shows what percentage of the code is covered by tests. A high coverage ratio helps ensure that changes don't break the functioning of the application.
- **Mean Time to Defect (MTTD):** This metric helps track how quickly the team detects defects. The faster a defect is detected, the faster it can be resolved.
- **Test Velocity**: Measures how many tests are running in a sprint. This metric helps you understand how well your team is testing and which areas may require additional effort.

**6. Use a collaborative environment**

Agile relies on close communication and collaboration between team members. Using collaboration tools helps the entire team share information quickly, discuss problems, and find solutions.

- **Confluence**: A documentation and collaboration platform. Confluence allows you to store all the necessary test cases, documentation, and reports, and facilitates knowledge sharing within your team.
- **Slack**: Instant messaging and group channels help your team quickly resolve testing issues and respond quickly to project changes.

**7. Performance Monitoring and Load Testing Tools**

To check how the system behaves under load, it is important to use performance monitoring tools. This helps identify bottlenecks and improve the overall performance of the application.

- **JMeter**: One of the most popular tools for load testing. JMeter allows you to create scripts to test the performance of web applications and other systems.

- **Gatling**: Another performance testing tool that supports Scala and provides easy-to-use reports for analyzing load test results.
- **New Relic**: A real-time application monitoring platform that allows you to monitor performance and identify problems during the application process.

The use of appropriate tools and methods allows you to implement a test strategy in Agile/Scrum and ensures effective testing at each stage of development. Test automation, integration with CI/CD processes, and the use of approaches focused on business values and user behavior are key aspects of successfully implementing a test strategy.

# Chapter 5. Test automation in Agile

Test automation plays a key role in the Agile methodology, helping to ensure high product quality, speed up development processes, and reduce risks. With the frequent and short sprints that characterize Agile, automation allows the team to quickly and efficiently check product health, identify defects, and ensure system stability.

Implementing automated tests into the development process allows you to reduce the time for routine and repetitive tasks, freeing up testers for more complex and creative types of testing, such as exploratory testing. Test automation also improves the continuous integration and delivery (CI/CD) process by ensuring that tests are run regularly whenever code changes and minimizing the likelihood of defects in production.

The purpose of this chapter is to take a closer look at how test automation can be integrated into the agile development process, what tools and approaches to use for this, as well as best practices that will help the team get the most out of automated tests. We will discuss how to choose the right tools and frameworks, how to prepare the infrastructure for automation, and how to integrate automation into CI/CD processes. This chapter will help the Agile team build a sustainable and effective test automation strategy that promotes reliable and high-quality products.

## How to choose the right tools and frameworks

Choosing the right tools and frameworks for Agile test automation is a key step to successfully implementing a testing strategy. Choosing the right tools can speed up the testing process, increase its efficiency, and ensure high product quality. In this section, we'll look at the main criteria for choosing tools and frameworks, as well as some popular solutions that can be useful for your team.

**1. Defining requirements and goals**

The first step in choosing tools and frameworks is to define your team's requirements and goals. It's important to understand what tasks need to be solved with automation, what types of testing need to be performed, and what quality metrics will be used.

**Basic steps:**

- Identify key test automation goals, such as reducing regression testing time, improving test coverage, and improving system stability.

- Define tool requirements, such as support for specific programming languages, integration with other tools and platforms, and the ability to run tests on different devices and browsers.

**2. Analysis of existing solutions**

After defining the requirements and goals, it is important to analyze the existing test automation tools and frameworks on the market. Consider popular and proven solutions that meet your requirements.

**Basic steps:**

- Explore available tools and frameworks such as Selenium, JUnit, TestNG, Appium, and more.
- Evaluate the advantages and disadvantages of each tool based on your team's requirements.
- Study the experience of other teams and feedback on the selected tools to understand how they work in practice.

**3. Tool Testing and Evaluation**

Before making a final selection of tools and frameworks, it is important to test and evaluate their performance in real-world conditions. This will ensure that the solutions you choose are truly right for your team and meet all requirements.

**Basic steps:**

- Conduct pilot projects using selected tools to assess their effectiveness and usability.
- Assess how easy it is to integrate tools into your existing development and CI/CD process.
- Pay attention to the performance of the tools, their scalability, and support from the community or developers.

**4. Integration with existing processes and tools**

To successfully implement your automation strategy, it's important to integrate the tools you choose with the processes and tools you already have in your team. This will ensure seamless operation and maximum automation efficiency.

**Basic steps:**

- Set up tool integrations with issue management systems like Jira to track progress and manage testing.

- Integrate automation tools with CI/CD systems such as Jenkins to automatically run tests whenever code changes.
- Ensure that tools are compatible with the programming languages, platforms, and devices used.

**5. Valuation and Licensing Conditions**

Another important aspect when choosing tools and frameworks is to assess their cost and licensing conditions. It's important to consider not only upfront costs, but also potential maintenance, support, and upgrade costs.

**Basic steps:**

- Compare the cost of licenses for different tools and choose the ones that fit your team's budget.
- Evaluate the availability of free or open source alternatives such as Selenium, Appium, and more.
- Review the licensing terms and ensure that they meet your team's needs and requirements.

**6. Support and documentation**

Tools and frameworks with good support and documentation make the implementation and use process much easier. This is especially important for test automation, as having detailed documentation and an active community can help resolve issues quickly.

**Basic steps:**

- Assess the quality of your tools' documentation by reviewing the official manuals, use cases, and API documentation.
- Explore forums, communities, and resources about the tools you choose to make sure you have active support.
- Determine whether support is provided by the developers of the tool, and what communication channels are available to resolve technical issues.

**7. Integration and Expansion Options**

Test automation tools and frameworks should be flexible and easily integrated with other systems and tools used by your team. This allows for more efficient and seamless collaboration during the development process.

**Basic steps:**

- Evaluate the possibilities of integrating tools with issue management systems, CI/CD, and other tools such as Jira, Jenkins, and others.
- Check for available plugins and extensions that can improve the functionality of the selected tools.
- Ensure that the tools support integration with the programming languages and platforms used.

**8. Team Training**

Properly training the team to work with the selected tools and frameworks is a critical step for successful test automation. Training helps the team quickly adapt to new tools and effectively use their capabilities.

**Basic steps:**

- Organize training and workshops for the team to introduce them to the selected tools and frameworks.
- Support the learning process and knowledge sharing within the team through regular meetings and discussions.
- Create a knowledge base and documentation on how to use the tools that will be available to the entire team.

So, choosing the right tools and frameworks for test automation in an agile environment requires a thorough analysis of the team's requirements, goals, and capabilities. It is important to consider many factors, such as support, integration, cost, and extensibility. Choosing the right tools helps the team effectively automate testing processes, ensure high product quality, and achieve their goals.

## Prepare your infrastructure for automation

To successfully implement test automation in an agile environment, you need to carefully prepare the infrastructure that will support all aspects of this process. Infrastructure provisioning includes selecting and configuring tools, creating test environments, providing access to the necessary resources, and integrating automation into continuous integration and delivery (CI/CD) processes. These elements are critical to creating a sustainable and effective automation strategy that will maintain high product quality and ensure system stability.

In an agile environment where development cycles are short and intensive, test automation infrastructure must be flexible, scalable, and reliable. It should be able to run

tests quickly, minimize manual steps, and integrate with existing development processes. This allows the team to focus on core tasks and quickly identify and resolve defects.

In this section, we'll look at the key steps and best practices for preparing your infrastructure for test automation. We will discuss the selection and configuration of tools, the creation of test environments, resource access management, and the integration of automation into CI/CD processes. Following these steps will help your team build an effective infrastructure that supports continuous improvement of product quality and meets the requirements of the Agile methodology.

**1. Selection and Configuration of Automation Tools**

The first step in provisioning your infrastructure is to select and configure test automation tools. This includes installing the necessary software tools, configuring their settings, and integrating them with other systems used in the team.

**Basic steps:**

- Identify key test automation tools such as Selenium for web testing, JUnit or TestNG for unit testing, and Appium for mobile app testing.
- Install and configure the selected tools on all workstations and servers where you will run the tests.
- Integrate automation tools with task management systems (e.g., Jira) and CI/CD (e.g., Jenkins) to ensure that tests are automatically run whenever code changes.

**2. Create test environments**

Test environments should reflect the actual operating conditions of the product as closely as possible so that the test results are relevant and accurate. Test environments should be isolated from production to avoid the impact of testing on real data and users.

**Basic steps:**

- Create and configure test environments that include all the necessary system components, such as application servers, databases, and APIs.
- Isolate test environments from production environments to ensure that tests do not impact real data and users.
- Automatically update test environments to keep them up-to-date and up to date with current development requirements.

**3. Ensuring access to the necessary resources**

For successful test automation, it is important to ensure that you have access to the necessary resources, such as test data, configuration files, and libraries. This allows testers and developers to work efficiently with tests and get relevant results.

**Basic steps:**

- Create and maintain up-to-date test datasets that will be used in automated tests.
- Provide access to the configuration files and libraries you need to run your tests.
- Set up access control to resources to ensure security and control over data usage.

**4. Integration of automation into CI/CD**

Integrating test automation into your continuous integration and delivery (CI/CD) process is a key element of a successful automation infrastructure. This allows you to automate the execution of tests every time the code changes, which ensures that defects are identified and corrected in a timely manner.

**Basic steps:**

- Set up a CI/CD system such as Jenkins, GitLab CI, or Travis CI to automatically run tests on every commit or pull request.
- Integrate automated tests into your pipeline CI/CD so that tests run automatically at every stage of build and deployment.
- Set up notifications and reports on test results so that the team is informed about identified defects and issues in a timely manner.

**5. Ensure scalability and reliability**

Your test automation infrastructure needs to be scalable and reliable to support test execution at different stages of development and as the workload increases. This includes the use of cloud services, virtualization, and other technologies.

**Basic steps:**

- Consider using cloud services to run tests, such as AWS, Azure, or Google Cloud, to ensure scalability and availability.
- Use virtual machines or containers (such as Docker) to create isolated and reproducible test environments.
- Set up infrastructure monitoring and logging to proactively identify and resolve issues that affect test execution.

**6. Test Data Management**

Effective test data management is an important part of preparing your infrastructure for automation. Test data should be current, reliable, and accessible for test execution.

**Basic steps:**

- Create and maintain databases with test data that will be used in automated tests.
- Ensure that test data is regularly updated to reflect current scenarios and requirements.
- Set up access control for test data to ensure security and control over data usage.

**7. Ensuring Security and Privacy**

Test automation infrastructure must ensure data security and confidentiality, especially when tests involve sensitive information.

**Basic steps:**

- Set up security measures to protect test data, such as encryption, access control, and activity monitoring.
- Ensure compliance with privacy and data protection requirements such as GDPR or HIPAA, if applicable.
- Develop and implement procedures to securely dispose of test data after tests are completed.

Preparing your infrastructure for Agile test automation is a key step to successfully implementing an automation strategy. This includes selecting and configuring tools, creating test environments, managing test data, integrating automation into CI/CD, and ensuring data security and privacy. Following these steps will help your team create an efficient and reliable infrastructure that maintains high product quality and meets the requirements of the Agile methodology.

## Integration of automation into CI/CD

CI/CD (Continuous Integration, Continuous Delivery) is a technology for automating testing and delivering new modules of the project under development to stakeholders (developers, analysts, quality engineers, end users, etc.).

The concept of continuous integration and deployment refers to agile software development methodologies. Its main goal is to pay sufficient attention to business requirements, security, and code quality of the final product. Within the framework of the approach, the following tasks are solved:

- automation of sequential assembly, packaging and testing of software products;
- automate application deployment across environments
- minimization of errors and vulnerabilities of the software product.

CI/CD development follows the following basic principles:

- **Distribution of responsibility.** Tasks and stages of development are divided between team members or its subgroups (when working on a large project). The workflow is organized taking into account business logistics, the implementation of end-to-end functions, testing, data storage security, etc.
- **Reduce risk.** Each developer or subgroup of developers should strive to minimize vulnerabilities and errors at all stages of development. To do this, business logistics are constantly monitored, user testing of the product is carried out, data storage and processing are optimized, etc.
- **Optimization of feedback.** The success of a project depends on how developers, clients, and users work with each other. This affects the speed of making adjustments and updates to the application. While building and testing can be automated, many other operations require human participation.
- **Create a work environment.** For the convenience of collaboration, developers should have a common workspace. In addition to the main branch of the process, it should have a side branch - it is more convenient to conduct testing, make adjustments, track fault tolerance, etc.

CI/CD System Setup and Tool Selection:

The first step in integrating test automation into CI/CD is to select and configure the right system for continuous integration and delivery. There are many CI/CD tools and platforms that can be used depending on your team's needs and infrastructure.

**Basic steps:**

1. Choosing a CI/CD System:
    - Assess your team's needs and choose the right CI/CD system. Popular solutions include Jenkins, GitLab CI, Travis CI, CircleCI, and more.
    - Consider factors such as integration capabilities, cost, support, and documentation when choosing a tool.
2. Installation and configuration of the system:
    - Install your chosen CI/CD system on a server or use cloud solutions for rapid deployment.

- Set up the basic configuration of the system, including integration with version control systems (e.g., Git) and issue management (e.g., Jira).

3. Creating a pipeline for CI/CD:
    - Define the main stages of the pipeline for continuous integration and delivery. This can include the build, test, deployment, and notification phases.
    - Create configuration files for the pipeline that define the sequence of stages and the commands that will be executed at each stage.
4. Integration with version control systems:
    - Integrate your CI/CD system with your version control system (e.g., GitHub, GitLab, or Bitbucket).
    - Ensure that the pipeline starts automatically whenever the code changes, such as committing or creating a pull request.
5. Setting up notifications:
    - Set up notifications and alerts to inform the team about the status of the pipeline and test results.
    - Use notification tools like Slack, Email, or other messaging apps to promptly inform your team of issues and defects.
6. Define the test execution environment:
    - Define and configure test execution environments, such as virtual machines or containers (such as Docker), to ensure test isolation and reproducibility.
    - Set up automatic creation and updates of test environments to keep them up to date.

```
pipeline {
    agent any
    stages {
        stage('Checkout') { // Checkout stage
            steps {
                Check out the latest version of code from a version
control system (for example, Git)
                checkout scm
            }
        }
        stage('Build') { // Stage "Build"
            steps {
```

```
                Running a Project Build Command Using Maven
                sh 'mvn clean package'
            }
        }
        stage('Test') { // Stage "Test"
            steps {
                Running project tests using Maven
                sh 'mvn test'
            }
        }
        stage('Deploy') { // Stage "Deploy"
            steps {
                Deploy a Collected Artifact
                sh 'mvn deploy'
            }
        }
    }
    post {
        always {
            Analyze test results and generate a report
            junit '**/target/surefire-reports/*.xml'
            Archiving Assembly Artifacts
            archiveArtifacts artifacts: '**/target/*.jar',
fingerprint: true
   }

      success {
            Sending a notification to Slack in case of a successful
build
          slackSend channel: '#ci-cd', color: 'good', message:
"Build succeeded: ${currentBuild.fullDisplayName}"
        }
        failure {
            Send a notification to Slack in case of a failed build
            slackSend channel: '#ci-cd', color: 'danger', message:
"Build failed: ${currentBuild.fullDisplayName}"
        }
    }
}
```

In this configuration file for Jenkins:

1. **'Checkout' stage:**
    - The latest version of the code is checked out from a version control system (for example, Git).
    - The checkout scm command is used, which automatically pulls the code from the repository configured in Jenkins.
2. **Stage 'Build':**
    - A project build command is running using Maven.
    - Use the sh 'mvn clean package' command to clean up previous builds and package the project.
3. **The 'Test' stage:**
    - Project tests are performed using Maven.
    - Use the sh 'mvn test' command to run all the tests in the project.
4. **The 'Deploy' stage:**
    - Deploy the assembled artifact.
    - Use the sh 'mvn deploy' command to upload the artifact to a remote repository or server.
5. **The 'post' stage:**
    - Actions are performed after all stages of the assembly are completed:
        - Analyze the test results and generate a report using the junit '**/target/surefire-reports/*.xml' command.
        - Archiving assembly artifacts using the command archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true.
6. **Notifications:**
    - Send a notification to Slack if the build is successful using the command slackSend channel: '#ci-cd', color: 'good', message: "Build succeeded: ${currentBuild.fullDisplayName}".
    - Send a notification to Slack in case of a failed build using the command slackSend channel: '#ci-cd', color: 'danger', message: "Build failed: ${currentBuild.fullDisplayName}".

**Develop and manage automated tests**

Developing and managing automated tests are key aspects of successfully integrating automation into your CI/CD processes. In this article, we will look at how to create and maintain automated tests, as well as how to organize their execution within CI/CD.

**Basic steps:**

1. **Development of automated tests:**
    - Determine the types of testing that will be automated. These can be unit tests, integration tests, functional tests, regression tests, and end-to-end tests.
    - Use the right tools and frameworks to develop automated tests. For example, JUnit or TestNG for unit tests, Selenium for web tests, Appium for mobile tests, and JMeter for performance testing.
    - Create detailed and readable test cases that cover all critical product features and use cases. Ensure that you have the right test data and predictable results.
2. **Organization of the test structure:**
    - Organize tests into logical groups to make them easier to run and manage. For example, you can divide tests into modules, components, or functional areas.
    - Use code naming and organization standards for automated tests to ensure they are readable and supported. This includes naming tests, methods, and classes, and using comments and documentation.
3. **Test Data Management:**
    - Ensure that you have up-to-date and reliable test data to run automated tests. This can be either static data or dynamically generated data.
    - Set up procedures for creating, updating, and cleaning test data to ensure repeatable tests and avoid data contamination.
4. **Integration of tests into the CI/CD pipeline:**
    - Add automated tests to your CI/CD pipeline so that they run automatically at every stage of the build and deployment process. For example, you can configure unit tests to run during the build phase, integration tests during the testing phase, and end-to-end tests during the pre-release deployment phase.

- Configure test execution settings such as parallel execution, timeouts, and priority levels to optimize resource utilization and speed up test execution.

5. **Reports and analysis of test results:**
    - Generate reports on test results so your team can quickly analyze the results and identify defects. Use reporting tools like the Surefire Report Plugin for Maven, Allure for beautiful reports, or the built-in capabilities of CI/CD systems.
    - Analyze test results to identify recurring issues, defect patterns, and areas for improvement. Use your data to make informed decisions and improve your development and testing processes.

**Sample test integration in Jenkins**

Let's take a look at an example of a configuration file for Jenkins that includes running automated tests at different stages of the pipeline:

```
pipeline {
    agent any
    stages {
        stage('Checkout') { // Checkout stage
            steps {
                Check out the latest version of code from a version
control system (for example, Git)
                checkout scm
            }
        }
        stage('Build') { // Stage "Build"
            steps {
               Running a Project Build Command Using Maven
                sh 'mvn clean package'
            }
        }
        stage('Unit Tests') { // Stage "Unit Tests"
            steps {
                Performing unit tests of the project using Maven
                sh 'mvn test'
            }
        }
        stage('Integration Tests') { // Stage "Integration Tests"
            steps {
```

```
                Run integration tests using Maven's
"integration-tests" profile
                sh 'mvn verify -Pintegration-tests'
            }
        }
        stage('End-to-End Tests') { // Stage "End-to-End Tests"
            steps {
                Run end-to-end tests using Maven's end-to-end-tests
profile
                sh 'mvn verify -Pend-to-end-tests'
            }
        }
    }
    post {
        always {
            Analyze test results and generate a report using the
Surefire Report Plugin for Maven
            junit '**/target/surefire-reports/*.xml'
            Archiving build artifacts (such as JAR files) for future
use
            archiveArtifacts artifacts: '**/target/*.jar',
fingerprint: true
        }
        success {
            Send a notification to Slack if a build is successful
with the build name
            slackSend channel: '#ci-cd', color: 'good', message:
"Build succeeded: ${currentBuild.fullDisplayName}"
        }
        failure {
            Send a notification to Slack in case of a failed build
with the build name
            slackSend channel: '#ci-cd', color: 'danger', message:
"Build failed: ${currentBuild.fullDisplayName}"
        }
    }
}
```

In this configuration file for Jenkins:

1. **'Checkout' stage:**

- The latest version of the code is checked out from a version control system (for example, Git).
- The checkout scm command is used, which automatically pulls the code from the repository configured in Jenkins.

2. **Stage 'Build':**
   - A project build command is running using Maven.
   - Use the sh 'mvn clean package' command to clean up previous builds and package the project.
3. **'Unit Tests' stage:**
   - Unit tests of the project are performed using the sh 'mvn test' command.
4. **'Integration Tests' stage:**
   - Integration tests are run using the Maven integration-tests profile.
5. **'End-to-End Tests' stage:**
   - End-to-end tests are performed using the Maven end-to-end-tests profile.
6. **The 'post' stage:**
   - Actions are performed after all stages of the assembly are completed:
     - Analyze the test results and generate a report using the junit '**/target/surefire-reports/*.xml' command.
     - Archiving assembly artifacts using the command archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true.
7. **Notifications:**
   - Send a notification to Slack if the build is successful using the command slackSend channel: '#ci-cd', color: 'good', message: "Build succeeded: ${currentBuild.fullDisplayName}".
   - Send a notification to Slack in case of a failed build using the command slackSend channel: '#ci-cd', color: 'danger', message: "Build failed: ${currentBuild.fullDisplayName}".

These phases allow you to integrate different kinds of automated tests into your CI/CD pipeline, enabling continuous testing and quality assurance at every stage of development.

**Optimize test execution**

Streamlining test execution in the CI/CD process is key to ensuring fast and efficient testing. In this article, we will look at various methods and techniques that will help improve the performance and reduce the execution time of automated tests, as well as ensure their stability and reliability.

**Basic steps:**

1. **Parallel Test Execution:**
    - Configure parallel test execution to run multiple test cases at the same time. This significantly reduces the overall time to run tests and allows you to get results faster.
    - Leverage the capabilities of CI/CD tools for parallel execution, such as parallel steps in Jenkins or parallel tasks in GitLab CI.
2. **Distributed testing:**
    - Implement distributed testing so that tests run on multiple machines or containers. This allows for more efficient use of resources and faster execution of large volumes of tests.
    - Set up a distributed testing system such as Selenium Grid for web testing or distributed environments based on Docker and Kubernetes.
3. **Optimization of test scenarios:**
    - Analyze existing test cases and determine which ones can be optimized. Eliminate duplicate tests, reduce the time it takes to run individual tests, and use efficient data generation techniques.
    - Use equivalent class and boundary value techniques to minimize the number of tests while maintaining coverage.
4. **Caching and reusing results:**
    - Configure caching and reuse of test results so that you don't run retests on immutable parts of your code. This can significantly reduce the execution time of the CI/CD pipeline.
    - Leverage the capabilities of CI/CD tools for caching artifacts, such as Jenkins Pipeline Caching or GitLab CI Cache.
5. **Dynamic management of test environments:**
    - Set up dynamic creation and deletion of test environments based on your current needs. This allows you to use resources efficiently and avoid unnecessary costs for maintaining unused environments.

- Implement automation to create and delete virtual machines or containers using tools such as Terraform or Ansible.

6. **Test monitoring and logging:**
    - Set up monitoring and logging of test execution to quickly identify problems and analyze their causes. Use monitoring tools such as Prometheus or Grafana, as well as logging systems such as ELK Stack (Elasticsearch, Logstash, Kibana).
    - Implement alerting and notifications to inform your team of issues and failures when running tests.

Let's take a look at an example of a configuration file for Jenkins that involves running tests in parallel at different stages of the pipeline:

```
pipeline {
    agent any
    stages {
        stage('Checkout') { // Checkout stage
            steps {
                Check out the latest version of code from a version
control system (for example, Git)
                checkout scm
            }
        }
        stage('Build') { // Stage "Build"
            steps {
                Running a Project Build Command Using Maven
                sh 'mvn clean package'
            }
        }
        stage('Parallel Tests') { // Stage "Parallel Tests"
            parallel {
                stage('Unit Tests') { // Substage "Unit Tests"
                    steps {
                        Performing unit tests of the project using
Maven
                        sh 'mvn test'
                    }
                }
                stage('Integration Tests') { // Substage
"Integration Tests"
```

```
                    steps {
                        Run integration tests using Maven's
"integration-tests" profile
                        sh 'mvn verify -Pintegration-tests'
                    }
                }
                stage('End-to-End Tests') { // Sub-stage "End-to-End
Tests"
                    steps {
                        Run end-to-end tests using Maven's
end-to-end-tests profile
                        sh 'mvn verify -Pend-to-end-tests'
                    }
                }
            }
        }
    }

    post {
        always {
            Analyze test results and generate a report using the
Surefire Report Plugin for Maven
            junit '**/target/surefire-reports/*.xml'
            Archiving build artifacts (such as JAR files) for future
use
            archiveArtifacts artifacts: '**/target/*.jar',
fingerprint: true
        }
        success {
            Send a notification to Slack if a build is successful
with the build name
            slackSend channel: '#ci-cd', color: 'good', message:
"Build succeeded: ${currentBuild.fullDisplayName}"
        }
        failure {
            Send a notification to Slack in case of a failed build
with the build name
            slackSend channel: '#ci-cd', color: 'danger', message:
"Build failed: ${currentBuild.fullDisplayName}"
        }
    }
```

```
}
```

In this configuration file for Jenkins:

1. **'Checkout' stage:**
    - The latest version of the code is checked out from a version control system (for example, Git).
    - The checkout scm command is used, which automatically pulls the code from the repository configured in Jenkins.
2. **Stage 'Build':**
    - A project build command is running using Maven.
    - Use the sh 'mvn clean package' command to clean up previous builds and package the project.
3. **'Parallel Tests' stage:**
    - Includes parallel execution of three sub-stages: 'Unit Tests', 'Integration Tests' and 'End-to-End Tests'.
    - Parallel execution can reduce the overall time to run tests and speed up time to results.
4. **Sub-stage 'Unit Tests':**
    - Unit tests of the project are performed using the sh 'mvn test' command.
5. **'Integration Tests' sub-stage:**
    - Integration tests are run using the Maven integration-tests profile.
6. **'End-to-End Tests' sub-stage:**
    - End-to-end tests are performed using the Maven end-to-end-tests profile.
7. **The 'post' stage:**
    - Actions are performed after all stages of the assembly are completed:
        - Analyze the test results and generate a report using the junit '**/target/surefire-reports/*.xml' command.
        - Archiving assembly artifacts using the command archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true.

8. **Notifications:**
    - Send a notification to Slack if the build is successful using the command slackSend channel: '#ci-cd', color: 'good', message: "Build succeeded: ${currentBuild.fullDisplayName}".
    - Send a notification to Slack in case of a failed build using the command slackSend channel: '#ci-cd', color: 'danger', message: "Build failed: ${currentBuild.fullDisplayName}".

These steps help streamline test execution and integrate them into the CI/CD process, ensuring continuous testing and quality assurance at every stage of development.

**Manage and monitor automated tests**

Managing and monitoring automated tests are key aspects of successfully integrating automation into your CI/CD processes. In this article, we will look at how to organize test management and set up monitoring to ensure the stability and reliability of automated testing.

**Basic steps:**

1. **Test Case Management:**
    - Create a framework for managing test cases that includes versioning and change management. This helps you track changes to your tests and ensure that they are up to date.
    - Use a version control system (such as Git) to store and manage test scripts, and to collaborate on tests.
2. **Configuring Test Execution Monitoring:**
    - Monitor test execution to track test status and identify issues in a timely manner. Use monitoring tools like Prometheus and Grafana to collect and visualize metrics.
    - Set up alerting and notifications to keep your team informed of failures and issues when running tests. This allows you to quickly respond to emerging problems and take action to eliminate them.
3. **Logging and analysis of test results:**
    - Set up logging of test results to be able to analyze them and identify the causes of failures. Use logging systems such as ELK Stack (Elasticsearch, Logstash, Kibana) to store and analyze logs in one place.

- Regularly analyze test results to identify defect patterns and areas for improvement. Use the data to optimize test scenarios and improve development processes.

4. **Ensuring the stability and reliability of tests:**
    - Implement practices to ensure the stability and reliability of automated tests. This includes regularly updating test data, configuring test isolation, and minimizing dependencies between tests.
    - Conduct regular checks and updates of test cases to ensure that they remain current and meet current product requirements.
5. **Feedback and process improvement:**
    - Set up processes to get feedback on automated test results. This can include regular team meetings, retrospectives, and discussions of test results.
    - Use the feedback you receive to continuously improve your testing and automation processes. Implement new practices and tools to improve the efficiency and quality of testing.

**Sample test monitoring configuration using Prometheus and Grafana**

Let's look at an example configuration for monitoring test execution using Prometheus and Grafana. In this example, we'll set up metrics collection and visualization.

**Prometheus Setup Steps:**

1. **Installing and configuring Prometheus:**
    - Install Prometheus and configure the prometheus.yml configuration file to collect metrics.
    - Add a configuration to collect test execution metrics:

```
scrape_configs:
  - job_name: 'jenkins'
    static_configs:
      - targets: ['localhost:8080']
```

2. **Setting up metric exporters:**
    - Install and configure metric exporters to collect data from Jenkins and other systems used for automated testing.

**Grafana Setup Steps:**

1. **Installing and configuring Grafana:**

- Install Grafana and set up a connection to Prometheus as a data source.
- Go to the "Configuration" section and add a new Prometheus data source.

2. **Creating dashboards for visualizing metrics:**
    - Create dashboards in Grafana to visualize test execution metrics. This can include graphs of test status, run time, and the number of tests that pass and fail.
    - Set up dashboards to display important metrics and set alerting to keep your team informed of issues in a timely manner.

Thus, the integration of test automation into CI/CD processes plays a key role in ensuring high product quality and prompt detection of defects. For a successful integration, it is important to consider the following aspects:

1. **CI/CD System Setup and Tool Selection:** Selecting the right CI/CD system, installing and configuring tools, creating and configuring a pipeline.
2. **Development and management of automated tests:** Creation of test scripts, test structure, test data management, integration of tests into the pipeline.
3. **Optimize test execution:** Parallel test execution, distributed testing, test script optimization, caching and reusing results, dynamic management of test environments.
4. **Manage and monitor automated tests:** Manage test scenarios, set up monitoring and logging, ensure test stability and reliability, get feedback, and improve processes.

Following these steps will help your team build an efficient and reliable infrastructure for test automation, ensuring high product quality and compliance with the requirements of the Agile methodology.

# Chapter 6. Integrate testing into sprints

In Agile methodology, testing is an integral part of the development process, ensuring high product quality and compliance with customer requirements. It is important that testing is integrated at all stages of development, from planning to retrospective. This allows you to identify and correct defects at an early stage, minimize risks and ensure the stability of the system.

Integrating testing into sprints involves close interaction between testers and developers, analysts, and product owners. Testers should be active team members involved in planning, developing, and evaluating the results of the sprint. This allows for an atmosphere of cooperation and mutual understanding, which contributes to more effective work and the achievement of common goals.

In this chapter we consider how testing can be integrated into each stage of the sprint to ensure continuous quality control and product consistency. We'll discuss the roles and responsibilities of testers in the sprint, testing methods and techniques, and best practices to help your team achieve high results. This chapter will help you understand the importance of integrating testing into sprints and provide guidance on how to successfully implement this approach into your Agile team.

## Sprint-level test planning

Sprint-level test planning is an important step in Agile methodology that helps the team clearly understand tasks, assign roles, and perform testing effectively. This process

requires close collaboration between testers, developers, and product owners to create and agree on test plans that cover all aspects of the sprint.

**1. Participation of testers in sprint planning**

Testers should be active participants in sprint planning to ensure that testing is integrated at all stages of development. Their participation makes it possible to identify potential problems in advance, clarify requirements and ensure that acceptance criteria are agreed.

*What you need to do:*

- Testers attend sprint planning meetings to understand sprint goals and objectives, as well as participate in user story and task discussions.

- Collaborate on acceptance criteria for each user story to determine which tests need to be run to validate functionality and compliance.

- Discussion and assessment of risks associated with testing, and planning actions to minimize them.

**2. Development of test plans**

Developing test plans at the sprint level includes defining the types of testing that will be carried out, as well as creating detailed test scenarios and cases. This allows you to ensure that all tasks and user stories are fully covered by tests.

*Recommended actions:*

- Identify the types of testing, such as unit testing, integration testing, regression testing, and end-to-end testing, that need to be done as part of the sprint.

- Create test cases and cases for each user story and task to ensure that all functional and non-functional requirements are validated.

- Prioritizing testing and assigning tasks to team members to optimize resource and time usage.

**3. Integration of automated tests**

Test automation plays a key role in an agile environment, allowing you to speed up the testing process and increase its efficiency. It's important to plan for the integration of automated tests in each sprint to ensure continuous quality control.

*How to proceed:*

- Define tasks for automation and create automated tests for regression, integration, and end-to-end testing.

- Integration of automated tests into the CI/CD process for automatic execution with every code change.

- Maintain and update automated tests in each sprint to ensure they are up-to-date and up to date.

**4. Evaluation and adjustment of the test plan**

The test plan should be flexible and adaptable to changes that may occur during the sprint. It's important to regularly evaluate and adjust your test plan to account for new challenges, changes in requirements, and identified risks.

*Corrective actions:*

- Holding regular team meetings to discuss testing progress, identified issues, and necessary plan adjustments.

- Evaluation of test results and adjustment of plans based on feedback and data analysis.

- Adapting the test plan to changes in tasks and requirements to ensure continuous quality control and meeting customer expectations.

Therefore, sprint-level test planning is an important element of success in an agile team. It includes active participation of testers in planning, the development of detailed test plans, the integration of automated tests, and regular evaluation and adjustment of the plan. These steps ensure high product quality, prompt identification and correction of defects, as well as compliance with customer requirements and expectations.

## Automation priorities

Test automation is a powerful tool for improving the efficiency of QA processes, but not every task or scenario makes sense to automate. Prioritizing automation correctly saves resources by focusing on areas where it will deliver the most value. Let's look at the basic principles of prioritizing automation with examples.

**Recurring tasks**

Automation has the greatest effect for tasks that are repeated from version to version. For example, checking the basic functionality after each code change. Such scenarios often include smoke and regression tests that provide confidence in the stability of the application.

*Example:*

- Smoke tests: check that the application is running, the main functions are working, and the data is successfully saved.

- Regression tests: Checking old functionality to ensure that it remains functional after implementing new changes.
- Verify that the API is working correctly: Validate that requests and responses remain unchanged.

**Complex, labor-intensive scenarios**

If running a manual test is time-consuming or requires complex sequences of steps, automation can help reduce the time required and reduce the risk of tester errors.

*Example:*

- Checking the checkout process on the e-commerce platform, including adding an item to the cart, calculating the cost, entering data for delivery and payment.
- Scenarios with many steps that depend on different user roles, such as creating and approving documentation in enterprise systems.

**Business-critical features**

Automating tests that test key features that directly impact revenue or user experience should be a priority.

*Example:*

- Checking the processing of payments in the banking application.
- Testing the ticket booking function in the travel service.
- Verify report generation in data-intensive financial systems.

**Tests related to different configurations**

If your app runs in different environments or needs to support multiple combinations of browsers, operating systems, and devices, automation can help reduce the time it takes to run tests for all of these configurations.

*Example:*

- Testing the web application on different browsers: Chrome, Firefox, Safari.
- Checking the mobile app on Android and iOS devices.
- Validation of distributed systems under different network settings and restrictions.

**Non-functional requirements**

The performance, security, and scalability of the application require an automated approach due to the complexity and volume of data that needs to be processed. Such tests provide confidence that the system will cope with the load and is protected from threats.

*Example:*

- Stress tests to determine the maximum load on the server.
- Security tests to check for vulnerabilities such as SQL injection or cross-site scripting.
- Load tests to analyze the response time of the system under increased load.

**Tests with long-term value**

Automation is especially useful in scenarios where tests will be used for a long time and will only be updated when there are significant changes. Such tests often accompany the product throughout its life cycle.

*Example:*

- Checking basic user flows, such as authorization, registration, and password recovery.
- Checking functionality related to the integration of external services, for example, payment systems or APIs of third-party providers.

**Low priority for automation**

Automation is not always justified. One-time scenarios, complex tests with minimal repeatability, or checks that require visual evaluation are best left for manual testing.

*Example:*

- Checking the creative design of the user interface.
- One-time checks during the application behavior investigation.
- Tests where automation takes more time and resources than manual execution.

Automation priorities depend on project goals, budget, and available resources. The ability to prioritize correctly allows you to use automation most effectively, minimizing costs and ensuring maximum benefit to product quality. Thanks to a competent approach to the choice of tests, automation becomes a tool that not only speeds up the testing process, but also increases its reliability. This ultimately contributes to the successful implementation of the project, providing users with a high quality product and a positive experience.

## Implementation of testing in Definition of Done (DoD)

Definition of Done (DoD) is an important tool in the Agile methodology that helps the team define completion criteria and set quality standards. Incorporating testing into DoD

ensures continuous quality control at every stage of development and ensures that every completed task meets high standards. Implementing testing in DoD helps the team avoid misunderstandings, minimize risks, and ensure the stability and reliability of the product.

**Who determines the readiness criteria?**

The product owner, product manager, development team, and Scrum Master work together during sprint planning to establish the Definition of Done. Defining completion criteria is a collaborative process, as each team member has unique knowledge that can be helpful. If multiple Scrum teams are working on a product, they all need to agree on what is considered "complete."

The Product Owner plays a key role in shaping the Definition of Done, ensuring that it meets stakeholder expectations and strategic business goals. The development team or technicians contribute their peer review to refine the criteria, and product managers analyze the requirements to ensure they are consistent. The Scrum Master facilitates teamwork by eliminating possible misunderstandings in the definition. Ultimately, the entire Agile team is responsible for creating a clear and unambiguous definition of Definition of Done.

**How to Create a Definition of Done**

Here are some key steps to create a Definition of Done:

1. Gather a team for a comprehensive discussion of customer needs and product or project requirements.
2. Jointly define specific criteria, including functional, technical, and quality standards.
3. Engage other teams and stakeholders to ensure cross-team collaboration and alignment.
4. Document the criteria you set. Summarize them briefly to make them easier to understand.
5. Strive for continuous improvement by regularly reviewing and refining the DoD.

**When are the completion criteria considered approved?**

The Definition of Done is generated by the team during the sprint planning process. The product owner is responsible for refining the backlog and adding new items to the plan for the next sprint. The main goal is to reach a common understanding of these elements before the start of the sprint.

During planning, the team defines specific activities and criteria for each user story or product. This helps to maintain a stable quality of the project.

**Common mistakes**

Sometimes it can be difficult for team members and stakeholders to agree on a Definition of Done. Differences in perceptions, levels of experience, and approaches can make the process difficult. Problems can also be caused by insufficient communication, lack of time, or lack of clarity. Here are the main mistakes:

1. **Ignoring criteria**: In a hurry, important aspects can be missed. This leads to incomplete or low-quality work, customer dissatisfaction, and the need for improvements.
2. **Excessive attention to detail**: Perfectionism can slow down the process. Teams need to find a balance between quality and efficiency without getting caught up in the nitty-gritty.
3. **Specific criteria for features**: Individual features may have unique criteria or dependencies. It is important for the team to clearly define these features to avoid confusion during task execution.
4. **Ignoring the Retrospective**: For continuous improvement, teams should conduct regular sprint retrospectives. This helps identify opportunities to improve the performance of the Definition of Done.

**Definition of Done Role and Use Cases**

The easiest way to explain the meaning of the term Definition of Done is with an everyday example. Let's say there are 3 housewives in the kitchen: a daughter, a mother and a grandmother. For everyone, the phrase "dinner is ready" implies a different thing. For grandma, "ready" is a dish that has just been sent to bake, for mom it is a dish that was taken out of the oven on a baking sheet. And for my daughter, "dinner is ready" is when everything is already on the plates and on the table. The same is true in any other process where several team members are involved.

For example, Definition of Done (DoD) in IT for one developer can be expressed in the completion of the main phase of work, but before tests and code reviews. And for another, it is a feature that the client can open and use.

For the development process to be effective, it is important that all team members speak the same language and understand the "finished task" in the same way. This is why the term Definition of Done (DoD) appeared.

**Differences between Readiness Criteria and Acceptance Criteria**

- **Definition of Done** is a set of requirements that define when a task or project is considered complete by organizational standards. It includes minimum requirements

for completing work that meets quality standards and ensures consistency between different functions and stories.

- **Acceptance Criteria**, on the other hand, describe the specific conditions or functionality required for stakeholder adoption of a user story or feature.

The Definition of Done ensures that the work meets the basic standards, and the Acceptance Criteria detail the unique requirements for each aspect of the project.

**What is DOD in Scrum and other agile methodologies?** This is the name of an agreement that clearly describes a set of criteria for the readiness of each task. It is part of the Agile process, which is needed to ensure that each element of the product is completed at a certain level of quality before being handed over to a client or another employee.

Let's go back to the example of a family – they should draw up a DoD, which will clearly state what the finished dinner should look like. For example:

- baked meat;
- boiled potatoes;
- cut and mixed vegetables for salad;
- The table was laid.

Only after drawing up such an agreement will it be clear to everyone that it is time to sit down at the table at the phrase "dinner is ready".

A simple example of DoD in development:

- The code has been written and passed Code Review
- unit tests cover at least 80% of the code;
- integration tests have been passed;
- the function is QA tested and meets the acceptance criteria;
- Documentation on how to use the feature has been updated.
- The function is deployed on a Staging server.

**Why DoD is important to implement: key benefits**

If the team has created and implemented the Definition of Done, then this provides several advantages at once:

- **The quality of the product improves**. Working with DoD in mind ensures that every element of the product is at an agreed level of quality. And it is this level of quality that will be able to meet the needs of customers and increase their loyalty;

- **Risks are reduced**. If the team uses DoD, then the number of accidental defects, errors and flaws will be reduced in the work: the team will check how polished each element of the product is and whether it meets the required level of quality even before handing it over to the client;
- **Communication in the team improves**. Each employee perfectly understands what he must do and in what order to consider his task completed. With this approach, there are no misunderstandings, resentments, overloads or irresponsible behavior in the department: for each stage or action, there is a responsible person and inspection criteria.

Please note that DoD can be formed for each stage of development and even action, if it is easier for the team to work this way. Employees will be able to use DoD to define criteria for the completion of an item before moving to the next level, not just to review the final product. With this approach, you can intermediately check the quality and functionality of an element or function at each stage.

In addition, this approach allows you to clarify the deadlines for the delivery of the finished product - when the manager knows how much time is needed for each stage and how much it actually took, it is easier for him to predict when all the work will be completed.

The main advantage of Definition of Done is that the team can work with peace of mind and know that each stage and element is thought out, developed and tested, therefore completed correctly and in full.

**Disadvantages of DoD**

Definition of Done is an important and useful aspect for an agile development methodology, but it also has its drawbacks:

- **complexity and congestion**. If the DoD is too detailed and extensive, it can significantly complicate the development process. For example, the team will sit back and review each item of numerous requirements or explanations before passing on to the next stage of development;
- **inconsistency**. If the document is ambiguous, then this will slow down the whole work - each team member will perceive readiness in his own way, this will lead to conflicts and misunderstandings;
- **changes in the process**. If you start changing the requirements in the DoD, it can cause confusion and reduce the credibility of the document. You should be careful with changes and make them only if necessary;
- **limitation of creativity**. If the DoD is written for creative work, it can limit the team in finding new approaches and solutions;

- **Focus on the process**. Sometimes the team can focus only on the formal implementation of each DoD item and forget that the main goal of the work is to create value for the user, and not to comply with the lines from the checklist. This is how projects that do not bring real benefits appear.

Definition of Done is a useful tool that can be used to increase productivity and transparency, reduce turnaround time, and make all work unambiguous. But it needs to be used wisely and adapted to the needs of the project, team, and clients.

- After that, you need to discuss the project with the team. This will help to take into account:
- experience and suggestions from employees;
- possible risks and technical limitations that have not been identified earlier.
- When this is ready, you can proceed to document all the agreed criteria in a format that is accessible to each participant.

Implementing testing in Definition of Done (DoD) is critical to ensuring high product quality and customer compliance. This process includes defining mandatory types of testing, integrating automated tests, ensuring transparency and consistency within the team, and documenting and regularly updating the DoD. Following these principles allows the Agile team to achieve high results and create reliable and high-quality products.

# Chapter 7. Maintaining product quality in a fast-release environment

In a rapid release environment, product quality becomes a key factor for the successful completion of the project and customer satisfaction. Rapid releases require high flexibility, responsiveness, and effective risk management on the part of the development team. In such an environment, maintaining product quality plays a crucial role in ensuring stability, reliability, and meeting user expectations.

Rapid releases are usually accompanied by accelerated development cycles, increased volume of changes, and limited testing time. It is important to implement strategies and practices that will help maintain high product quality without compromising the speed of development. This requires close interaction between all team members, integration of testing at all stages of development, and the use of modern tools and technologies for automation and monitoring.

In this section, we'll look at the key aspects of maintaining product quality in a fast-release environment. We will discuss how to effectively organize testing and automation processes, how to manage risks and minimize defects, as well as best practices and approaches that will help your team achieve high results. These recommendations will ensure high quality of the product, even in conditions of limited time and resources.

## Technical debt management

Technical debt is a metaphor that describes the consequences of short-term decisions in software development, which over time can lead to an accumulation of problems and

hinder product support and development. Technical debt management plays a key role in ensuring the high quality and stability of the system, as well as in preventing the cost of maintaining and developing it.

**1. Determination and assessment of technical debt**

The first step in managing technical debt is to identify and evaluate it. It is important to understand where and why technical debt has accumulated, and how it affects the project. To do this, you need to audit the system and code, as well as take into account the opinion of the development team.

**System and code analysis:**

- Conduct regular code audits to identify areas of technical debt. These can be poorly documented code fragments, outdated libraries, workarounds, and crutches.
- Using automated tools for code analysis, such as SonarQube, PMD, and others. These tools can help you identify potential issues such as duplicate code, outdated methods, low test coverage, and more.

**Assessment of the impact of technical debt:**

- Assess the impact of technical debt on various aspects of the project, such as performance, security, stability, and scalability. Metrics such as load time, response time, error rate, and others can be used to do this.
- Conduct surveys and interviews with team members to understand which issues are most difficult and require immediate attention.

**Documentation of technical debt:**

- Creation of technical debt documentation, including a description of the identified problems, their impact on the project, priority and proposed solutions. This will help to systematize the work to eliminate technical debt and ensure transparency for all team members.

**2. Prioritization of tasks to eliminate technical debt**

Not all tasks to eliminate technical debt require immediate implementation. It's important to evaluate which tasks have the most importance and impact on the project and prioritize them accordingly.

**Classification and categorization:**

- Division of tasks into categories according to their importance and urgency. For example, critical tasks that affect the stability and security of the system, important tasks that improve performance and scalability, and less urgent tasks that improve code readability and documentation.

- Use a priority matrix to visualize tasks along the axis of importance and urgency. This will help the team focus on the most important tasks and allocate resources efficiently.

**Planning and distribution of tasks:**

- Creating a timeline for the completion of technical debt tasks, including a time frame, responsible persons and completion criteria. This will help organize the work and ensure control over the process.
- Integrate technical debt elimination tasks into the overall development plan and sprints. For example, allocate a certain percentage of time to work on technical debt in each sprint to avoid accumulating it.

**3. Integrate technical debt into sprints**

Dealing with technical debt should be part of a regular development process to ensure that issues are fixed systematically and continuously.

**To include tasks in sprints:**

- Include tasks to eliminate technical debt in the product backlog and evaluate them along with other tasks. This allows the team to pay enough attention to solving technical problems and not postpone them for later.
- Determining the time that the team will allocate to work on technical debt in each sprint. For example, allocating 10-20% of the time to tasks to eliminate debt and improve the code.

**Conducting reviews and retrospectives:**

- Regular code reviews to identify and eliminate technical debt. This allows the team to monitor the quality of the code and respond quickly to emerging issues.
- Conduct retrospectives after each sprint to discuss progress and identify areas for improvement. The feedback received will help optimize processes and improve code quality.

**4. Implementation of best development practices**

To prevent the accumulation of technical debt in the future, it is important to implement best development practices and follow them at all stages of work. This helps maintain high quality code and makes it easier to maintain and evolve.

**Development and implementation of standards:**

- Develop and implement coding standards that include rules and guidelines for writing code. This ensures consistency and high quality of the code.
- Define documentation standards so that each team member creates clear and detailed comments and documentation for their code.

**Application of testing methods:**

- Implement testing methodologies such as test-based testing (TDD) and behavior-based testing (BDD). This allows you to identify and eliminate errors at an early stage of development.
- Provide high code coverage with tests to minimize risks and ensure the stability and reliability of the system. Running automated tests regularly helps identify issues before they are released.

**Use of automated tools:**

- Implement automated tools for static code analysis, such as SonarQube, ESLint, or Pylint. These tools allow you to identify potential problems and violations of coding standards at an early stage.
- Use dependency management systems such as Maven or Gradle for version control and library updates. This helps to avoid the use of outdated and vulnerable dependencies.

**5. Team Training and Development**

Team training and development play a key role in effective technical debt management. It is important to ensure continuous training and knowledge sharing between team members in order to maintain a high level of professionalism and competence.

**Organization of trainings and workshops:**

- Organize regular training and workshops for the team on development best practices, tools and testing methods. This helps the team stay up to date with new technologies and methods, as well as improve their skills.
- Invite experts and conduct workshops on relevant topics to enrich the team's knowledge and stimulate professional growth.

**Supporting the learning process and knowledge sharing:**

- Create conditions for the regular exchange of knowledge and experience within the team through joint discussions, code reviews and retrospectives. This helps identify and resolve problems early and improve code quality and development processes.

- Use internal resources, such as wikis, knowledge bases, or document management systems, to store and organize information. This makes it easy for the team to find and use the data and documents they need.

**Encouragement of self-education and professional development:**

- Encourage team members to educate themselves and develop their skills by providing access to training materials, courses, conferences, and workshops. This helps the team to keep abreast of new technologies and methods, as well as improve their professional skills.
- To support the participation of the team in professional communities and forums in order to exchange experience with colleagues from other companies and industries.

Technical debt management is an important aspect of the software development process. Effective technical debt management helps to improve product quality, minimize risks and optimize support and development costs. In this conclusion, we will summarize the aspects of technical debt management considered and emphasize their importance for the successful development of the project.

Identifying and assessing technical debt allows the development team to clearly understand where and why technical debt has accumulated, and how it affects the project. Regular code audits, the use of automated tools for analysis, and an assessment of the impact of problems on system performance, security, and stability help identify and document technical debt.

Prioritizing tasks to eliminate technical debt plays a key role in effective management. Classifying tasks by their importance and urgency, using priority matrices, and planning tasks allow you to focus your team's efforts on the most important issues and allocate resources efficiently. Integrating technical debt into sprints ensures that issues are fixed systematically and continuously. Including tasks to eliminate technical debt in the backlog, determining the time to complete them in each sprint, conducting code reviews and retrospectives help the team monitor the quality of the code and quickly respond to emerging problems.

The implementation of best development practices prevents the accumulation of technical debt in the future. The development and implementation of coding and documentation standards, the use of testing methods, the use of automated tools for static code analysis and dependency management contribute to maintaining high code quality and facilitate its maintenance and development.

Team training and development play a key role in successful technical debt management. Organizing training and workshops, supporting the learning process and sharing knowledge within the team, encouraging self-education and professional development help the team to keep abreast of new technologies and methods, improve their skills and maintain a high level of professionalism and competence.

Following these steps allows the development team to maintain the high quality of the product, prevent the accumulation of problems, and ensure the successful development of the project. Effective technical debt management not only improves code quality, but also contributes to a more productive and satisfying team, as well as increases customer and user trust and satisfaction.

## Quality metrics: how to measure success

In the world of software development, the quality of a product plays a crucial role in its success and user satisfaction. In order to measure and maintain a high level of quality, you need to use a variety of metrics to help measure success and identify areas for improvement. In this section, we'll look at key quality metrics, what they mean, and how to use them to measure project success.

**1. Code metrics**

Code metrics help you assess the quality of your source code and identify potential problems. They provide an understanding of the structural aspects of the code and its compliance with standards.

**Key metrics:**

- **Code Test Coverage:** The percentage of code that has been tested by tests. High test coverage indicates good code validation and reduces the risk of undetected bugs.
- **Cyclomatic Complexity:** Measures the complexity of logical paths in a program. High complexity may indicate that the code needs to be refactored and simplified.
- **Code duplication:** The percentage of code that is repeated in different parts of the project. Reducing duplication contributes to improved readability and maintainability of the code.

**2. Testing Metrics**

Test metrics allow you to evaluate the effectiveness of the testing process and the quality of test scenarios. They help determine how well the product has been tested and protected from defects.

**Key metrics:**

- **Test Pass Rate:** The percentage of tests that pass out of the total number of tests passed. A high pass rate indicates the stability and reliability of the product.

- **Number of defects found:** The number of defects identified during testing. This metric helps you evaluate the quality of your source code and the effectiveness of your testing.
- **Mean time to fix a defect:** The time it takes to identify and fix a defect. Reducing this time results in a faster release of the product.

**3. Performance Metrics**

Performance metrics help you evaluate how well a product performs under real-world operating conditions. They cover various aspects of system performance and load.

**Key metrics:**

- **Response time:** The average time taken to process a user's request. Optimizing response time increases user satisfaction.
- **Throughput:** The number of requests or operations that the system can process in a given period of time. Increased throughput can handle the increased load.
- **Resource utilization:** The level of utilization of various system resources (CPU, memory, disk). Optimizing resource utilization helps to avoid bottlenecks and improve system efficiency.

**4. Security metrics**

Security metrics assess the level of protection of a product from potential threats and vulnerabilities. They help identify and resolve security issues before they appear in the production environment.

**Key metrics:**

- **Number of vulnerabilities identified:** The number of vulnerabilities found during security testing. A decrease in this number indicates an increase in the security level of the product.
- **Time to remediate:** The average time it takes to identify and remediate a vulnerability. Reducing this time helps you respond quickly to security threats.
- **Security Compliance:** The extent to which the product complies with various security standards and regulations (e.g., GDPR, OWASP). Compliance with security standards ensures data protection and minimizes risks.

**5. User Satisfaction Metrics**

User satisfaction metrics allow you to assess how a product meets user expectations and needs. They help identify areas for improvement and increase satisfaction.

**Key metrics:**

- **Net Promoter Score (NPS):** A measure of users' willingness to recommend a product to others. A high NPS indicates high user satisfaction and loyalty.
- **User rating:** The average rating of a product received from users (e.g., on the App Store or Google Play). High ratings indicate a good quality of the product.
- **Number of Returns and Complaints:** The number of product returns and user complaints. A decrease in this number indicates an increase in quality and satisfaction.

Using code, testing, performance, security, and user satisfaction metrics helps you identify areas for improvement, optimize processes, and meet user requirements and expectations. Regular monitoring and analysis of metrics allows the development team to respond to emerging issues in a timely manner and make informed decisions to improve the quality of the product.

## Chapter 8. The future of testing in Agile

The agile methodology continues to evolve, and with it, approaches to software testing are evolving. In the quest for a faster, better, and more agile development process, new technologies and techniques are emerging that promise to change the future of testing. In this section, we will look at the key areas that will shape the future of testing in Agile: the development of automation using AI/ML, hybrid testing approaches, and continuous training and development of specialists.

Modern technologies such as artificial intelligence (AI) and machine learning (ML) are opening up new possibilities for test automation. These technologies are able to analyze large amounts of data, identify patterns and anomalies, and predict possible defects, making the testing process more efficient and accurate. The development of automation using AI/ML will be an important aspect of the future of testing in Agile.

Hybrid testing approaches that combine best practices from different methodologies will also play a key role. By combining automated and manual testing, as well as using a variety of techniques and tools, teams can achieve maximum flexibility and efficiency. Hybrid

approaches will allow you to adapt to changing requirements and development conditions, ensuring high product quality.

Continuous training and development of specialists remain important elements of successful testing in Agile. With rapidly changing testing technologies and methods, it is important that professionals are constantly improving their skills and knowledge. Learning and development will help teams be ready for the challenges of the future and effectively use new tools and approaches.

In the following subsections, we'll take a closer look at these key areas, their implications, and their impact on the future of testing in Agile.

## Advancing automation with AI/ML

Modern technologies such as artificial intelligence (AI) and machine learning (ML) are opening up new possibilities for software test automation. These technologies enable smarter and more adaptive systems that can analyze large amounts of data, identify patterns and anomalies, and predict possible defects. In this section, we'll look at how AI and ML can change the future of test automation and what benefits they can bring.

**1. Intelligent Test Creation**

AI and ML can be used to automatically create test scenarios and cases based on the analysis of the source code, requirements, and history of previous tests. This allows you to reduce the time for test development and provide more complete coverage.

*Example:*

- **Code-Based Test Generation:** Use AI to analyze source code and automatically create tests that check different branches and states of the program.
- **Requirements Analysis:** Use ML to analyze requirements and create test scenarios that verify that a product's functionality meets its stated requirements.
- **Interactive Test Cases:** Create interactive test cases that automatically adapt to changes in code and requirements, ensuring relevance and complete coverage.

**2. Anomaly detection and defect prediction**

AI and ML can analyze test results and identify anomalies that may indicate the presence of defects. In addition, these technologies can predict possible problems based on the analysis of historical data and patterns.

*Example:*

- **Log and metrics analysis:** Use ML to analyze logs and test execution metrics to identify anomalies, such as performance anomalies or system instability.
- **Defect prediction:** Applying AI to predict the sections of code that are most likely to contain defects, based on an analysis of past changes and test results.
- **Pattern Analysis:** Automatically identify patterns associated with defects and generate recommendations for their elimination.

**3. Automatic defect correction**

AI and ML can not only identify defects, but also offer ways to fix them automatically. This allows you to speed up the troubleshooting process and improve the quality of your code.

*Example:*

- **Automatic code refactoring:** Use AI to analyze code and propose changes to eliminate identified defects and improve readability and performance.
- **Self-correcting tests:** Use ML to automatically update and adjust tests in case of changes in code or requirements.
- **Automatic error correction:** AI can generate patches to fix detected errors and automatically apply them to the code.

**4. Test Optimization**

AI and ML can help streamline the testing process by prioritizing tests and allocating resources based on data analysis.

*Example:*

- **Test prioritization:** Using AI to analyze functionality criticality and defect probability to determine which tests should be performed first.
- **Optimize Resource Utilization:** Apply ML to allocate testing resources based on system load and tester availability to maximize the efficiency of the testing process.
- **Dynamic Scheduling:** Automatically schedule tests and resources based on current priorities and system health.

**5. Enhanced security**

AI and ML can be used to improve the security posture of a product by identifying potential vulnerabilities and predicting possible threats.

*Example:*

- **Code Security Analysis:** Using AI to analyze source code for vulnerabilities such as SQL injection or XSS attacks.
- **Threat Monitoring and Prediction:** Apply ML to monitor the system and predict possible threats based on the analysis of behavior patterns and events.
- **Adaptive Protection:** AI can dynamically adapt security measures based on identified threats and the context of the system's operation.

**Benefits of Using AI/ML in Test Automation**

**1. Improve accuracy and efficiency**

AI and ML can significantly improve the accuracy and efficiency of testing by analyzing large amounts of data and identifying hidden defects and anomalies. These technologies allow you to identify problems faster and more accurately and offer solutions, which reduces testing costs and improves product quality.

**2. Reduce testing costs and time**

Test automation using AI/ML allows you to reduce the time for developing and executing tests, as well as reduce the cost of their support. This is especially important in the context of rapid releases and increasing requirements for product quality.

**3. Adaptability and scalability**

AI and ML enable testing adaptability, allowing systems to automatically adapt to code changes and requirements. These technologies also allow you to scale your testing processes to validate different aspects of your product across different platforms and environments.

The development of automation using AI/ML opens up new horizons for software testing. These technologies enable smarter and more adaptive systems that can automatically generate tests, identify and correct defects, streamline the testing process, and improve safety. Implementing AI and ML in test automation will help development teams achieve high levels of quality and efficiency, cope with the increasing complexity and volume of projects, and meet the growing demands of users and customers.

## Hybrid testing approaches

Hybrid testing approaches are a combination of different methods and techniques, combining the benefits of automated and manual testing. These approaches allow development teams to adapt to changing conditions and requirements, ensuring flexibility and high product quality. In this section, we'll look at the basic principles and strategies of hybrid testing approaches, their benefits, and use cases.

**1. Principles of hybrid testing approaches**

Hybrid approaches are based on several key principles that help development teams effectively combine different testing methods and achieve the best results.

**Combination of methods:**

- Hybrid approaches involve the use of both automated and manual testing. This allows you to use the strengths of each method and compensate for their shortcomings.
- The inclusion of various testing techniques such as functional, regression, load, security, and cross-browser testing provides comprehensive coverage of all aspects of the product.

**Adaptability and flexibility:**

- Hybrid approaches allow teams to adapt to changing development requirements and conditions. This includes the ability to change test priorities, add new tests, and modify existing scenarios.
- Using agile methodologies such as Agile and DevOps allows you to integrate hybrid approaches into your development and quality assurance processes.

**Cooperation and interaction:**

- Hybrid approaches require close collaboration between testers, developers, analysts, and product owners. This helps to ensure consistency and transparency in the team.
- Regular discussion and evaluation of test results, as well as the exchange of knowledge and experience, contribute to improving processes and product quality.

**2. Hybrid Testing Strategies**

Hybrid approaches include a variety of strategies that help development teams effectively combine testing methods and techniques. These strategies may include the following elements:

**Integration of automated and manual testing:**

- Use automated testing to validate routine and repetitive tasks such as regression testing and API functionality validation.
- Apply manual testing to test complex and non-standard scenarios, as well as to evaluate the user interface and system interaction.
- Regularly update and maintain automated tests to ensure they are up-to-date and up-to-date.

**Combination of different testing techniques:**

- Enable functional testing to verify that individual features and components are working correctly.
- Use load testing to evaluate system performance under different load conditions.
- Apply security testing to identify vulnerabilities and ensure data protection.
- Conducting cross-browser testing to check the compatibility of the product on different platforms and devices.

**Adaptive planning and prioritization:**

- Prioritization of testing based on risk analysis and functionality criticality. This helps you focus on the most important aspects of the product and optimize the use of resources.
- Adaptive test planning based on current priorities and project status. This includes the ability to change plans and add new tests in response to changing requirements and conditions.

**Use of modern tools and technologies:**

- Implementation of test automation tools such as Selenium, Appium, JUnit, TestNG, and others. These tools help automate various aspects of testing and improve the efficiency of the process.
- Apply artificial intelligence (AI) and machine learning (ML) technologies to intelligently create tests, detect anomalies, and optimize testing.
- Use test management systems such as Jira, TestRail, and others to organize and control testing processes.

**3. Benefits of hybrid testing approaches**

Hybrid testing approaches have a number of advantages that make them especially useful in today's software development.

**Increased efficiency:**

- Combining automated and manual testing allows teams to use their resources more efficiently and reduce the time to review the product.
- Automating routine tasks frees up testers to perform more complex and creative tasks, which improves the overall productivity of the team.

**Quality improvement:**

- Hybrid approaches provide more comprehensive test coverage of all aspects of the product, which helps to identify and eliminate defects at an early stage.
- Regular updates and support of tests help to keep the requirements up-to-date and compliant, which contributes to the quality of the product.

**Flexibility and adaptability:**

- Hybrid approaches allow teams to quickly adapt to changes in development requirements and environments, which is especially important in Agile and DevOps environments.
- The ability to change test plans and priorities helps teams respond to emerging issues in a timely manner and optimize processes.

**Risk mitigation:**

- Comprehensive coverage of all aspects of the product and regular testing help identify potential problems and minimize risks related to system quality and stability.
- The use of various testing techniques provides a more complete understanding of the state of the product and allows you to make informed decisions on its development and release.

**4. Hybrid Testing Use Cases**

Let's look at a few examples that illustrate the use of hybrid testing approaches in different scenarios.

***Example 1: E-commerce web application***

- **Functional testing:** Automated testing of key features such as user registration, adding items to cart, and checkout.
- **Regression testing:** Automating regression testing to verify that new changes have not disrupted existing functionality.
- **Manual testing:** Manual testing of the user interface and system interaction to evaluate usability and identify visual defects.
- **Load testing:** Conducting load testing to evaluate the performance of the system under high traffic and identify bottlenecks.

***Example 2: Mobile application for a bank***

- **Functional testing:** Automate the testing of basic features such as logging into an account, transferring money, and paying bills.

- **Cross-platform testing:** Checking the operation of the application on various platforms (iOS, Android) and devices using automated tests.
- **Security testing:** Conducting security testing to identify vulnerabilities related to the processing and storage of financial data.
- **Manual testing:** Manual testing of complex scenarios and UI to assess usability and identify issues that may be missed by automated tests.

***Example 3: Enterprise Project Management System***

- **Functional testing:** Automate the testing of key features such as creating and managing projects, assigning tasks, and tracking progress.
- **Integration testing:** Checking the interaction of various modules of the system using automated tests.
- **Performance Testing:** Perform performance tests to evaluate the performance of a system with a large amount of data and users.
- **Manual testing:** Manual testing of complex scenarios and user interactions with the system to identify defects and improve usability.

Hybrid testing approaches are an effective combination of different methods and techniques that allow development teams to adapt to changing conditions and requirements, ensuring high product quality. These approaches include the integration of automated and manual testing, the combination of different testing techniques, adaptive planning and prioritization, the use of modern tools and technologies, and close collaboration and collaboration within the team. Examples of hybrid approaches in different scenarios demonstrate their effectiveness and flexibility, enabling teams to achieve the best results in today's software development.

## Continuous training and development of specialists

In the fast-paced world of technology and methodologies, it's important for test and development professionals to continually improve their skills and knowledge. This helps teams to be ready for new challenges, effectively use modern tools and methods, and maintain a high level of product quality. In this section, we will look at the main aspects of continuous training and development of specialists and offer strategies for their implementation.

**1. On-the-job training**

One of the most effective ways of training is on-the-job training, which allows specialists to acquire new knowledge and skills in the process of performing their duties.

**Possible methods:**

- **Mentoring and mentoring:** Creating a mentoring system in which experienced specialists help newcomers master new technologies and methods. This contributes to faster and better training.
- **Periodic code reviews:** Regular code reviews allow specialists to share knowledge and experience, identify bugs, and learn about development best practices.
- **Involvement in project work:** Involving specialists in a variety of tasks and projects allows them to develop a wide range of skills and knowledge.

**2. Formal Curricula**

Formal curricula play an important role in the systematic training and development of professionals. They provide structured and in-depth knowledge on various aspects of testing and development.

**Possible methods:**

- **Courses and training:** Organizing courses and training on various topics, such as test automation, security testing, load testing, and others. This allows specialists to deepen their knowledge and master new tools and technologies.
- **Certifications:** Support professionals in obtaining professional certifications such as ISTQB, CSM, and others. Certifications validate the qualifications of professionals and increase their competitiveness in the labor market.
- **Conferences and seminars:** Participation in professional conferences and seminars allows specialists to learn about the latest trends and achievements in the field of testing and development, as well as to exchange experience with colleagues.

**3. Self-education and self-study**

Self-education and self-study play an important role in maintaining a high level of professionalism and competence of specialists. They allow them to stay up-to-date with the latest technologies and techniques.

**Possible methods:**

- **Online Courses and Webinars:** Using online courses and webinars for self-paced learning. Platforms like Coursera, Udemy, LinkedIn Learning, and others offer a wide variety of courses on a variety of topics.
- **Reading professional literature:** Regularly reading professional books, articles, and blogs helps professionals stay up-to-date with the latest research and practices.
- **Participation in professional communities:** Joining professional communities and forums such as Stack Overflow, Reddit, LinkedIn groups, and others allows professionals to share knowledge and experience with peers.

**4. Feedback and assessment of development**

Feedback and assessment of development play an important role in the process of training and development of specialists. They help to identify strengths and weaknesses, set goals and plan further development.

**Possible methods:**

- **Regular meetings and discussions:** Hold regular meetings with management and colleagues to discuss progress, identify problems, and plan for further development.
- **Goal setting and planning:** Setting short-term and long-term goals for each professional, and developing a plan to achieve them. This helps professionals focus on the most important aspects of their development.
- **Evaluation of results and achievements:** Regular evaluation of the results and achievements of specialists using various metrics and indicators. This allows you to track progress and adjust training and development plans.

**5. Learning and development support from the company**

The company plays an important role in supporting the training and development of its employees. It can create conditions and provide resources for training, as well as stimulate employees to grow professionally.

**Possible measures:**

- **Establishment of training centers:** Establishment of internal training centers and laboratories where specialists can be trained and practice in the use of new tools and technologies.
- **Providing access to training materials:** Providing professionals with access to training materials such as books, articles, online courses, and other resources.
- **Financial support for training:** Payment for courses, trainings, certifications and participation in conferences. This stimulates specialists to learn and increases their motivation.

- **Promotion of professional growth:** Introduction of incentive and reward programs for achievements in the field of learning and development, such as bonuses, bonuses and recognition.

On-the-job training, formal training programs, self-education and self-study, feedback and evaluation of development, as well as support from the company help specialists maintain a high level of professionalism and competence. This allows development teams to be ready for new challenges, effectively use modern tools and methods, and maintain a high level of product quality. The implementation of continuous learning and development strategies contributes to the successful development of the project and the satisfaction of customers and users.

## Chapter 9. Conclusion

In this book, we have looked at the key aspects of software testing in an Agile environment. Testing plays a crucial role in ensuring product quality and meeting customer requirements. With the rapid development of technologies and methodologies, it is

important to constantly improve testing approaches, introduce modern tools and practices, and maintain a high level of professionalism and competence of the team.

We discussed various testing methods, including manual and automated testing, their advantages and disadvantages, and recommendations for use. We also looked at the importance of sprint-level test planning, technical debt management, maintaining product quality in a rapid release environment, and implementing testing in Definition of Done (DoD).

The future of testing in Agile is associated with the development of automation using AI/ML, hybrid approaches to testing, and continuous training and development of specialists. Following these best practices and strategies will help your team achieve high results, create high-quality and reliable products, and meet the growing demands of users and customers.

## Summary of key ideas

1. **The role of testing in Agile:**
    - Testing is an integral part of the development process, ensuring high product quality and compliance.
    - Integration of testing at all stages of development and close interaction of the team contribute to the successful implementation of the project.
2. **Manual and automated testing:**
    - Manual testing provides flexibility and the ability to identify complex and non-standard defects.
    - Automated testing allows you to speed up the process, increase the accuracy and repeatability of tests.
    - The combined use of manual and automated testing provides the best results.
3. **Sprint-level test planning:**
    - Active participation of testers in sprint planning and the development of detailed test plans ensure that all tasks and user stories are fully covered by tests.
    - Integrating automated tests into the CI/CD process speeds up test execution and increases test efficiency.
4. **Technical debt management:**

- Identifying, assessing, and prioritizing technical debt tasks help maintain code quality and prevent problems from accumulating.
- Implementing best development practices and training the team contributes to successful technical debt management.

5. **Maintaining product quality in the conditions of rapid release:**
   - Integration of test automation, risk management, and continuous process improvement help maintain high product quality.
   - Use quality metrics to measure success and identify areas for improvement.
6. **The future of testing in Agile:**
   - The development of automation using AI/ML opens up new horizons for software testing.
   - Hybrid testing approaches provide flexibility and efficiency by combining different methods and techniques.
   - Continuous training and development of specialists helps teams to be ready for new challenges and effectively use modern tools and methods.

These key insights will help you understand the importance and basic aspects of testing in an agile environment, as well as prepare for future challenges and opportunities in software development. Following these guidelines will allow your team to achieve high results and create high-quality and reliable products.

## The importance of a quality culture in an Agile team

A culture of quality is a fundamental component of a successful Agile team. It defines how the team perceives and approaches product quality assurance at all stages of development. In an agile environment where change is rapid and continuous, a culture of quality helps teams maintain high standards and meet customer and user expectations.

**1. Principles of quality culture**

**Cooperation and mutual understanding:**

- In an agile team, all development participants, including developers, testers, analysts, and product owners, must actively communicate and collaborate. This ensures consistency and a common understanding of goals and objectives.
- Regular meetings, such as daily stand-ups, retrospectives, and sprint planning, facilitate the sharing of knowledge and experience, which helps improve processes and product quality.

- Transparent and open communication between team members helps to avoid misunderstandings and conflicts, and contributes to more effective problem solving.

**Continuous improvement:**

- The culture of quality in an agile team involves a constant desire to improve processes and products. This includes regularly analyzing the results, identifying problems, and taking action to fix them.
- Using approaches like Kaizen and Lean helps teams focus on continuous improvement and increasing efficiency.
- Regular retrospectives allow the team to evaluate the work done, identify areas for improvement, and develop plans to implement changes.

**Responsibility and involvement:**

- In an Agile team, each team member must feel responsible for the quality of the product. This includes performing their duties at a high level and actively participating in testing and improvement processes.
- The team's involvement in development and testing helps to identify and eliminate defects at an early stage, which reduces risk and improves quality.
- Creating an atmosphere of trust and support helps the team work smoothly and effectively, which contributes to achieving high results.

**Innovation and adaptability:**

- The culture of quality in an agile team encourages innovation and experimentation. This allows us to find new solutions and methods to improve the quality and efficiency of development.
- The team's adaptability to changes and new requirements helps to respond quickly to challenges and maintain a high level of product quality.
- The introduction of modern technologies and tools, such as test automation, AI, and ML, helps to improve processes and increase the quality of the product.

**2. Tools and practices to support a culture of quality**

**Test Automation:**

- Implementing automated testing helps teams validate product functionality quickly and efficiently. This reduces the risk of errors and defects, and speeds up the testing process.
- The use of tools such as Selenium, Appium, JUnit, and TestNG allows you to automate various aspects of testing and increase its accuracy and repeatability.
- Automating regression testing and integrating tests into the CI/CD process ensures continuous quality assurance and rapid problem identification.

**Code Review and Pair Programming:**

- Conducting regular code reviews helps to identify errors and shortcomings, as well as share knowledge and best practices within the team.
- Pair programming, in which two developers work together on the same computer, contributes to better and more stable code, as well as better collaboration and mutual understanding.
- Involving all team members in the review and pair programming processes helps to create uniform coding standards and improve code quality.

**Integration of testing into CI/CD:**

- Integrating testing into the continuous integration and delivery (CI/CD) process helps teams automate test execution and code review with every change. This ensures continuous quality control and quick identification of problems.
- Using tools such as Jenkins, GitLab CI, and CircleCI allows you to automate the build, test, and deployment processes.
- Setting up alerting and monitoring helps teams respond quickly to emerging issues and maintain system stability.

**Metrics and monitoring:**

- Using quality metrics helps teams measure success and identify areas for improvement. Metrics such as code coverage, test pass rate, average time to fix defects, and others help you track progress and make informed decisions.
- Setting up monitoring and alerting allows teams to quickly respond to emerging problems and maintain system stability.
- Regular analysis of metrics and reports helps the team identify trends and develop strategies to improve processes and product quality.

**3. The Role of the Leader in Creating a Culture of Quality**

**Example to follow:**

- Agile team leaders must demonstrate high quality standards in their work and serve as an example for the rest of the team. This includes performing their duties at a high level and actively participating in testing and improvement processes.
- The example of a leader helps to create an atmosphere of responsibility and involvement, which contributes to maintaining a culture of quality in the team.
- Leaders should actively support and encourage the team's initiatives to improve processes and product quality.

**Support and motivation:**

- Leaders must support and motivate team members to continuously learn and develop. This includes organizing training, providing access to training materials, and encouraging participation in professional communities.
- Motivating the team to achieve high quality standards helps create a culture focused on continuous improvement and innovation.
- Leaders should create conditions for the professional growth and career development of team members by providing them with opportunities to learn and participate in interesting projects.

**Creating the conditions for success:**

- Leaders must create the conditions for the success of the team by ensuring access to the necessary tools and resources, as well as maintaining flexibility and adaptability of processes.
- Creating a favorable working atmosphere and encouraging cooperation help the team achieve high results and maintain a high level of product quality.
- Leaders must actively participate in solving emerging problems and support the team in difficult situations to ensure the stability and continuity of the development process.

The quality culture in an agile team plays a key role in ensuring a high level of product quality and meeting customer requirements. It is based on the principles of cooperation, continuous improvement, responsibility and involvement, innovation and adaptability. Using tools and practices such as test automation, code reviews, CI/CD test integration, and quality metrics helps maintain a culture of quality throughout all stages of development. The role of a leader in creating and maintaining a culture of quality is also critical, as leaders lead by example for the team, support and motivate team members, and create the conditions for their success.

Maintaining a quality culture in an Agile team contributes not only to the creation of a reliable and high-quality product, but also improves the atmosphere and workflows in the

team. This allows us to achieve high results, meet the requirements of customers and users, as well as contribute to the professional growth and development of all team members.

## Application

### List of Test Automation Tools

- Selenium: A popular tool for cross-platform testing of web applications.

- TestRigor: Supports desktop, mobile, and API testing.
- TestRail: A test management tool that supports integration with JIRA and other bug tracking systems.
- Zephyr Enterprise: A test management solution that includes the creation of test cases and reports.
- TestComplete: A functional testing tool that supports testing desktop, web, and mobile applications.
- Katalon Studio: Supports test automation for web apps, mobile apps, and APIs.
- UFT (Unified Functional Testing): A tool for testing desktop and web applications, supports integration with bug tracking systems.
- Watir (Web Application Testing in Ruby): A tool for testing web applications based on the Ruby programming language.
- Appium: An open tool for testing mobile applications, supports various platforms and programming languages.
- Cucumber: A behavior-based testing tool that supports BDD (Behavior-Driven Development).

## Useful links and resources for learning

- Agile Testing: The Role and Strategies of Quality Assurance - Kurshub
- Agile testing best practices and why they matter - Atlassian
- Introduction to Agile Methodology and Its Impact on Testing - Habr
- Agile Testing: A Practical Guide for Testers and Agile Teams - Lisa Crispin, Janet Gregory Amazon
- Agile Testing: How to Succeed in an Ever-Changing World - Lisa Crispin, Janet Gregory Amazon
- Agile Testing with Cucumber - Richard Lawrence, Paul Rayner Amazon
- Continuous Testing for DevOps Professionals: A Practical Guide from Industry Experts - Eran Kinsbruner, Antoine Craske, Stephen Feloney Amazon
- The Agile Tester: Foundations of Software Testing - Dorothy Graham, Rex Black Amazon
- Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing - Elisabeth Hendrickson Amazon
- More Agile Testing: Learning Journeys for the Whole Team - Janet Gregory, Lisa Crispin Amazon

### Templates for Agile Test Planning

- **TestRail**: Provides flexible templates for test planning, integrates with CI/CD pipelines.
- **Zephyr Enterprise**: Supports the creation and management of test plans and reports.

- **Smartsheet:** Offers a wide range of templates for agile project management, including templates for testing.
- **Tipsographic:** Agile Test Plan Template for Excel that can be downloaded for free.
- **Agile Seeds:** Samples and best practices of Agile test plan templates.
- **SlideTeam:** Top 7 Agile Test Plan Templates with Examples and Samples.
- **Atlassian Jira:** Agile project management platform with test planning tools.
- **TestLodge:** A template management tool for Agile.
- **TestLink:** A test management system with the ability to create templates for Agile.
- **Microsoft Excel:** You can create your own templates for Agile test planning using Excel spreadsheets.