What Is Software Testing?
All the Basics You Need to Know

Contents

Introduction to software testing

Picture this. Your organization has been working hard on its product for two years. The release of the first version is approaching quickly, and everyone is excited. The stakes are high, so people are nervous, but they're also confident on the quality of their work.

Then, the big day finally comes… and it's terrible. Sure, the application looks great, and its features are fantastic. However, the app is riddled with embarrassing bugs. Users aren't satisfied and the reviews are unforgiving.

How could this be prevented? The answer is, of course, software testing.

To err is human. No matter how much of a perfectionist you are, we're all bound to make mistakes. Every organization has an end goal that comes with its own set of expectations.

For some businesses, success is indicated by a high frequency of real outcomes matching expected results. But before reaching their end goal, every firm has to face the consequences of human errors.

No enterprise can use manual error as an excuse for delivering a compromised product. To ensure a high-quality product, there has to be something in place to pick out errors. Software testing is an essential solution to this problem for software development companies.

In this post, I'm going to cover some software testing basics that you need to know. Here are some of the topics you'll read about:

- Software testing definition
- The importance of software testing
- The types of software testing
- How does the software testing process work?
- How can you decide which types of software tests to use?

What Is Software Testing? A Definition

Software testing is the process of finding errors in the developed product. It also checks whether the real outcomes can match expected results, as well as aids in the identification of defects, missing requirements, or gaps.

Testing is the penultimate step before the launch of the product to the market. It includes examination, analysis, observation, and evaluation of different aspects of a product.

Professional software testers use a combination of manual testing with automated tools. After conducting tests, the testers report the results to the development team. The end goal is to deliver a quality product to the customer, which is why software testing is so important.

Importance of Software Testing

It's common for many start-up's to skip testing. They might say that their budget is the reason why they overlook such an important step. They think it would lead to no major consequences. But to make a strong and positive first impression, it needs to be top-notch. And for that, testing the product for bugs is a must.

To really understand why software testing is important, we need to correlate it with real world examples, which has caused serious issues in the past, a few examples includes;

- In October 2014, Flipkart an e-commerce in India Company had an offer called the "Big Billion Sale." When it was launched it had a lot of traffic and as a result, its website couldn't handle the enormous load of traffic leading to the website downtime, cancellation of orders etc. The reputation of the organization was badly impacted by this issue.
- In 2015, the Royal Bank of Scotland, due to a bug, couldn't process about 600,000 payments. Because of this, they were fined 66 million pounds
- Yahoo in September 2016, had a major data breach where 500 million users' credentials got compromised.

- Recently, Okta, an American authentication firm, had a digital breach due to a software bug that may have affected their user's details. This has also affected the reputation of Okta.

Similarly, established organizations need to maintain their client base and their impression. So they have to ensure the delivery of flawless products to the end-user. Let's take a look at some points and see why software testing is vital to good software development.

Enhance Product Quality

An enterprise can bring value to their customers only when the product delivered is ideal. And to achieve that, organizations have to make sure that users don't face any issues while using their product. The fool-proof way of doing it is by making your product bug-free.

Organizations have to focus on testing applications and fix the bugs that testing reveals before releasing the product. When the team resolves issues before the product reaches the customer, the quality of the deliverable increases.

Improve Security

When customers use the product, they are bound to reveal some sort of personal information. To prevent hackers from getting hold of this data, security testing is a must before the software is released. When an organization follows a proper testing process, it ensures a secure product that in turn makes customers feel safe while using the product.

For instance, banking applications or e-commerce stores need payment information. If the developers don't fix security-related bugs, it can cause massive financial loss.

The other part of security is not losing your data. It's common today for people to store data in cloud storage. You also probably have your photos and files stored on iCloud or Google drive.

What if something goes wrong and you lose all your data? One of your nightmares isn't it? Security of a product not only protects information from hackers but also makes sure it's not lost or gets corrupted.

Detect Compatibility with Different Devices and Platforms

The days are gone when customers worked exclusively on hefty desktops. In the mobile-first age, testing a product's device compatibility is a must.

For instance, let's say your organization developed a website. The tester must check whether the website runs on different device resolutions. Additionally, it should also run on different browsers.

Another reason why testing is gaining more importance is ever-increasing browser options. What works fine on Chrome may not run well on Safari or Internet Explorer. This gives rise to the need for cross-browser testing, which includes checking the compatibility of the application on different browsers.

Classifications of Software Testing

Software testing isn't a single thing. Instead, it comes in many different variations, which you can categorize according to several criteria.

For instance, you can categorize testing types into manual or automated testing. When it comes to the automated variety, tests can be code-based or codeless—and you can also have hybrid approaches that mix the best of both worlds.

Tests can also be categorized regarding how much they know about the internal implementation of the system under test. Regarding this criterion, we can classify tests as white-box, black-box or grey-box. Finally, we can also group tests into functional and non-functional tests, depending on whether they validate the business requirements for the application.

Functional Testing

Functional testing verifies each function of an application or software. The tester verifies functionality with a specified set of requirements. So the source code of a software or an application doesn't play a major role in this case. Testing the behaviour of the software is the main concern.

The different types of functional testing include:

- **Unit testing**. In unit testing, the tester checks individual software components. The aim is to test whether the components behave according to the requirements.
- **Integration testing**. Integration testing deals with testing individual components or modules after they are combined in a group.
- **System testing.** Here, the tester executes test cases for verifying the compliance of integrated and completed software along with specifications.
- **Sanity testing**. This tests logical reasoning related to the working of the program.
- **Smoke testing**. Smoke testing tests simple and basic functionalities, such as if the user is able to log in or log out.
- **Interface testing.** These tests check whether the communication between two software systems is correctly carried out.
- **Regression testing.** This is probably one of the most important testing phases. Here, the old test cases of the entire application are executed after a new functionality has been implemented.
- **Beta/acceptance testing**. Here, the intended users try the product and report bugs.

Non-Functional Testing

Non-functional testing considers parameters such as reliability, usability, and performance. A non-functional test might be checking how many users can log in to the system at the same time.

Non-functional testing types include:

- **Performance testing**. The performance or speed of the application is tested under the required workload.
- **Load testing.** This tests an application's behaviour under a huge workload. So, if you're testing a website, load testing checks the site's functionality and performance under high traffic.
- **Stress testing.** Stress testing determines software robustness by assessing whether it's working beyond regular operation.
- **Volume testing.** This tests the performance of the system by loading the database to an increased volume of data.

- **Security testing**. Here, test cases are executed to check whether the system is safeguarded against sudden or deliberate attacks from internal and external sources.
- **Compatibility testing**. Test cases are executed to check whether the application is compatible with varying environments. For example, if you're testing a web application, compatibility testing deals with how the website works on different browsers or devices.
- **Install testing**. These tests check if a product works according to expectations after installation.
- **Recovery testing**. Here, testers determine an application's capacity to recover from hardware crashes and failures.
- **Reliability testing**. This procedure checks where an application can perform a particular task without failure within a specific timeframe. For example, suppose you're testing a cryptocurrency mining application. The scenario where the application can mine continuously for eight hours without crashing might be something you look for during reliability testing.
- **Usability testing**. Usability testing explores the end-user's ease of use in terms of learning, operating, and preparing inputs and outputs.
- **Compliance testing**. This determines the system's compliance with external and internal standards.
- **Localization testing.** Here, testers check the behaviour of a product according to local or cultural settings and environment.

Based on the amount of information you know about the product to test it, software testing can be divided into different types: Black-box testing, White-box testing, and Grey-box testing.

Black-box Testing

In this type of testing, you have the least amount of information on how the product is built. You don't know about the structure of the product, its code, or logic. You would use the product as an end user would. Because in black-box testing, you'd have the same amount of information as your customer, it is used for functional testing.

This type of testing can only happen when the code is executed. Hence, dynamic testing is used. Dynamic testing is the type where you have to execute the code and test the product while the code execution is in process. It is mostly done to

check how the would be when it's up and running and how the user would experience it.

White-box Testing

In white-box testing, you have most of the information about the product. White-box testing is mostly used to make the code better. Finding inefficiencies in code, poor coding practices, and unnecessary lines of code are identified in this type of testing. Most of the code optimization and security fixes happen as a result of this testing.

White-box testing doesn't mainly focus on how the web application is working. It rather focuses on how it can be made better. You can make a lot of improvements to your product but the last few steps to make it perfect is difficult. And it can't be perfect until it has no issues whatsoever.

Making it perfect requires a thorough inspection. Because a product in execution can't give you all the insights, you'll have to check the code without execution. This is known as static testing.

Static testing is also used in the early stages of development when it's simple and you needn't wait for product deployment.

Gorilla Testing

Gorilla testing is a type of software testing, where a module is tested frequently using some random inputs, and ensuring that modules are checked with no errors. This type of testing is done manually and repeatedly, where only a few selected modules of the system are subjected to testing with the goal of determining whether or not the module is functioning properly. Other names for Gorilla testing include; torture testing, fault tolerance testing, and frustrating testing.

Grey-box Testing

In this type of testing, you have partial information about the product. This type of testing is helpful to find out bugs that the user wouldn't know about.

To give you a very simple example, if you've designed an element to have a blue shade but it has a green shade. The user wouldn't know that it's a bug because they'd think that's how it's supposed to be. But your partial knowledge of the product would help you identify such bugs.

Now that you understand what testing's all about, it's time you know how to go about the software testing process.

Software Testing Process

Like any other process, software testing can also be divided into different phases. This sequence of phases is often known as the software testing life cycle. Let's look at them in brief.

Planning

Every process starts with planning. In this phase, you collect all the required details about the product. You collect a list of tasks that has to be tested first. If you're testing after a bug fix, then you'd want to know what the bug was and what the ideal behaviour is.

Then you have the prioritize your checklist of tasks. If a complete team is involved, then division of tasks can also be done in this phase.

Preparation

Once you know what you have to do, you have to build the foundation for testing. This includes preparing the test environment, collecting test-cases, researching product features and test-cases. Gathering tools and techniques for testing and getting familiar with them should also be done here.

Execution

This is when you actually run tests on the product. You execute test-cases and collect the results. Then you compare the results with the expected result and see if the product is working as expected or not. You make a note of all the successful and failed tests and test-cases.

Reporting

This is the last phase of software testing where you have to document all your findings and submit it to the concerned personnel. Test-case failures are of most interest here. A proper and clear explanation of tests run and outputs should be mentioned.

For complex tests, steps to reproduce the error, screenshots, and whatever is helpful should be mentioned.

Two Ways to Test

As we know, in the current age of machines, everything that involves manual effort is slowly automated. And the same thing is happening in the testing domain. There are two different ways of performing software testing—manual and automation.

Manual labour in any field requires a lot of time and effort. Manual testing is a process in which testers examine different features of an application. Here, the tester performs the process without using any tools or test scripts. Without using any automated tools, testers perform execution of different test cases. Finally, they generate a test report.

Quality assurance analysts test the software under development for bugs. They do so by writing scenarios in an excel file or QA tool and testing each scenario manually.

But in automated testing, testers use scripts for testing (thus automating the process). The pre-scripted tests run automatically to compare actual and expected outcomes. With test automation, when constant human intervention is not necessary, things like regression testing and execution of repetitive tasks don't seem like much effort.

Is Automated Testing Making Manual Testing Obsolete?

Now that you've got the gist of what manual and automated testing are, we need to clarify an important question.

Is automation testing making manual testing obsolete? No.

Even though the automatic performance of most processes takes place in automation testing, some manual labour is still a must. Generating the initial script for testing requires human efforts. Also, in any automated process, human supervision is mandatory.

Automation simply makes the testing process easier. However, it doesn't make manual testing obsolete. You only get the best result by combining both manual and automated tests.

Why Is There Such a Huge Demand for Test Automation?

Since testing is more efficient and speedy, there's a huge demand for automation testing compared to manual testing. And the reason is that it helps find more bugs

in less time. By checking every single unit, automated testing also increases test coverage. As a result, an organization's productivity increases.

How to Choose Between the Different Types of Software Testing?

Enter the Test Automation Pyramid

As you've seen, software testing comes in many shapes and sizes. Each type provides a different kind of feedback, which means you can't use them interchangeably. Also, each type of testing comes with its own costs and associated challenges.

Considering that your team and organization have limited resources, how can you choose between the many available types of testing in a way that maximizes test coverage, ensuring you can ship high quality software while using your resources in the most efficient way?

That's where the concept known as the test automation pyramid comes in handy.

We do have an entire post about this concept, but here's the condensed version: the test automation pyramid—aka the testing pyramid, for short—is a concept that helps you think about the different types of software testing and pick between them.

At the bottom of the pyramid, you have unit tests. Unit tests are easier and cheaper to write than most other forms of testing. Since they don't talk to external dependencies, they run fast and are extremely precise in the feedback they provide. So, it makes sense to have lots of them.

The middle of the pyramid is comprised of service tests, or integration tests. They offer a more "realistic" feedback than the unit tests, due to validating the integration between units and talking with real dependencies. But because of that they're also slower to run, more complex to write and maintain, and offer a less precise feedback.

Integration tests are valuable, but due to their limitations, it makes sense to employer fewer of them.

Finally, the top of the pyramid contains end-to-end tests. End-to-end tests are the most realistic of all the software testing types since they exercise the application in the same way a real user would. However, they tend to be slower and fragile, besides being more expensive to write, maintain and execute.

It pays to have these types of tests on your test suite, but you'd be wise to have few of them.

Testing Is Software's Lifeline

No company can underestimate the importance of delivering the best possible product to customers. And the types of testing keep evolving and the list keeps going on. Depending on the nature and scope of your products, you can run different testing procedures.

Once the testing team gives the green signal, the deliverable is ready to go out into the market. But enterprises still need to keep in mind that customer trust doesn't come easily. To help earn customer trust, you need to provide consistent, reliable products. That's why testing is an integral part of the software development life cycle.

Automating the Tests

The next step will be to automate these tests. An application without a UI like a REST API can be tested by executing a series of calls to the API and checking the results. Tools like Postman or a Cucumber library are useful here.

UIs are traditionally harder to run in automated tests. Tools like Selenium can run automated UI tests, but easily lead to fragile tests. They also require a decent amount of work to build and maintain.

With Testim, you can create tests just by using your application. Testim records your actions and can then run those actions as an end to end test. Testim is unique because it will learn about your application.

If you change your UI, Testim is able to notice this and adapt the test. This leads to less fragile tests, allowing you to focus more on improving your product.