SCA
Open Source or 3rd party Libraries

SAST
Vulnerabilities in code

DAST
Black box test your web application

API Scanning
Vulnerabilities in your APIs

Container Vulnerabilities
Vulnerabilities in your Containers

Kubernetes Vulnerabilities
Vulnerabilities in your K8S setup

Terraform Vulnerabilities
Vulnerabilities in your Terraform

CSPM - Cloud Security Posture Management
Secure your cloud from threats

OSS license management

Testing

10 Best Practi ure Testing Projec

Mark Preston
24 Aug 7 min read

Open Source or 3rd party licenses

The complexity of modern era software applications, with the need to deploy them across different platforms and devices has made software testing imperative.

The added urgency of the news constantly brimming with successful cyberattacks has also heightened the need to embrace thorough quality assurance (QA) testing methodologies.

When testing best practices are following, the resulting quality assurance yields high-performing software that performs as expected to the delight of customers.

However, with so many software testing practices constantly being heralded as "the-next-best-thing," it can be daunting to pin down which among them really constitute the best practice to be followed.

This post is here to allay those concerns by providing a comprehensive list of the 10 best software testing practices to follow on a project.

Best Practices for Software Testing Projects in 2020

Poor quality assurance is the root cause of most products failures.

Companies strive to eliminate bugs and prevent errors in their application, since these problems tend to drive away customers and make their product susceptible to malicious attacks.

One of the ways they achieve this is by putting in place a system or team accountable for quality control within the organization's structure.

One of these is software quality assurance (SQA), which falls under the rubric of quality management.

It is comprised of a set of organizational actions whose goal is to foster the creation of a culture that improves software development. This quality assurance (QA) accomplishes this objective through the introduction and setting of measurable, attainable standards of quality.

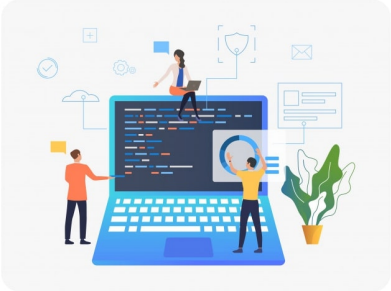
Quality Management and Quality Control

Software quality control (SQC), or quality control for short, is also part of quality management. Quality control (QC) however, is focused on enacting a set of activities (and culture) geared to fulfilling quality requirements.

Quality control is product-oriented, and it gives it's stamp of approval on products that attain predetermined quality requirements by certifying them before release. QC doesn't work in isolation; the processes and decisions of QC are driven by the standards set by the QA team.

The purpose of software testing is to detect issues, bugs, and problems in a software product so these can be resolved or mitigated expeditiously.

1. Adopt a Controlled Security Test Environment with a Dedicated Team



The cloud has many benefits but it has created a false sense of security among software development teams who effectively outsource their security obligations to it, despite its many vulnerabilities.

Moreover, most products deployed to the cloud are usually provided as Software-as-a-service (SaaS), where vulnerabilities are mostly discovered by end-users, to the frustration of these customers.

Instead of capitalizing on the cloud, companies should rather invest in having their own secure testing infrastructure to avoid the pitfalls of cloud and internet-based vulnerabilities.

Maintaining a Dedicated Software Security Testing Team

First and foremost, organizations that have the resources should maintain a dedicated software security testing team committed to performing the range of penetration tests necessary to fortify the software in order to discover potential vulnerabilities.

Preferably, they should be familiar with relevant OWASP critical security risks profiles and ISO security standards, not to mention undergo regular, rigorous security training.

By creating local testing environments and labs, QA teams can simulate regional test scenarios, and assess with how reliably responsive their user-interface reacts under different conditions.

Hard-won knowledge garnered from stress testing applications on controlled environments provides test engineers the insight to discover design flaws before anyone spends time writing one line of code.

Therefore, a controlled environment should be setup for software test cycles before code is deployed to the cloud or otherwise released to customers.

2. Proactively Plan Software Test Cycles

QA tests and their associated processes should be deliberately planned ahead of time. In that way, terms and objectives can be defined and a robust documentation process created.

Good documentation greases the wheels of effective communication among software test teams. It builds institutional knowledge of the best practices for software testing that has been adopted. Hence, when new members are added to the team, or original ones leave, there is still adequate paper trail documenting these best practices.

To plan ahead, some type of quality assurance plan is required. Quality assurance planning includes the following: having a quality management plan, developing a test strategy, coming up with test plans, and building test cases.

Quality Management Plan

This document contains the software quality requirements. It outlines the acceptable level of quality aimed for, and how the project ought to go about to achieve this level of quality.

Test Strategy

Instead of software requirements, test strategy focuses on the project's business requirements. It is therefore, a more high level document that falls within the purview of the project manager.

Test Plan

This is an operational document that pinpoints the testing scope and other associated activities - what to test, how to test, when to test, and who is responsible for the testing.

Test Case

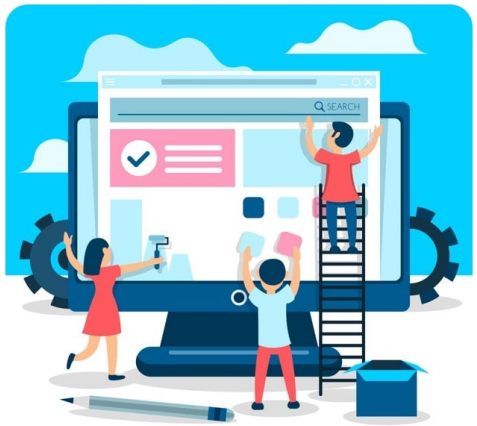
Outlines the actions or set of conditions to be tested.

3. Use Development Practices That Are Test-Oriented

It is self-defeating and grossly inadequate to wait until the QA stage to test software under development.

Rather, testing should be ingrained from the beginning of the development process, using test-oriented development practices such as Test-drive Development (TDD), pair-programing, and unit-testing.

In so doing, bugs can be caught much earlier in the process and eliminated sooner.



Test-Drive Development (TDD)

This practice adopts a test-first approach whereby test are written before the actual code for the software being implemented. A new feature starts with an automated test case that will initially fail.

Subsequently, the code is written to ensure the functionality passes the test. TDD is known to have positive effects on productivity, code quality, simplification, and executable documentation.

Pair Programming

When it comes to testing scrutiny, two heads are better than one. This is a form of extreme programming where two engineers are paired to work collaboratively on a single computer. At any point, one writes the code while other makes suggestions.

When tests are implemented throughout the software development cycle, it builds confidence among developers, the quality control team, and management that the software being shipped is robust and of acceptable quality.

Apart from improving quality, the approach of implementing testing throughout development reduces the maintenance and labor costs of fixing bugs at a much later stage.

4. Ensure All Tests Are Integrated in CI/CD Pipeline

The era of the DevOps methodology has arrived, and any software testing team worth its salt needs to have an automated CI/CD pipeline. Besides, manual QA processes are error prone. Therefore, it doesn't make sense to neglect a crucial aspect of modern quality assurance testing.

Employing testing automation as part of the CD/CI setup is essential to make certain only bug-free code is deployed into the production system.

Therefore, in order to get the best results from development efforts, every code checked into the CI pipeline should be automatically tested.

Moreover, automated CI/CD pipelines make it unavoidable to test as early as possible, which enhances security by enabling issues to be detected faster in the production chain.

CI/CD pipelines remove unnecessary complexity from the testing process. Since testing can be run by the execution of a single command, or by click of a button, software testers and other stakeholders involved in quality control are more likely to embrace it.

5. Adopt Tests Written for Maximum Coverage

Different types of testing are adopted by QA teams for different scenarios. However, the overarching objective should be to make sure

that the tests are written extend to the maximum coverage possible, if 100% isn't exactly practical.

To make the products requirements are testable, test cases should be designed to cover them also for adequate analysis.

It is impossible to anticipate all potential threats a software product might face, not to mention hidden vulnerabilities that might be exposed once it is let out into the wild. To counteract this uncertainty, it is advisable to incorporate a two-tier approach to test automation.

The first tier could be triggered whenever code is committed to the shared repository. The tests involved at this level are incorporated for quick validation of the changes a developer is introducing into the main branch of the project, accompanied with sanity and until tests that normally don't run longer than a couple of minutes.

The second-tier portion is usually more exhaustive and typically runs at night when there is more time to test the changes that have been made. In this phase, regression tests are included to ensure that the changes made haven't broken or negatively impacted existing functionality.

6. Run Regular QA Technical Reviews

Formal technical reviews are a beneficial tool to include in the software testing arsenal because they reveal whether there are logical or functional errors present, especially at the early stages of product development.

A logical error is caused by a bug that makes a feature behave incorrectly, though it doesn't crash or terminate the application. Functional errors, on the other hand,

occurs when poor exception handling breaks referential transparency.

Technical Reviews

These technical reviews more or less occur in formal settings like a group meeting where stakeholders with different roles ensure the software satisfies certain requirements and standards.

These meetings are called formal technical reviews (FTR) and are usually conducted on mature products where the market segmentation and target users are already entrenched.

A review report is produced after an FTR which addresses the following questions: what exactly was reviewed, who did the reviewing, and what findings were discovered during the review, including subsequent decisions reached.

FTR's aren't monolithic measures, but they come in different size and methods, namely:

Review Meeting

This is a formal review conducted by the product's author to introduce it to the other reviewers. Modifications are entertained and release time-frames discussed.

Walkthrough

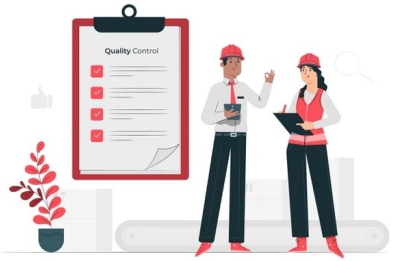
A technical meeting where the product's source code is examined with the intention of detecting bugs, including assessments of its design and documentation processes.

Inspection

Held to evaluate whether to expand initial standard, or assess if earlier bugs are still present.

7. Enact User Acceptance Testing (UAT)

User personas are contrived in product development to approximate and target the ideal user for the product.



They are relevant for QA teams who envision the behavior of these fictional characters to guide them in anticipating how a bug or logical error might be triggered in their interaction with the product.

But even with effective tool of a user persona, software testing teams are still unable to anticipate the full extent of unexpected behavior patterns that actual users might exhibit.

Therefore, the introduction of end-users is paramount in the software testing cycle. They are usually reserved for the final phase of a product's quality control testing. Reaching this stage of testing is an important milestone that indicates the application is almost production ready.

End-users see an application or product with fresh eyes; it's features and responses are novel to them and they therefore have the unique perspective of looking from an outside-in paradigm.

There are 5 types of UAT, namely:

Contract Acceptance Testing

Determines whether the software meets contract requirements.

Alpha and Beta Testing

Both are pre-release phase testing. Alpha is done at the early stage while beta testing often involves willing customers in their environment.

Regulation Acceptance Testing

Determines whether the software meets legal requirements.

Black Box Testing

The content of a black box is unknown. Likewise, in this type of test, the testers know what an application should do, but are unaware of how it ought to do it or accomplish the task.

Production Readiness Testing

As the name implies, it ascertains whether the software is ready for usage, with workflows that have been verified.

8. Measure Code Quality.

There is a correlation between the quality of a software product and the quality of its code. But if it can't be measured, then it cannot be effectively evaluated.

As a result, the best way to improve QA quality is to ensure its objectives are measurable.

This quantifiability, in turn, makes it possible to track, review, and document quality control metrics for the product being tested.

However, there isn't any standard way to measure code quality; only common sense guidelines and rules of thumb to be followed.

Luckily, there's no need to reinvent the wheel: there exists a CISQ Software Quality Model, which details four crucial areas of software quality, namely: security, performance efficiency, reliability, rate of delivery, and maintainability.

Security

The ability of a system to protect its information and prevent data from being breached. This is measured in the number of vulnerabilities discovered, and the time taken to fix those errors.

Reliability

This measures how well the system can run without failing or experiencing downtime. It is gauged by the number of failures, or bugs encountered within a calendar time.

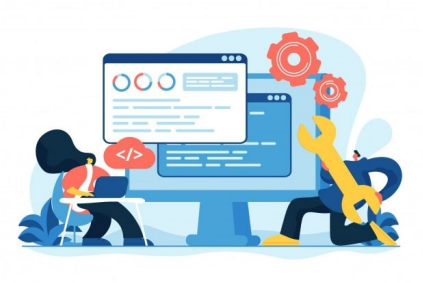
Maintainability

The degree of ease which the application presents itself to be modified, extended, or adapted to new business requirements. Lines of code in an application are a proxy measure for maintainability, since more voluminous code tends to increase complexity.

Performance Efficiency

Response time measured by load and stress testing.

9. Develop Requirements That Are Testable



In software testing, the manner in which business or functional requirements are written determine their testability - or lack thereof.

A well-written requirement will aptly describe the behavior of a software feature in such a way that tests can likely be developed that evaluate whether the requirement's conditions have been met.

As was intuited in the previous section on measuring code quality, a fundamental necessity for a requirement to be testable is that it has to be measurable. In addition, it ought to be written with clarity, completeness, and without any ambiguity.

Ideal Methods to Writing Requirements

There are several approaches to writing tests that accommodate a range of different possibilities; from the more conventional requirements documents, to the more nimble agile methods such as the following:

Test-Driven development (TDD)

In this developer testing mode, requirements are written as unit tests. However, testers can also work with developers to construct better tests, using techniques such as equivalence partitioning, and

boundary value analysis.

Behavior-Driven Development (BDD)

BDD uses real examples instead of abstract terminology using a syntax called Gherkin.

User Stories

This method focuses on the user or customer. They are written in the format of *user-role*, *feature*, and *goal*; and are thus meaningful to end-users.

Acceptance Test-Driven Development (ATDD)

This combines user stories with acceptance criteria to depict whether the software application is working as it was intended.

10. Understand Product Objectives in Order to Design Effective Test Strategies

To be able to interrogate and test a system effectively, testers must understand the product from a business standpoint, and not only see it from a technical perspective.

When testers are armed with only the knowledge of the scope of their tests, they usually can't see the forest for the trees. To successful test software, it isn't enough to have a narrow mindset fixated on technical functionality.

The hallmark of a good software tester is understanding how the entire product architecture comes together to address business and customer needs through its various dependencies, components, data flow, and integration points.

Test Execution

During test execution, this global view and purpose perspective will equip them as QA analysts to identify the root cause of a problem or defect when it occurs because they understand the flow of the application.

Moreover, grasping the underlying business objectives will help the tester design more robust test scenarios that'll consider all possible user interactions.

In addition to more easily identifying risk, this knowledge will also positively impact other facets of the testing regime, such as defining the test strategy, and preparing the test schedule.

Summary

High-quality products are not the result of random luck. Such products evolve from a concerted effort to create a culture of excellence, usually by establishing high-level quality assurance teams to oversee the software testing and development cycles.

Companies that are serious about competing on quality imbibe all, if not a combination of some of the best software testing practices listed in this article.

Mark Preston

Mark Preston is a passionate software engineer, with over two decades of experience. His main fields of interest are DevSecOps, software development concepts and practices, and open source tools.

Related Posts

Embracing the Benefits of Single Delivery Platform Cybersecurity

You needn't worry any longer as the search for answers to the latest security issues is over. Learn how embracing the benefits of single delivery platform will help protect your business.

Mark Preston
21 May 7 min read

How to Avoid OSS License Compliance Lawsuits and Vulnerabilities

Worried about OSS licenses? You needn't worry any longer as we got you covered with our how to avoid OSS license compliance lawsuits and vulnerabilities guide.

Barbara Ericson
10 min read

Web Application Security

With web application security add an extra layer of protection to your application and stop DDoS attacks and data breaches before they even occur.

Mark Preston
27 Jan 11 min read