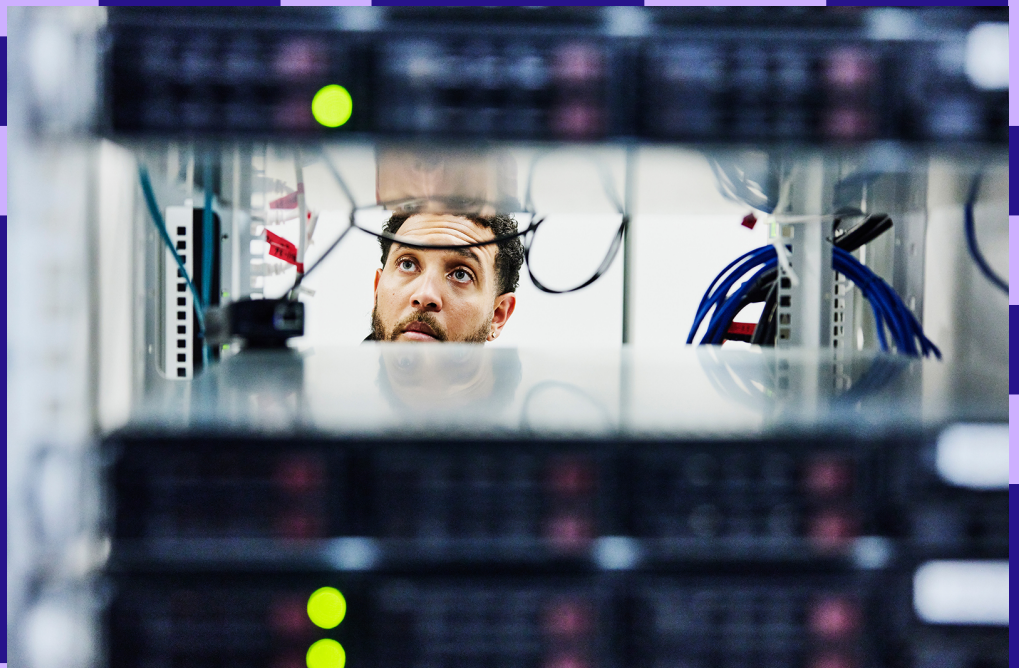


A beginner's guide to software testing

Automate. Accelerate. Deploy Quality.

Get everything you need to know to make educated decisions around manual testing, automated testing and software quality management.



1. Introduction to Software Quality Management (SQM)

SQM, or Software Quality Management, refers to the administrative processes involved in creating software. SQM is a multi-faceted process that begins with a product concept and continues through its conception, testing, and public release.

The main components of SQM are Quality planning, Quality Assurance (QA), Quality Control (QC) and software testing.

Quality planning

Software quality management begins during the planning phase when testers create goals and objectives for the software and create a strategic plan to help achieve those goals. Quality planning is the backbone of SQM since it establishes the parameters by which the whole process will be carried out.

Quality Assurance (QA)

Quality Assurance (QA) is a part of Software Quality Management (SQM) that ensures the software product is built according to an approved design specification. Testers follow the development process, checking for conformance to standards and ensuring that all design aspects are working correctly.

Quality Control (QC)

Quality control is the testing phase where defects are identified, functionality is evaluated, and other tasks are performed to iron out kinks and make final minor adjustments.

The ISTQB® defines quality control as “activities designed to evaluate the quality of a component or system.” While Quality Assurance is about creating and maintaining the process that leads to the end-result, QC examines the final outcome – it looks at the quality of the end-products.

The terms quality assurance and quality control (QC) are often used interchangeably, albeit incorrectly — as QA is process-oriented and focused on preventing defects, and QC is product-oriented and focused on identifying defects – mainly by testing.

Software testing

Software testing is just one subset of Quality Control (QC) and is a process that assures the quality of software products by testing and validating them before being released to users. It helps ensure that the end product is of the best quality based on design, functionality, and specifications and meets defined standards (such as ISO 9000, CMMI, TMMI, etc.).

Software testing also helps companies avoid making expensive mistakes by catching flaws before launch. These measures all contribute to giving consumers the best experience possible.

In this guide, we'll focus predominantly on the final aspect of SQM, testing.

What types of tests can be performed?

Software testing can be manual or automated, depending on the product's nature. As the name suggests, manual testing requires human interaction and testing without automated tools. QA testers follow a test plan and interact with the application as though they were a genuine user to analyze the functionality and experience of the program.

On the other hand, automated QA testing uses automation tools to execute test cases. Therefore, it is more suitable for large-scale testing and helps to speed up repetitive testing tasks.

Functions of quality assurance

- **Validation:** To determine whether the product satisfies the specific needs of the user or not. It also helps show that the program works as intended.
- **Documentation:** To keep track of activities related to the development and testing processes
- **Assuring Quality of Products:** To ensure that products meet customer expectations regarding functionality, performance, safety, and durability.
- **Quality Improvement Plan:** To effectively monitor and control quality across the software development life cycle (SDLC)

Quality assurance certifications

There are several industry certifications that organizations can earn to prove that they meet industry standards and maintain quality practices. Clients also use these certifications as a way to evaluate software vendors.

- **ISO 9000:** International standards for establishing and maintaining an adequate quality assurance (QA) system for businesses. To be ISO 9000 certified, an organization must be audited on its functions, products, services, and processes.
- **Capability Maturity Model Integration (CMMI) Level:** Used to analyze the maturity of an organization's processes and provide recommendations for process improvement.
- **TMMI:** A five-level model that provides a framework to help companies assess the maturity of their testing processes and optimize them.

In the next section, we'll discuss manual testing in detail — and exactly how you can use it for specific use cases, including an introduction to using Test Center for manual testing.

2. The role of manual testing in software development

Companies carry out manual software testing to ensure their new applications or products are free of bugs and defects before being released to the public. It involves having a Quality Assurance (QA) team interacting with the program as a user would, reviewing the behavior of software against predefined expected behavior, and reporting any issues they may find.

If a company was launching an app, for example, then the QA team would have to:

- Click on buttons or links to see if they work correctly.
- Check if users can enter data into text fields.
- See if search bars, drop-down menus, and navigation is working.

Testing types used in manual testing processes

Black Box testing

This method, also called "behavioral testing," helps to look at the app from the perspective of its intended users. Black box testing shields the QA team from knowledge of the program's inner code. By simulating real-world user actions, the app's functional and non-functional behavior can be tested confidently and help find any bugs that have gone unnoticed.

White Box testing

White box testing, also known as transparent box testing or structural testing, is performed when the tester has access to and is aware of the software's underlying structure and logic. The tester selects inputs, conducts the test using code, and determines outputs. Its goal is to make the application more secure while enhancing its aesthetics and usability.

Note: You might come across the term "Gray box testing" — this simply refers to a combination of Black Box and White Box testing.

Unit testing

Also known as module testing or component testing, unit testing tests each unit or component of an application's source code to ensure that each function works as planned. Since it requires a comprehensive understanding of the program's architecture and code, developers typically execute this type of test — rather than test engineers. In addition to making error detection and prevention more straightforward, it also reduces debugging.

Integration testing

Integration testing evaluates programs with several integrating components. It is carried out after unit testing and seeks to find issues with the interfaces and their interactions.

System testing

Sometimes referred to as end-to-end testing, system testing is performed after unit testing and integrating each component to test the system as a whole. It validates the final application by contrasting it to the initial requirements.

Acceptance testing


Acceptance testing is where the actual end-user verifies that the application satisfies their needs and meets the agreed-upon requirements — typically in the last phases of the project.

Step-by-step: how to do manual testing

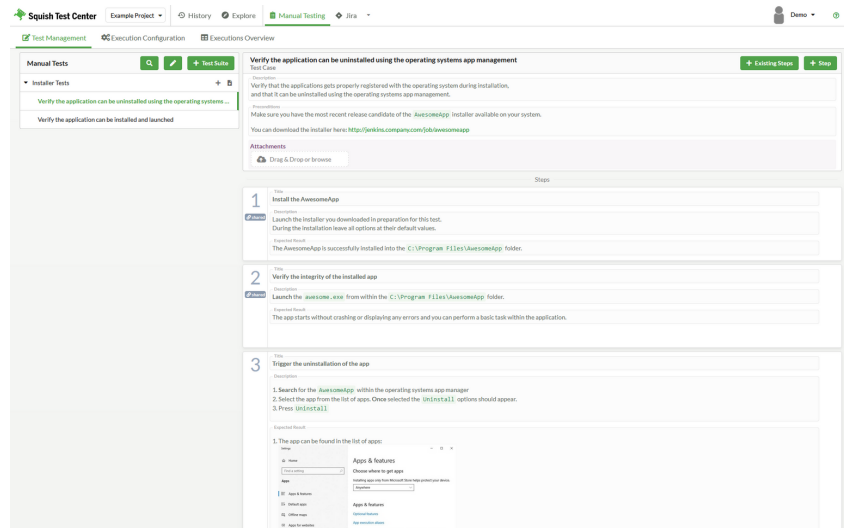
- The project requirements are examined to check for client expectations, what needs to be done and to know what the expected behavior of the software should be
- A test plan is made
- Test cases that meet the requirements and cover various scenarios are created
- Test cases are executed to identify any issues
- Any issues found are reported and repaired
- Once repaired, the failed tests need to be rerun to validate the fixes



2.1. Manual testing with Test Center

 **Test Center** is a centralized, compact platform for managing test results and integrating testing throughout the entire development cycle.

It comes with a results dashboard that serves as a consolidated hub for storing, tracking and evaluating test results gathered at every stage in a project's development. The built-in statistical reporting features also make it easy to assess the overall health of development projects over time and as they progress.

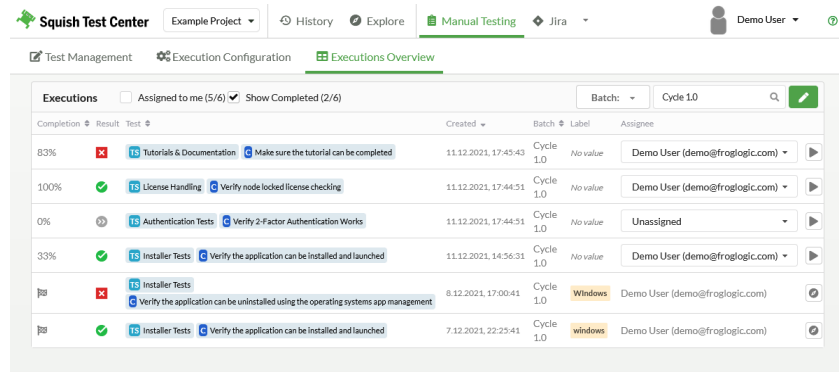


The test management page has all the necessary information to generate manual tests, making creating and maintaining tests a breeze.

The text editing feature comes with markdown support, saving time and eliminating laborious text editing. In addition, attachments and in-line images can be used, allowing testers easy access to all relevant data and documentation while carrying out manual tests. Another helpful tool is shared steps, allowing you to reuse existing stages. This is especially helpful for manual testing that needs distinct setup procedures.

Managing a test cycle

In Test Center, manual test cycles are conducted in batches. This connects manual and automated tests. When a batch of manual testing is complete, the corresponding results will automatically be reflected in the relevant result views.



The execution overview page summarizes the status of all manual tests and how far along in the cycle they are. Depending on the available testers, manual testing can also be assigned or reassigned here.

Similarly, testers can utilize this view to see a run-down of all the assigned manual tests and initiate the manual testing process. Execution overview also enables completed test cycles to be reviewed and easily rescheduled if they need to be rerun.

Manual test execution

Manual testing is just as simple to execute as it is to create. You can use the same text-editing tools you used to create your tests to run them.

Testers can easily add descriptions, screenshots, or other attachments to the observed manual test results if they spot any errors. This makes it easy for them to document a test failure with supporting information.

Issue tracking & traceability

Test Center 2.0 introduced improved integration with Jira, especially issue creation, by supporting various custom Jira fields. Additionally, the release of [Test Center 3.0](#) earlier this year supports [Coco](#) coverage reports, enabling users to browse and analyze their code coverage reports directly from within the Test Center. It also has the enhancements of earlier versions of the Test Center.

When testers find defects, they can quickly create issues on Jira while doing manual tests within Test Center. Bugs created during manual testing will be accessible to all system users. In particular, shared steps ensure that errors discovered in earlier test runs will also be accessible to other testers.

Shifting from other manual tools

Due to Test Center's dedicated API-based import mechanism, those already handling manual tests in Zephyr or Xray can easily switch to Test Center's manual testing with little configuration. It is also possible to import Gherkin feature files, which eases the transition for users who have documented their manual tests in this language.

Next, we'll dive into automated testing, four reasons to use automated testing, and a run-down of how to use Squish for automated testing.

3. Benefits of automated testing in software development

Automated testing is the process of using scripted sequences to execute a test case suite. Examples of languages used by QA testers are Python, JavaScript, and Ruby.

The script is employed to test for bugs, defects, and any other errors that could occur during the development of a product. After completing the test, it produces a report outlining its findings, which can then be compared to the results of previous tests. The main benefit of automated testing is simplifying manual labor into a set of scripts that can be executed repetitively at any time of day.

Reasons to use automated testing

1. **Cost-effective:** Automation testing reduces the overall cost of testing by eliminating many manual tests, which can be time-consuming and expensive
2. **Speed:** Automatic tests can run faster than manual tests because they don't require human interaction or input. As such, automatic testing can produce results much more quickly.
3. **Lower risk:** Manual testers often make mistakes when running their test cases, which may not happen if testers use automated tools.
4. **Improved product quality:** Having fewer bugs in your software means a better user experience for customers and clients, increasing sales for businesses using your software.

What kinds of tests should be automated?

To maximize test automation ROI, you must choose the most suitable tests to be automated.

Here are some considerations:

- Tests that are vulnerable to human error
- Tests that are repetitive and monotonous
- Extensive tests that necessitate the use of many data sets
- Tests that cannot be carried out manually (for example, tests with extensive data sets)
- Time-consuming tests
- Business-critical or high-risk tests that must be completed consistently
- Tests that must be performed on multiple hardware or software platforms

Types of automation tests

Code analysis

Code analysis is the process of checking source code for errors and vulnerabilities. This can be done with different tools, such as a dynamic or static analysis tool. Dynamic analysis is performed while the program is running, while static analysis happens before software testing begins without having to run the program.

Unit tests

Each program's function is tested in a unit test to ensure it functions properly. This ensures that the software's components have been thoroughly tested before the final release.

Integration tests

End-to-end tests, commonly referred to as integration tests, determine if the individual components of your application are functioning correctly. The application models are integrated and tested to assess how effectively they perform, allowing communication between modules to be evaluated.

Automated Accepted Test (AAT)

AAT is usually the last stage of testing. Finally, it tests the whole concept in a production-like setting to demonstrate that the application performs as intended by the user.

Smoke tests

The purpose of a smoke test is to provide a quick way to get a preliminary assessment of the software's functionality. If it doesn't pass the assessment, it is marked as an "unstable build" and returned to the developers. Then, they can perform more tests to find the problem's source.

3.1. Automated testing with Squish

Automated testing is an integral part of the software development process. Testing is used to verify your application's functionality and uncover bugs that need squishing.

One solution for automated testing is (the aptly named) Squish, a tool that offers functional and visual (GUI) testing capabilities with support for Qt, Java, Web, Windows, iOS, Android and more.

Squish provides an IDE and a framework that allows you to implement and automate acceptance testing for GUIs and verify in real-time whether or not an application complies with its requirements.

It can record test scripts written in JavaScript and other languages and employs property-based object identification. It has two parts: the runner, which reads and runs scripts, and the server hooks, which control the application being tested.



Squish features:

- Supports all major GUIs frameworks
- Full support for PC, mobile, web, and embedded platforms
- Test script recording
- Strong object ID and verification
- Well-integrated environment
- Support for behavior-driven development
- Full control with command line tools

Broad support for automated GUI testing

Squish's UI technology supports automated GUI testing of Windows, Mac, Java, and web apps, as well as mobile platforms like Android and iOS.

Squish is easy to use and operate. Users can create test cases with standard programming languages like Python, JavaScript, Ruby, Perl, and Tcl. They also can use the Gherkin testing language to test using the Behavior-Driven Development (BDD) approach.

Furthermore, Squish features automatic test script recording and recognition of high-level interactions and objects instead of low-level events. All the tester has to do is press the record button and then proceed with their work within the app. As soon as the actions are performed, Squish automatically creates a test script.

Verification and recognition tests

Squish has various verification and recognition features that make it easy to test your application. For example, you can verify by object property, compare two screenshots, visually compare complex items' information, geometry, topology, and aesthetic appeal, or perform OCR-text and picture searches.

Testing Qt Apps

Qt is fully supported by Squish, which allows users to test Qt applications easily. Qt applications can be tested on one supported platform and run on another without changing the code. Squish usually just needs the binaries.

Squish also supports Qt Widgets, QML, and QtQuick controls in addition to typical and complicated buttons, menus, lists, tables, and so on. Qt WebKit and Qt Web Engine-implemented embedded web content are also recognized.

Whether it's Qt and QML controls, models or other objects, Squish makes them all accessible. Users can also link test-script functions to Qt signals and Qt events so that a test script can respond to application signals and events.

Squish supports automated testing of basic gestures like touch, flick, swipe, and more. Pinch and other complicated multi-touch movements are supported too.

The Squish development team has invested a lot of time and energy in testing In-Vehicle Infotainment (IVI) apps using the Qt IVI module. Both C++ and QML allow access to these vehicle features. In addition, a core API for adding new IVI features and bindings to the Qt IVI module for testing vehicle interactions are also available.



Applications

Because Squish can work with many kinds of software developed using advanced technology, the tool is popular in software development. In addition, it can be used for a variety of testing methodologies.

For example, Companies can use Squish for user interface testing, functional testing, regression testing, and keyword-driven testing. Squish can also test applications on various platforms like Android and Java.

Benefits of Squish

- High consistency and minimal redundancy in testing
- Excellent reusability and scalability capabilities
- Modifiable according to the requirements of integration
- High-performance and improved efficiency during the test execution phase

3.2. Using Test Center to improve test management

Using a test result management platform, such as Test Center, can enhance both your test management and reporting capabilities. It helps serve as a centralized, organized repository of tests and results from all tests.

In Qt Test Center, you can view your automated and manual test suite statistics, visualize trends and analyze historical data of your test executions with built-in, automatic statistical reporting of your imported data.

4. Switching from manual to automated testing

Manual testing is a way of emulating user behavior to ensure software works as intended. Although manual testing is more efficient for some situations, it's not always the best choice. From the complexity of managing a test environment to the risk presented by human error, manual testing presents several roadblocks, prompting many businesses to reconsider whether it is the ideal option for their next project.

Automated testing methods can help companies overcome these obstacles and deliver more opportunities for testing in less time.

The Advantages of automation in testing

Less time spent on routine testing

Because it takes so much time, testers can only cover limited testing territory manually. Manual testing is also more challenging to scale as software grows more complex. Automated testing, on the other hand, expedites the quality assurance process by eliminating the need for human intervention during routine tests like regression testing.

Increased test coverage

With a manual testing approach, you can only run so many tests before hitting the ceiling, and the quality of those tests frequently relies heavily on the expertise of the specific tester. In addition, due to the tester-specific nature, it is difficult to replicate the same set of tests every time.

Test coverage can be increased with automated testing, as unlike manual testing, automated testing can perform several tests simultaneously, on different setups, and with varying parameters.

Reduced risk for human error

Human error is more likely to occur when individuals are involved in a manual testing process. No matter how careful, no one is perfect. There may be instances where someone skips a few steps, which may end up increasing their workload and delaying the development process even more. As such, automated testing lessens the likelihood of mistakes being made. Additionally, since every action is automatically recorded, any problems may be found and fixed immediately.

Making the shift from manual to automated testing

Here are some helpful tips for shifting to automated testing.

Decide which test cases should be automated

When planning to move to automated testing, determine how it can help maximize your team's efficiency. For instance, automating repetitive tests can save you time and effort.

- **Regression tests:** These are extensive, recurring, and dependent on the same input variables
- **Data-driven tests:** Many functional tests require testing using multiple divergent data sets to evaluate a range of positive and negative cases
- **Performance testing suites:** Automated testing can be used to speed up the testing of system performance under varying conditions
- **API tests:** Automated tests find flaws faster since they trigger API regressions whenever the API is modified

Your team should also be able to identify which test cases to automate and which not to. Some test cases are more effective when done manually or cannot be automated, for example:

- **Exploratory testing:** Real users explore programs differently from standardized routines. Exploratory testing can't be automated since it requires human cognition.
- **UX testing:** Automated tools must catch up when capturing intangibles like how people feel about a product, how likely they are to utilize it, and how aesthetically pleasing it is.
- **Testing accessibility:** This is best accomplished through manual testing, which examines the user's interaction with the process or application.

Determine the right framework

The efficiency of the test suite relies on the chosen framework. The right choice will depend on the programmer's skillset, the organization's software development procedures, and the nature of the software.

For reliable outcomes, choose frameworks that mesh well with the skillsets of your team members. For example, JavaScript, Python, Ruby, Tcl and Perl are often used for test automation frameworks. However, it's best to choose a framework that your team is most comfortable with to lessen the learning curve.

Choose the right tools

Shifting to test automation requires proper tooling. It is essential to choose something that meets your business objectives.

Similar to frameworks, it's essential to consider your objectives not just for now, but in the future. For example, will the tool still exist in five years? Are new releases issued frequently? How about support?

Ultimately, the right tools should help you address your issues and accommodate your evolving requirements.

Set manageable goals and a quick learning curve

Test automation is frequently hailed as the solution to many software testing challenges. However, transitioning from manual to automated testing may be challenging, so starting with manageable objectives is better.

You could automate the most-used regression test suite or the longest and most tedious one. First, identify recurring bugs and eliminate them, then gradually improve automation to cover more testing areas.

Measure the automated testing tool's performance

It is essential to have metrics to guarantee that your tool is effective and can meet your testing goals. A few starting points could be to determine:

- How fast the turnaround time for executing tests is now, compared to before using the tool
- The difference in time spent on creating and revising tests
- If the time for developing test suites has been shortened

5. Conclusion

Software testing is not only a requirement, but a critical factor in the market success of a product, and all future products a company might release. Implementing a software quality management strategy with a strong focus on testing ensures that your company will follow best practices and that your final product will meet or exceed the expectations of your users.

