Last Updated: 12 Feb, 2021

11 Ways to Improve Software Testing through Planning, Work Environment, Automated Testing, and Reporting

Reading time: 21 minutes

A lot of research has been done to identify the root causes of software startup failures. One of the main reasons for such failures turned out to be poor quality assurance during the software development process. The chief purpose of executing stringent quality assurance tests is to prevent the release of poor quality products. Small mistakes that slip through may lead to large financial losses.

A good example of QA's importance is Flud, a social news reader application for iPad, iPhone, Android, and Windows Phone. Flud was known as the "first true social newsreader." But the startup failed because of poor QA services. The main priority of the Flud team was the development process and code creation — almost to the exclusion of everything else. When the product was finally released, it was bug-infested and load with mismatches. Even though everything was fixed, the bad reputation and awful user experience prevented its success. So, Flud was discontinued.

The way to create high-quality software is to implement effective QA management that provides tools and methodologies for building bug-free products. Software quality management is an umbrella term covering three core aspects: quality assurance, quality control, and testing.

Watch our video explaining the main concepts of software testing

Software quality assurance (SQA) is the part of quality management that includes a planned set of organizational actions. The purpose of these actions is to improve the software development process, introducing standards of quality for preventing errors and bugs in the product.

Software quality control (SQC) is the part of quality management that includes a set of activities focused on fulfilling quality requirements. QC is about product-oriented activities that certify software products for their quality before release. The process of software quality control is governed by software quality assurance.

Testing is the basic activity aimed at detecting and solving technical issues in the software source code and assessing the overall product usability, performance, security, and compatibility. It's not only the main part of quality assurance; it is also an integral part of the software development process.

Agile software development cycle

This article will discuss the best practices of how to improve the software testing process and to increase the quality of your software products.

1. Plan the testing and QA processes

Test processes should be well-planned, defined, and documented. Good documentation is the tool that builds efficient communication within the software team. So, effective planning entails creation of quality and test plans for a project. Let's take a look at the main types of documentation that support the QA process.

A quality assurance planning scheme

Test policy

A test policy is the most high-level document that is created at the organizational level. It defines the test principles adopted by the company and the company's main test objectives. It also explains how testing will be conducted and how a company measures test effectiveness and success.

There is no standard approach to test policy creation, but it typically includes the following:

- Definition of what testing means for the company,
- Test objectives of the organization,
- General standards and criteria for software testing in projects,
- Definition of testing terms to clarify their further usage in other documents,
- Listing of tools to support the testing process,

- Methods and metrics for assessing efficiency of testing, and
- Ways to improve the testing processes.

Quality management plan

A quality management plan is a document that defines an acceptable level of product quality and describes how the project will achieve this level. It's not a mandatory document, but it will help you schedule all the tasks needed to make sure that the project meets your customer's needs and expectations. The main goal of this plan is to support project managers and help organize the process by defining roles, responsibilities, and quality standards to be achieved. Accordingly, it should include the software's quality requirements and describe how they should be assessed.

Key components of the quality management plan:

- Quality objectives
- Key project deliverables and processes to be reviewed for satisfactory quality level
- Quality standards
- Quality control and assurance activities
- Quality roles and responsibilities
- Quality tools
- Plan for reporting quality control and assurance problems

Test strategy

A test strategy is a more specific product-level document that derives from the Business Requirements Specification document. Usually, a project manager or a business analyst creates a test strategy to define software testing approaches used to achieve testing objectives. A test strategy is driven by the project's business requirements, which is why it meshes with a project manager's responsibilities.

The main components of a test strategy are:

- The scope of testing
- Test objectives
- Budget limitations
- Communication and status reporting
- Industry standards
- Testing measurement and metrics
- Defect reporting and tracking
- Configuration management
- Deadlines
- Test execution schedule
- Risk identification

In a small project, the test strategy is part of a test plan. But, for a larger project, the PM has to create a test strategy as a separate, static document from which each test plan can be further developed.

A good test strategy document answers the following questions:

- What is the product?
- What part(s) of it should be tested?
- How should they be tested?
- When should testing begin?
- What are the start/end criteria?

Test Plan

A test plan is a document that describes what to test, when to test, how to test, and who will do the tests. It also describes the testing scope and activities. The test plan includes the objectives of the tests to be run and helps control the risks. It's a good practice to have a test plan written by an experienced person like a QA lead or manager.

A good test plan should include the schedule for all necessary testing activities in order to control your team's testing time. It also should define the roles of every team member so that everyone is clear about what is required. There's no universal way of creating a test plan because it depends on the processes, standards, and test management tools implemented in the company. According to the IEEE standard 829, a test plan document should contain the following information:

- Test plan identifier
- Introduction
- References (list of related documents)
- Test items (the product and its versions)
- Software risk issues
- Features to be tested
- Features not to be tested
- Approach (Strategy)
- Item pass or fail criteria
- Suspension criteria
- Deliverables (test plan document, test cases, tools, error logs, problem reports, etc.)
- Test environment (hardware, software, tools)
- Schedule

- Staffing and training needs
- Responsibilities
- Risks
- Approvals

Here are some key guidelines for making the test plan more effective:

Make your test plan brief. Avoid repetition or irrelevance. It should contain only the relevant information.

Be specific. Include all details, e.g. editions and versions of the programs, to make the document searchable.

Update a test plan. It's a live document that must be frequently updated on an on-demand basis.

Share a test plan with your stakeholders. It will give them information about your testing processes. The quality of your test plan will represent the quality of the testing your team will perform.

Test cases

Preparation of effective test cases is an integral part of software testing improvements. According to the definition, given by ISTQB (International Software Testing Qualifications Board, the worldwide leader in the certification of competencies in software testing), "A test case is a set of input values, execution preconditions, expected results, and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement." It's one of the key instruments used by testers. The standard test case includes the following information:

- The test case ID
- Test case description
- Prerequisites

- Test steps
- Test data
- Expected result
- Actual result
- Status
- Created by
- Date of creation
- Executed by
- Date of execution

Below you can see the example of a standard test case.

Example of standard test case

Source: ISTQB certification

Use the following practices to write effective test cases:

Identify testable requirements. Identify the scope and purpose of testing before starting the test process.

Customer requirement. The specialist who writes the test case must have a good understanding of the features and user requirements. Each test case should be written keeping the client's requirements in mind.

Write on time. The best time to write test cases is the early requirement analysis and design phases. That way QA specialists can understand whether all requirements are testable or not.

Simple and clear. Test cases should be simple and easy to understand. Every test case should include only the necessary and relevant steps. N matter how many times and by whom it will be used, a test case must have a single expected result rather than multiple expected results.

Unique test cases. Each test case must have a unique name. This will help classify, track, and review test cases at later stages.

Test cases should be maintainable. If requirements change, a tester must be able to adjust a test case.

2. Employ test-oriented software development management

Implementation of test-oriented management approaches is a good way to improve the quality of software. One of the ways to achieve this is by using extreme programming (XP) – a software development methodology that aims to produce higher quality software with the ability to adapt to changing requirements. Check this article to learn more about extreme programming practices, principles, and values. Here, we'd like to focus on the two XP practices that are closely related to testing::

- Test-driven development
- Pair programming

Test-driven development

Test-driven development (TDD) is a software development process in which tests are written before any implementation of the code. TDD has a test-first approach based on the repetition of a very short development cycle. According to it, each new feature begins with writing a test. The developer writes an automated test case before he/she writes enough production code to fulfill that test. This test case will initially fail. The next step will be to write the code focusing on functionality to get that test passed. After these steps are completed, a developer refactors the code to pass all the tests.

Test-Driven Development lifecycle

The following are the benefits of using the TDD approach:

High quality. The quality of TDD-based products is usually much higher than that achieved with other methods.

Optimization of development costs. The cost of debugging at later stages is minimized as tests are run from the beginning of the design cycle.

Simplification of code. Engineers invest more effort in aligning code requirements to particular tests.

Positive effects on productivity. The TDD approach provides quick feedback on introducing a bug and fixing it. A developer notices a bug as soon as the test fails and then fixes it to pass the test.

Executable documentation. Use-cases are written as tests and other developers can view the tests as examples of how the code is supposed t work.

Pair programming

Pair programming is also an extreme programming technique. This development approach requires two engineers working in tandem at a single computer. One of them (the Driver) writes a code while the other one (the Navigator) watches and makes suggestions throughout the process. These roles can be swapped at any time. Two developers working at a single computer will produce software with a significantly higher quality. The increased code quality can reduce the debugging and refactoring cost of the project in the long run.

The benefits of pair programming:

High code quality. Fewer mistakes and bugs are introduced into the code as problems are caught before or during the code writing.

Pair programming reduces the number of mistakes

Better knowledge sharing among team members. You will have more people who know how the product works. In this case, if one of the pairs leaves the company, there will be someone remaining who is experienced with the code.

Clear code. You will receive a shorter and much clearer code.

This practice can also be applied to the testing process. The pair testing technique combines the knowledge and experience of two testers in a kind of brainstorming session that can lead to increased productivity.

3. Use a shift-left approach to start testing early and often

We have already described a test-driven programming practice inside an extreme programming framework. A **shift-left testing approach** reflects this idea and suggests conducting testing activities from the very beginning of the development process instead of making it a final step as traditional methodologies typically suggest.

Planning a testing strategy early. Pushing testing procedures off until the last week can create bottlenecks and slow down progress. So, consider planning a testing schedule from the early stages of the development process to detect and fix bugs and malfunctions as soon as possible.

Reviewing requirements. The shift-left concept doesn't necessarily move all testing activities closer to the beginning of the cycle. It can mean engaging testers in communication with customers and other stakeholders with the purpose of reviewing and analyzing requirements.

Testing often. Doing smaller tests more frequently throughout the development stages and creating a continuous feedback flow allows for immediate validation and improvement of the system.

Automating tests. Implementing automated tests whenever possible and maximizing test coverage would also expedite and improve the testing process. Read on to learn more about test automation and continuous delivery practice.

Prevention instead of reaction. Shifting left can also focus on problem prevention rather than fixing. For example, testers can pair with developers and contribute to the coding process or run tests before hitting the build. Or, testers can join discussion sessions, ask questions, and provide rapid feedback to influence development decisions.

Team collaboration. This Agile approach works best in cross-functional teams where members tightly collaborate and have broad skillsets as testers are involved in the development process and developers — in testing activities, creating a product with testability in mind.

That being said, it's important to remember that a **shift-right strategy** also exists and implies post-production testing of a completely built product. It involves getting feedback from real end-users and improving the quality based on these reviews.

4. Conduct formal technical reviews

A formal technical review (FTR) is an activity performed by software engineers to reveal functional and logical errors at the early stages. An FTR is a group meeting at which attendants with certain roles ensure that developed software meets the predefined standards and requirements.

The best time to run an FTR is when you have a mature product. But it depends on the type of review. A typical FTR requires a team of engineers specific roles as speakers, reviewers, or producers. At the end of each meeting, a review report should be prepared to answer the following ques

- What was reviewed?
- Who reviewed it?
- What findings and decisions were made?

The FTR represents a class of reviews, which includes the following types:

A formal review or **review meeting** is a presentation given by the author of a product. The main objective is to introduce the product to the rest of the reviewers. As a result, all the participants have to accept the product, suggest modifications, and discuss timeframes.

A walkthrough is a meeting during which reviewers examine the source code of the product referred to, its design, and documented requirements. A walkthrough meeting is held to detect bugs in the code. The author of the code is often present to answer questions.

An inspection is a review session that determines the additional properties of the product according to the requirements. While formal reviews and walkthroughs are used for bug detection, inspections are held to expand initial standards, or check to see if previous bugs are still present.

Conducting formal, technical reviews helps to prevent errors and reduce the risk of logical and implementation errors in advance. It also helps a production team observe the whole product's features, making development more manageable.

5. Ensure suitable work environment for QA team

Work environment directly impacts employees' productivity and attitude toward their job. Here are some ways to create comfortable work conditions and keep your team happy, engaged, and productive.

Define QA roles

Testing consists of various activities that are performed by different specialists. To organize a smoothly-run testing process, roles are specified at the planning stage in a test plan. There are six common roles in QA:

- Software test engineer

- Test analyst
- Test automation engineer
- Software development engineer in test
- Test architect
- Test manager

Each role has its own set of skills, responsibilities, and tools to operate with. To set a proper interaction with your QA team and understand specifics of each position, learn more about QA roles and their features.

Software testing roles and their core functions

Respect your testers

If you want to achieve high-level quality goals, you need to build trusting relationships between a QA team and developers with respect for each other. Also, it would be better to search for people with coding skills. Obviously, engineers will respect such testers more. They will also be able to code some of their own testing tools.

A QA lead has to recognize the progress of the team and individual achievements of its members at team meetings. It will encourage other specialists to do better work in the future.

Give business training to your QA team

Provide the necessary pieces of training for your QA specialists to expand their knowledge. You can organize internal and/or external training sessions and team-building exercises to improve the work of the entire team. A QA team lead should organize brainstorming sessions to create floods of collective creativity in the team. It will help invent new techniques for solving an existing problem.

Encourage communication

Collocate your testers and developers to improve communication efficiency. Face-to-face communication will help avoid misunderstandings and share effective solutions to problems encountered during tests. You also need a good team leader who will be able to effectively share feedback and ideas with testers. QA managers should encourage team members to speak about existing problems and other issues with the team that could impact productivity and efficiency. Retrospective meetings held by the entire development team at the end of each sprint or iteration are one of the ways to discuss achievements, problems, and plans for further work.

It's also important to give your testers a chance to talk about things privately, outside of group meetings. QA leaders should be flexible and open to new strategies to best serve their teams.

6. Implement user acceptance testing

In product development, we have user personas to identify a perfect customer or a typical user for your product. A user persona is a fictional character that has the behavior patterns and goals of your product's target audience. QA teams use personas to identify where and how to seek a bug. But, following persona guidance can't predict the entire spectrum of behavior patterns. To ensure that your application meets user needs, consider engaging end-users in testing.

End-user testing or **user acceptance testing** traditionally comes at the final stages of software development. Engaging end-users to test your application can help discover bugs that might not normally be found. It also proves that your software is production-ready and supplies your developers with user feedback during/after the production stage.

Types of user acceptance testing

User acceptance testing (UAT) can be done in various ways. According to Usersnap, there are 5 UAT types:

Alpha and beta testing are done at the pre-release stage. Alpha testing is carried out by internal stakeholders at the early stages of development. Business and end-users are often involved in alpha testing performed in the development environment. The feedback from internal teams is used to further improve the quality of the product and fix bugs. Beta testing is performed in the customer's environment to determine if an app is ready f
users.

Contract Acceptance Testing is a type of UAT done to check if developed software meets the contract requirements.

Regulation Acceptance Testing ensures that software complies with legal regulations.

Operational Acceptance or **Production Readiness Testing** is done to check if an app is ready for production and usage. It verifies if there is a proper workflow arranged (user training, backup plans, security checks, etc.).

Black Box Testing examines software functionality without seeing the internal code. That means testers are only aware of what an app should do without knowing how. This type of testing allows test teams to get the most relevant results comparable with end-user testing.

How to organize user acceptance testing?

User acceptance testing helps to identify problems missed during unit and integration tests. Considering the importance of end-user insight, check the following tips to organize UAT properly:

Find interested users. Getting just any user into testing is not a suitable option. Find a subject matter expert interested in testing your software. That will give your QA team and developers explicit insight into your design, features, and functionality.

Coach testers. Keep in mind that you ask a subject matter expert for help, not a QA engineer. For that reason, create comfortable conditions for an end-user to get acquainted with testing requirements. Coach them to deal with a certain testing environment or tools that you use.

Reduce the usage of test tools. Your end-users will be thankful if you give them a less complicated tool for testing and reporting their observations. Consider using web-based environments like Plutora or Usersnap.

Respect their time. It's especially important to remember that your end-users are your future customers. Organize the process to be as convenient for them as possible. The simpler the testing requirements you create for them the better.

Test User Documentation

Any type of software developed has its User Documentation (UD). UD is a guide or a manual on how to use an application or a service. So, make sure you test your user documentation as well. Manuals for your software can also be tested by a team of end-user testers. Internal testers and tech writers take care of structure and navigation, while external teams help figure out if it's actually usable.

It is also a good practice to include user onboarding in your app. User onboarding consists of a set of methods to help users adapt to the interface, navigation, and guide through the app in general. For example, check Canva – a designer tool for non-designers. Canva shows a good example of user onboarding using videos, a "do, show, tell" approach, and overall user-friendliness. If you don't have user documentation and you opt for onboarding guides only, make sure that you engage your users to check how helpful and effective the onboarding is.

7. Optimize the use of automated tests

If you really want to improve the quality of your software, then automated testing or using automation tools to run the tests is definitely worth taking into consideration. According to the World Quality Report 2020-2021 by Capgemini, Sogeti, and Micro Focus, two of three key trends are increasing test automation and widespread adoption of the Agile methodologies. Test automation saves time, reduces human errors, improves test coverage and test capabilities, does batch testing and parallel execution. Here are main cases when applying automated testing enhances the process:

- cross-device and cross-browser testing
- regression and smoke testing
- load testing
- performance testing

To reach a perfect mix in testing, read out material on how to strike a balance between manual and automated testing.

There is a wide variety of automation testing tools. They can be both open-source and commercial. Selenium, Katalon Studio, Unified Functional Testing, Test Complete, Watir are the most popular ones worth checking first. To choose from the variety of software, read our comparison of the biggest test automation tools or the full Selenium review.

While automated testing can be employed within traditional Agile workflows, it is also a part of DevOps methodology and continuous integration practice.

Continuous integration and continuous delivery

Continuous integration (CI) is a development practice that requires engineers to integrate the changes into a product several times a day. Each code piece runs the "integration tests" at every code change to detect errors and bugs quickly, and locate them more easily. A good practice is to combine the CI with the automated testing to make your code dependable. Bamboo, Hudson, and Cruise Control are open source tools that allow for introduction of continuous integration in your environment.

Continuous delivery (CD) is considered an evolutionary development of the Agile principles. This method means that you can release changes to your customers quickly in a sustainable way. CD allows the commitment of new pieces of code when they are ready without short release iterations. Generally, you automatically deploy every change that passes the tests. This is achieved by a high level of testing and deployment automation.

CI and CD practices require *continuous testing* which brings test automation to the next level. They organize separate automated tests in a single system, making it a part of *CI/CD pipeline*. Read our article about continuous delivery and continuous integration to learn more.

8. Implement exploratory and ad hoc testing

With all the obvious benefits of testing automation, it still has certain limits. When a product has to be reviewed from a user's perspective, automation is not the best option, giving way to other testing methods.

The main idea of exploratory and ad hoc testing is human creativity. Both of them require little to no documentation, limited or no planning, and both are somewhat random, discovering unusual defects or defects that are not covered in the scope of other, structured, tests.

So, **exploratory testing** is a process of investigating a product with no predetermined test cases to examine how this product actually works. To uncover bugs, it demands experience, intuition, and imagination from testers. Exploratory testing is conducted on the fly, with a test being designed and executed immediately. Then the results are observed and used to fix possible bugs and design the next tests.

It's one of the best ways to test usability as it involves trying various real-life scenarios and user behaviors. Using this technique, the system can be assessed quickly, getting immediate feedback and discovering areas for further testing.

The main challenge of exploratory testing is that it's difficult to document its execution, replicate failures, and report defects, as there are no planned scripts or rigid structure.

The **ad hoc testing** is the most spontaneous and least formal method of testing based on error-guessing technique. It's typically conducted after all other tests and its main benefit is speed as it doesn't require any preparation and has no structured procedure to follow. It's often associated with the so-called *monkey testing*, when random tests are executed with some random data with the aim of breaking the system.

Such chaotic checking can help detect defects that are hard to find with formal tests and are hard to reproduce. However, results of ad hoc testing are unpredictable and, well, random.

These two above methods have a lot in common and are often confused. However, there are some differences.

- Ad hoc testing studies the software in advance; while in exploratory testing, the tester learns about the product while testing it.
- The exploratory testing process has some predefined limitations and scope, giving it some structure, unlike the completely random ad hoc approach.
- Ad hoc testing is mostly performed at the end of the development process after formal testing; while exploratory testing can be done at any time during sprints.

The best strategy would be complementing automated testing with exploratory and ad hoc testing. This way you can increase testing coverage, improve user experience, and come up with additional testing ideas.

9. Employ code quality measurements

If you still wonder how to improve software testing, make sure your quality objectives are measurable, documented, reviewed, and tracked. There single right way to measure code quality. The best advice is to choose metrics which are simple and effective for your workflow.

The CISQ Software Quality Model defines four important aspects of software quality: reliability, performance efficiency, security, maintainability, and rate of delivery. Additionally, the model can be expanded to include the assessment of testability and product usability.

Let's look at each of the main five aspects of software quality and explore how they can be measured:

Reliability. This indicator defines how long the system can run without failure. The purpose of checking reliability is to reduce application downtime.

You can measure reliability by *counting the number of bugs found in production***,** or by reliability testing, specifically, *load testing*, that checks how the software functions under high loads. It could also be *regression testing* which verifies the number of new defects when software undergoes changes.

Performance efficiency means the responsiveness of a system to execute any action within a given time interval. Performance efficiency can be measured using the following metrics:

- *Stress testing* provides the understanding of the upper limit of the capacity of the system.
- *Soak testing* checks how long the system can handle a certain load and when performance starts to degrade.
- *Application performance monitoring* is the detailed metrics of performance from the user's perspective provided by special software.

Security is the capability of a system to protect information against the risk of software breaches and to prevent the loss of information. You can count the number of vulnerabilities by scanning the software application. The number of found vulnerabilities is a positive or negative measure of security.

- *Deployment of security updates* is the percentage of users that have actually installed a patch or security update.
- *Time to resolution* is the amount of time needed to fix a vulnerability.

Maintainability is the ability of the system to modify software, adapt it for other purposes, transfer it from one development team to another, or meet new business requirements with a degree of ease. A very simple metric of code maintainability is to check the *number of lines of code* in a feature or even the entire application. Software with more lines of code is harder to maintain. You can also use the software complexity metrics such as *cyclomatic complexity* to measure how complex software is. More complex code is less maintainable.

The rate of delivery. It's also important to measure the rate of software delivery. The *number of software releases* is the main metric of how frequently new software is delivered to users. Consider reading our piece on main Agile development metrics to broaden your view on this topic.

10. Report bugs effectively

A good bug report will help make software testing more efficient by clearly identifying the problem and in this manner navigating engineers towards solving it. It must be a cornerstone and an efficient form of communication between a QA specialist and developer. A badly written report can lead to serious misunderstanding. Here are the guidelines for an effective bug report:

Provide solutions if possible. The document must include not only the bugs scenarios but also provide solutions for them, i.e. describing the needed behavior of a feature.

Reproduce a bug before reporting it. When reporting a bug, you want to make sure it is reproducible. Include a clear step by step instruction of how to reproduce a bug. Make sure you specify the context and avoid any information that can be interpreted differently. In case a bug is reproduced periodically, it is still worth reporting.

Clarity. A bug report must be clear enough to help developers understand the failure, including information about what QAs see, and a statement of what they expect to see. It should detail what went wrong. Clarity also entails addressing only one problem per task.

Screenshots. Include a screenshot of the examples of a failure highlighting a defect. This simplifies the work of an engineer who fixes the issue.

Consider adding a bug summary. A precise bug summary helps determine the nature of the bug much quicker, reducing fixing time. It's also useful in case of searching a bug in a bug inventory, as bug IDs are hard to memorize.

Bug report example

Source: discuss-gurock.com

The latest automated testing tools have built-in integration with bug-tracking systems. They can automatically report the bugs and track their status. There are also separate bug reporting tools like JIRA or Mantis.

11. Get the most out of your test management tool

Test management tools, or systems, are software products that help QA teams structure and manage the testing process. Such platforms can be integrated with your test automation frameworks, CI/CD tools, bug tracking tools, and other software solutions. They can:

- Plan testing activities
- Capture requirements
- Store testing information and results

- Design test cases
- Manage test case environments
- Create test execution reports
- Provide visibility by tracking KPIs
- Enable data sharing between team members and across different teams

When choosing, you'll have to decide between an open source platform (cheaper but offering less functionality and support) and a commercial one (fast, easy to use, feature-rich, and pricy). Typically, open source tools are a good option for smaller companies.

There's a wide variety of test management tools on the market for different needs and budgets. Here is a brief overview of a few popular platforms with good functionality.

1. Zephyr is the world's leading provider of test management solutions that supports Agile and DevOps frameworks. It offers several editions: Zephyr for Jira — a flexible, single-project tool operating natively inside the Atlassian Jira software; Zephyr Scale — a scalable, cross-project platform also inside Jira; and Zephyr Enterprise — a robust stand-alone solution to sync multiple teams.
2. SpiraTest is a powerful QA suite that helps with scheduling and managing defects and requirements. It offers full traceability, various test types support, and multiple report options.
3. TestRail is a comprehensive solution that provides numerous integration options with issue and bug trackers and test automation tools. It also has powerful reporting capabilities with customizable dashboards to effectively track test results and important metrics and obtain actionable insights.
4. Kualitee is a flexible product that allows for efficient defect management, test case management, and reporting. It integrates with various testing systems and has a mobile version.
5. Testpad is a simple, lightweight tool that is easy to use by both professional testers and other specialists, i.e., clients, managers, etc. It lets you create checklist-based test plans, add new tests easily, and adapt to various testing styles (BDD, TCM, exploratory, and more).

Whatever tool you choose, using test management systems can increase productivity by organizing the process, supporting communication, and visualizing progress.

Conclusion

If you want your company to be competitive and achieve a winning position in the IT industry market, you must produce very high-quality products. Improving the quality of software products will have the biggest overall impact on your business and its financial performance. When managing your work processes don't save on testing, as the cost of mistakes may prove to be too high. Consequently, your quality strategy should cover all key aspects: effective planning, a test-oriented quality management approach, and a dedicated QA team.

7 Comments

Further Reading

Striking a Balance Between Manual and Automated Testing: When Two Is Better Than One

Continuous Delivery and Integration: Rapid Updates by Automating Quality Assurance

Agile Software Development Metrics and KPIs that Help Optimize Product Delivery