altexsoft
software r&d engineering

WHITEPAPER

Quality Assurance, Quality Control and Testing — **the Basics of Software Quality Management**

Introduction

When you buy a pear, you can instantly evaluate its quality: the size and shape, ripeness, the absence of visible bruising. But only as you take the first bite, will you be able to see if the pear is really that good. Even an extremely good-looking pear might taste sour or have a worm in it.

The same applies to almost any product, be it a physical object or a piece of software. A website you find on the internet might seem fine at first, but as you scroll down, go to another page, or try to send a contact request, it can start showing some design flaws and errors.

This makes quality control so important in every field, where an end-user product is created. Yet, a sour pear won't cause as much damage as a self-driving car with poor quality autopilot software. A single error in an EHR system might put a patient's life at risk, while an eCommerce website that has performance issues might cost the owner millions of dollars in revenue.

That is why we at AltexSoft put a premium on the quality of software we build for our clients. In this paper, we will share our insights on the quality assurance and testing process, our best practices and preferred strategies.

1. Software Testing Basics

1.1. The Concept of Software Quality

While to err is human, sometimes the cost of a mistake might be just too high. History knows many examples of situations when software flaws might cause billions of dollars in waste or even lead to casualties: from Starbucks coffee shops being forced to give away free drinks because of a register malfunction, to the F-35 military aircraft being unable to detect the targets correctly because of a radar failure.

In order to make sure the released software is safe and functions as expected, the concept of software quality was introduced. It is often defined as *"the degree of conformance to explicit or implicit requirements and expectations"* [1]. These so-called explicit and implicit expectations correspond to the two basic levels of software quality:

- **Functional** — the product's compliance with functional (explicit) requirements and design specifications. This aspect focuses on the practical use of software, from the point of view of the user: its features, performance, ease of use, absence of defects.
- **Non-Functional** — system's inner characteristics and architecture, i.e. structural (implicit) requirements. This includes the code maintainability, understandability, efficiency and security [2].

The structural quality of the software is usually hard to manage: It relies mostly on the expertise of the engineering team and can be assured through code review, analysis and refactoring. At the same time, functional aspect can be assured through a set of dedicated **quality management** activities, which includes quality assurance, quality control and testing.

1.2 Quality Assurance (QA), Quality Control (QC) and Testing – What's the Difference?

Often used interchangeably, the three terms refer to slightly different aspects of software quality management. Despite a common goal of delivering a product of the best possible quality, both structurally and functionally, they use different approaches to this task.

QA, QC and Testing in software development process

Quality Assurance is a broad term, explained on the Google Testing Blog as *"the continuous and consistent improvement and maintenance of process that enables the QC job"*. As follows from the definition, QA focuses more on organizational aspects of the quality management, monitoring the consistency of the production process.

Through **Quality Control** the team verifies the product's compliance with the functional requirements. As defined by Investopedia, it is a *"process through which a business seeks to ensure that product quality is maintained or improved and manufacturing errors are reduced or eliminated"*. This activity is applied to the finished product and performed before the product release.

In terms of manufacturing industry, it is similar to pulling a random item from an assembly line to see if it complies with the technical specs.

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 has a very narrow focus and is performed by the test engineers in parallel with the development process or at the dedicated testing stage (depending on the methodological approach to the software development cycle).

	QA	QC	Testing
Purpose	Setting up adequate processes, introducing the standards of quality to prevent the errors and flaws in the product	Making sure that the product corresponds to the requirements and specs before it is released	Detecting and solving software errors and flaws
Focus	Processes	Product as a whole	Source code and design
What	Prevention	Verification	Detection
Who	The team including the stakeholders	The team	Test Engineers, Developers
When	Throughout the process	Before the release	At the testing stage or along with the development process

If applied to the process of car manufacturing, having a proper **quality assurance process** means that every team member understands the requirements and performs his/her work according to the commonly accepted guidelines. Namely, it is used to make sure that every single action is performed in the right order, every detail is properly implemented and the overall processes are consistent, so that nothing can cause negative impact on the end product.

Quality control can be compared to having a senior manager walk into a production department and pick a random car for an examination and test drive. **Testing activities**, in this case, refer to the process of checking every joint, every mechanism separately, as well as the whole product, whether manually or automatically, conducting crash tests, performance tests and actual or simulated test drives.

Due to its hands-on approach, software testing activities remain a subject of heated discussion. That is why we will focus primarily on this aspect of software quality management in this paper. But before we get into the details, let's define the main principles of software testing.

1.3. The Main Principles of Software Testing

Formulated over the past 40 years, the seven principles of software testing represent the ground rules for the process [3]. These are:

Testing shows presence of mistakes	Testing is aimed at detecting the defects within a piece of software. But no matter how thoroughly the product is tested, we can never be 100 percent sure that there are no defects. We can only use testing to reduce the number of unfound issues.
Exhaustive testing is impossible	There is no way to test all combinations of data inputs, scenarios and preconditions within an application. For example, if a single app screen contains 10 input fields with 3 possible value options each, this means to cover all possible combinations, test engineers would need to create 59,049 (310) test scenarios. And what if the app contains 50+ of such screens? In order not to spend weeks creating millions of such less possible scenarios, it is better to focus on potentially more significant ones.
Early testing	As mentioned above, the cost of an error grows exponentially throughout the stages of the SDLC. Therefore it is important to start testing the software as soon as possible, so that the detected issues are resolved and do not snowball.
Defect clustering	This principle is often referred to as an application of the Pareto Principle to software testing. This means that approximately 80 percent of all errors are usually found in only 20 percent of the system modules. Therefore, if a defect is found in a particular module of a software program, the chances are there might be other defects. That is why it makes sense to test that area of the product thoroughly.

Pesticide paradox	Running the same set of tests again and again won't help you find more issues. As soon as the detected errors are fixed, these test scenarios become useless. Therefore, it is important to review and update the tests regularly in order to adapt and potentially find more errors.
Testing is context dependent	Depending on their purpose or industry, different applications should be tested differently. While safety could be of primary importance for a fintech product, it is less important for a corporate website. The latter, in its turn, puts an emphasis on usability and speed.
Absence-of-errors fallacy	The complete absence of errors in your product does not necessarily mean its success. No matter how much time you have spent polishing your code or improving the functionality, if your product is not useful or does not meet the user expectations it won't be adopted by the target audience.

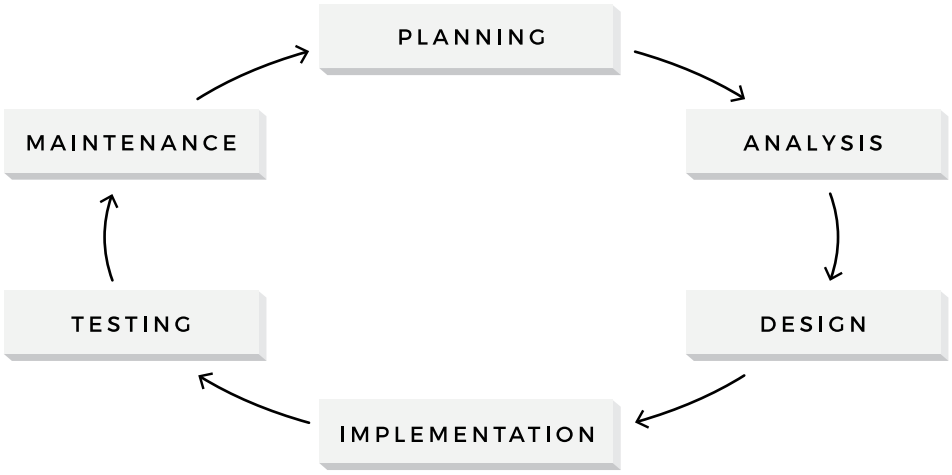
While the above-listed principles are undisputed guidelines for every software testing professional, there are more aspects to consider. Some sources note other principles in addition to the basic ones:

- Testing must be an independent process handled by unbiased professionals.
- Test for invalid and unexpected input values as well as valid and expected ones.
- Testing should be performed only on a static piece of software (no changes should be made in the process of testing).
- Use exhaustive and comprehensive documentation to define the expected test results.

2. The Process of Software Testing in Practice

2.1. The Role of Testing in Software Development Life Cycle

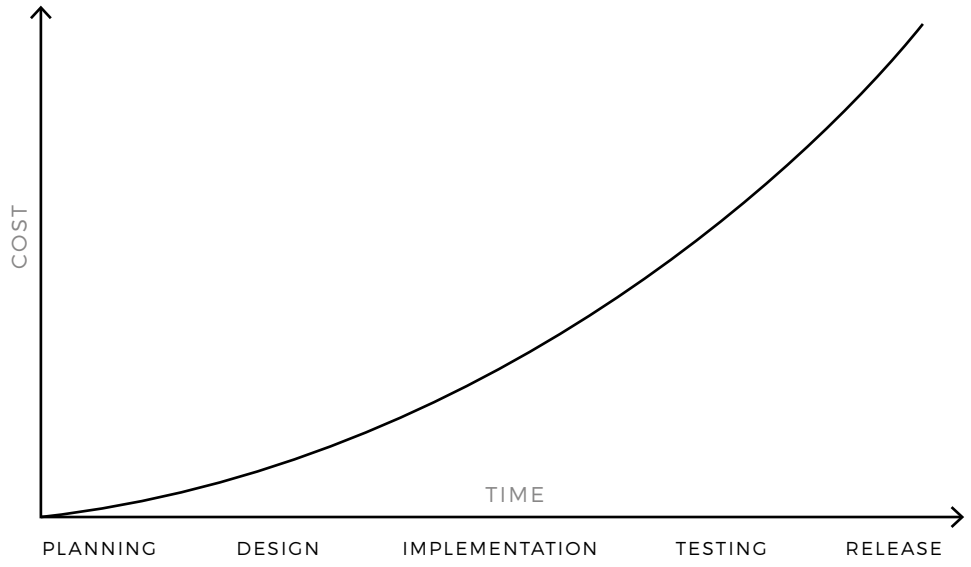
The traditional software development life cycle includes **6 consecutive steps**: planning, analysis, design, implementation, testing and maintenance.



Software Development Life Cycle

In **Waterfall** that follows this traditional model, testing is a separate stage that takes place after the implementation phase is over. A product, that is already designed and coded, is being thoroughly tested before the release.

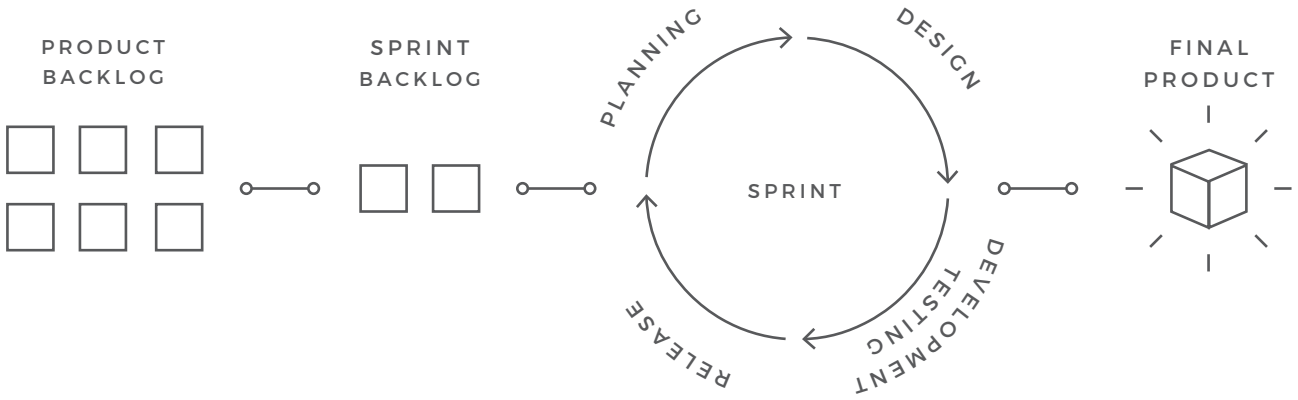
However, the practice shows that software errors and defects detected at this stage might be too expensive to fix, as the cost of an error tends to increase throughout the software development process.



The cost of an error throughout the SDLC

For example, if there is an error in the specifications, detecting it early in the planning stage wouldn't cause significant losses to your business. However, the damage grows exponentially throughout the further stages of the process. If such an error is detected at the design stage, you will need to rework your designs to fix it. But if you aren't able to detect the mistake before the product is built, you might need to make some major changes to the design as well as the source code. This will require a significant amount of effort and investment.

The same is the case for errors produced in the process of implementation. If a feature has a flaw in its logic, building more functionality on top of it might cause a serious damage in the long run. It is better to test every feature while the product is still being built. This is where **iterative** agile methods prove beneficial.



Agile Development Cycle

Agile testing is an integral part of the software development process. By breaking the development process into smaller parts, iterations and sprints, agile methodologies allow the testers to work in parallel with the rest of the team throughout the process and fix the flaws and errors immediately after they occur.

The main purpose of such process is to deliver new software features fast and with the best quality. Therefore, this approach is less cost-intensive: Fixing the errors early in the development process, before more problems snowball, is significantly cheaper and requires less effort. Moreover, efficient communication within the team and active involvement of the stakeholders speeds up the process and allows for better informed decisions.

According to the survey conducted by Zephyr, test management solutions provider, 71 percent of respondents refer to the process as the top challenge with agile testing. We at AltexSoft follow the three major steps in software testing process: planning, execution and reporting.

The stages of software testing

2.2. Test Planning: the Artifacts and Strategy

As any other formal process, testing activities are typically preceded by thorough preparations and planning. The main goal of this stage is to make sure the team understands the customer objectives, the main purpose of the product, the possible risks they need to address, and the outcomes they expect to achieve. One of the documents created at this stage, the **mission or assignment of testing**, serves to solve this task. It helps align the testing activities with the overall purpose of the product and coordinates the testing effort with the rest of the team's work.

Roger S. Pressman, a professional software engineer, famous author and consultant, states:

> *"**Strategy for software testing** provides a roadmap that describes the steps to be conducted as part of testing, when these steps are planned and then undertaken, and how much effort, time, and resources will be required."* [4]

Also referred to as test approach or architecture, test strategy is another artifact of the planning stage. James Bach, author of several books and creator of the Rapid testing techniques, identifies the purpose of a test strategy as *"to clarify the major tasks and challenges of the test project."* [5] A good test strategy, in his opinion, is product specific, practical and justified.

Depending on when exactly in the process they are used, the strategies can be classified as preventive or reactive. In addition to that, there are several types of strategies, that can be used separately or in conjunction [3]:

Strategy	Character	Primary Focus Area	Use Case
Analytical	Preventive	The strategy focuses on risk and requirements analysis to create a basis for planning, building and estimating the tests.	When building a well-defined product from scratch.
Model-Based	Preventive	The system is tested in accordance with the predefined model and should completely correspond to it in order to be considered valid.	When using an existing product as a basis for a new one, or enhancing the legacy system.
Methodical	Preventive	This strategy adheres to a custom pre-planned, systematic approach.	Often used in heavily-regulated industries, to build a product that complies with the requirements.

Strategy	Character	Primary Focus Area	Use Case
Process/ Standard-Compliant	Preventive	Unlike the previous one, this strategy relies on a standard strategy, with little or no adaptation.	Used by the teams that lack experience or time for building a custom testing approach.
Dynamic	Reactive	It prioritizes finding as many errors and defects as possible (the attack-based approach and the exploratory approach).	When there is a need to find and fix the issues with minimum time and effort.
Consultative/ Directed	Reactive	It relies on the users or developers to define the areas of testing or even to handle the tests themselves.	Applied to domain-specific products, that require additional expert guidance.
Regression-averse	Reactive	This strategy prioritizes the automation of functional tests either before the release or after.	Best used with live, well-established products.

While a test strategy is a high-level document, **test plan** has a more hands-on approach, describing in detail what to test, how to test, when to test and who will do the test [6]. Unlike the static strategy document, that refers to a project as a whole, test plan covers every testing phase separately and is frequently updated by the project manager throughout the process.

According to the IEEE standard for software test documentation, 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)
- Features to be tested
- Features not to be tested
- Item pass or fail criteria
- Test approach (testing levels, types, techniques)
- Suspension criteria
- Deliverables (Test Plan (this document itself), Test Cases, Test Scripts, Defect/Enhancement Logs, Test Reports)
- Test environment (hardware, software, tools)
- Estimates
- Schedule
- Staffing and training needs
- Responsibilities
- Risks
- Assumptions and Dependencies
- Approvals [7]

Writing a plan, which includes all of the listed information, is a time-consuming task. However, in agile methodologies, with their focus on the product instead of documents, such a waste of time seems insufficient.

To solve this problem, James Whittaker, a Technical Evangelist at Microsoft and former Engineering Director at Google, introduced The 10 Minute Test Plan approach. The main idea behind the concept is to focus on the essentials first, cutting all the fluff by using simple lists and tables instead of large paragraphs of detailed descriptions. While the 10-minute time-box seems a little bit unrealistic (None of the teams in the original experiment was able to meet this requirement), the idea of reducing and limiting the planning time itself is highly reasonable. As a result, 80 percent of the planning can be finished within only 30 minutes.

2.3. Design and Execution: Test Levels and Types

As a starting point for the test execution, we need to define what exactly we need to test. In order to answer this question, QA teams develop test cases. In a nutshell, a **test case** describes the preconditions, desired outcomes and postconditions of a specific test scenario, aimed at verifying that a feature meets the basic requirements.

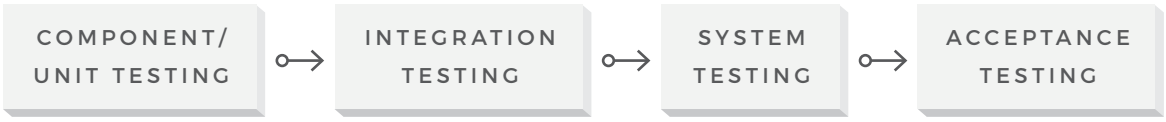
The next step in test execution is **setting up the testing environment**. The main criteria for this part is to make sure that the testing environment is as close to the end user's actual environment (hardware and software). For example, a typical test environment for a web application should include Web Server, database, OS, and browser.

As soon as the primary preparations are finished, the team proceeds with the execution. Usually, the different types of testing are conducted across several levels, using various tools.

2.3.1 The Levels of Testing

A piece of software is more than several lines of code. It is usually a multilayer, complex system, incorporating dozens of separate functional components and third-party integrations.

Therefore, efficient software testing should go far beyond just finding errors in the source code. Typically, the testing covers the following levels of software.

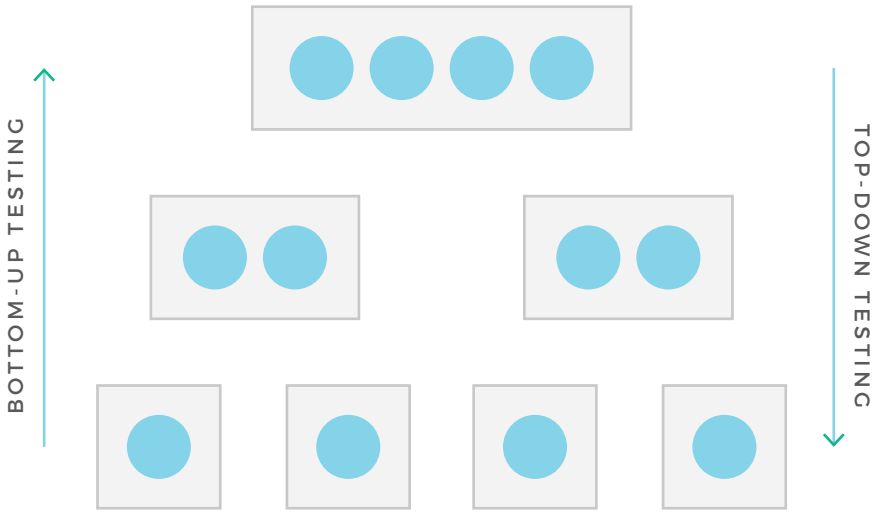


- **Component/Unit testing**

The smallest testable part of the software system is often referred to as a unit. Therefore, this testing level is aimed at examining every single unit of a software system in order to make sure that it meets the original requirements and functions as expected. Unit testing is commonly performed early in the development process by the engineers themselves, not the testing team.

- **Integration testing**

The objective of the next testing level is to verify whether the combined units work well together as a group. Integration testing is aimed at detecting the flaws in the interactions between the units within a module. There are two main approaches to this testing: bottom-up and top-down methods. The **bottom-up** integration testing starts with unit tests, successively increasing the complexity of the software modules under test. The **top-down** method takes the opposite approach, focusing on high-level combinations first and examining the simple ones later.



Integration testing approaches

- **System testing**

At this level, a complete software system is tested as a whole. This stage serves to verify the product's compliance with the functional and technical requirements and overall quality standards. System testing should be performed by a highly professional testing team in an environment as close to the real business use scenario as possible.

- **Acceptance testing**

This is the last stage of the testing process, where the product is validated against the end user requirements and for accuracy. This final step helps the team decide if the product is ready to be shipped or not. While small issues should be detected and resolved earlier in the process, this testing level focuses on overall system quality, from content and UI to performance issues. The acceptance stage might be followed by an alpha and beta testing, allowing a small number of actual users to try out the software before it is officially released.

	Unit testing	Integration	System	Acceptance
Why	To ensure code is developed correctly	To make sure the ties between the system components function as required	To ensure the whole system works well when integrated	To ensure customer's and end user expectations are met
Who	Developers / Technical Architects	Developers / Technical Architects	SDET / Manual QA / Business Analyst / Product Owner	Developer / SDET / Manual QA / Product Owner / Product End Users
What	All new code + refactoring of legacy code as well as Javascript unit Testing	New web services, components, controllers, etc.	Scenario Testing, User flows and typical User Journeys, Performance and security testing	Verifying acceptance tests on the stories, verification of features

	Unit testing	Integration	System	Acceptance
When	As soon as new code is written	As soon as new components are added	When the product is complete	When the product is ready to be shipped
Where	Local Dev + Continuous Integration (CI, as a part of the build	Local Dev + CI (part of the build)	Staging Environment	CI / Test Environment
How (tools and methods)	Automated, Junit, TestNG, PHPUnit	Automated, Soap UI, Rest Client	Automated (Webdriver) Exploratory Testing	Automated (Cucumber)

In agile software development, the testing typically represents an iterative process. While the levels generally refer to the complete product, they can also be applied to every added feature. In this case, every small unit of the new functionality is being verified. Then the engineers check the interconnections between these units, the way the feature integrates with the rest of the system and if the new update is ready to be shipped.

2.3.2. The Types and Methods of Software Testing

Based on the main objective of the process, the testing can be of different types.

Testing Objective	Testing Type
Software functions — the testing of the functions of component or system.	Functional testing (black-box testing) • Requirement-based testing • Business-process-based testing
Software characteristics — testing the quality characteristics of the component or system.	Non-Functional testing • Reliability testing • Usability testing • Efficiency testing • Maintainability testing • Portability testing • Baseline testing • Compliance testing • Documentation testing • Endurance testing • Load testing • Performance testing • Compatibility testing • Security testing • Scalability testing • Volume testing • Stress testing • Recovery testing • Internationalization testing and Localization testing
Software Structure (architecture) — testing of the structure of the system or component.	Structural testing (white-box testing)
Changes — testing modifications and updates to make sure that the system still works as needed.	Confirmation testing (re-testing), regression testing

While the above-listed types of testing are considered the major ones, the list is far from being complete. Adopted by 66.3 percent of the teams [8], **exploratory testing** is one of the most popular, yet often misunderstood, software testing approaches.

The term was first described by Cem Kaner, a software engineering professor and consumer advocate, as:

> *"a style of software testing that emphasizes the personal freedom and responsibility of the individual tester to continually optimize the value of her work by treating test-related learning, test design, test execution, and test result interpretation as mutually supportive activities that run in parallel throughout the project."* [9]

As opposed to the scripted approaches (most of the types listed above), exploratory testing does not rely on a predefined and documented test cases and test steps.

Instead, it is interactive and free-form process, with the main focus on validating user experience, not code. It has much in common with the ad hoc or intuitive testing, but is more systematic. Its procedure includes specific tasks, objectives, and deliverables. When practiced by the skilled testers, it can provide valuable and auditable results.

While testing was traditionally handled manually, automation trends are gaining wider popularity. Accordingly, 58.5 percent of ISTQB® survey respondents [8] state test automation activities as the main improvement area in their organizations. This is probably due to the ever-growing adoption of agile methodologies, which promote both test automation and continuous integration practices as the cornerstone of effective software development.

The process of test automation is typically conducted in several consecutive steps:

- Preliminary Project Analysis
- Framework Engineering
- Test Cases Development
- Test Cases Implementation
- Iterative Framework Support

Automation can be applied to almost every testing type, at every level. As a result, the automation minimizes the human effort required to efficiently run the tests, reduces the cost of an error and increases time to market, as the tests are performed up to 10 times faster, when compared to manual testing process. Moreover, such a testing method is more efficient as the framework covers over 90 percent of the code, uncovering the issues that might not be visible in manual testing, and can be scaled as the product grows.

However, the most effective testing approach combines both manual and automated testing activities in order to achieve the best results.

2.3.3. Documentation and Reporting

As there is no perfect software, the testing is never 100 percent complete. It is an ongoing process. However, there exists the so-called "exit criteria", which defines whether there was "enough testing" conducted, based on the risk assessment of the project.

IBM offers several options for exit criteria:

- Test case execution is 100 percent complete.
- A system has no high priority defects.
- Performance of the system is stable regardless of the introduction of new features.
- The software supports all necessary platforms and/or browser
- User acceptance testing is completed [10].

As soon as all of these criteria (or any custom criteria that you have agreed on in your project) are met, the testing comes to its closure.

The testing logs and status reports are documented throughout the process of the test execution. Every issue found in the product should be reported and handled accordingly. The test summary and test closure reports are prepared and provided to the stakeholders. The team holds a retrospective meeting in order to define and document the issues that occurred during the development and improve the process.

Conclusion

In 2012, Knight Capital Americas, a global financial firm, experienced an error in its automated routing system for equity orders - the team deployed untested software to a production environment. As a result, the company lost over $460 million in just 45 minutes [12], which basically led to its bankruptcy.

History knows many more examples of software incidents which caused similar damage. Yet, testing remains one of the most disputed topics in software development. Many product owners doubt its value as a separate process, putting their businesses and products at stake while trying to save an extra penny.

Despite a widespread misbelief that a tester's only task is to find bugs [11], testing and QA have a greater impact on the final product success. Having deep understanding of the client's business and the product itself, QA engineers add value to the software and ensure its excellent quality. Moreover, applying their extensive knowledge of the product, testers can bring value to the customer through additional services, like tips, guidelines and product use manuals. This results in reduced cost of ownership and improved business efficiency.

References

1. http://softwaretestingfundamentals.com/software-quality/
2. http://www.davidchappell.com/writing/white_papers/The_Three_Aspects_of_Software_Quality_v1.0-Chappell.pdf
3. https://www.amazon.com/Foundations-Software-Testing-ISTQB-Certification/dp/1844809897
4. https://www.amazon.com/Software-Engineering-Practitioners-Roger-Pressman/dp/0073375977/
5. http://www.satisfice.com/presentations/strategy.pdf
6. http://www.testingexcellence.com/test-strategy-and-test-plan/
7. https://standards.ieee.org/findstds/standard/829-2008.html
8. http://www.istqb.org/references/surveys/istqb-worldwide-software-testing-practices-report.html
9. http://kaner.com/?p=46
10. https://www.ibm.com/developerworks/community/blogs/Govind_Baliga/entry/defining_exit_criteria_for_testing5?lang=en
11. http://www.ibm.com/developerworks/rational/library/apr06/rose/
12. https://www.sec.gov/litigation/admin/2013/34-70694.pdf

About AltexSoft

AltexSoft is a Technology & Solution Consulting company co-building technology products to help companies accelerate growth. The AltexSoft team achieves this by leveraging their technical, process and domain expertise and access to the best price-for-value Eastern European engineers. Over 100 US-based and 200 worldwide businesses have chosen the company as their Technology Consulting Partner.

US Sales HQ

701 Palomar Airport Road,

Suit 300, Carlsbad, CA 92011

+1 (877) 777-9097

Global HQ

32 Pushkinskaya Str.,

Kharkiv, Ukraine 61057

+38 (057) 714-1537