SmartBear
SOFTWARE

6 Tips to Get Started with Automated Testing

Automated testing will shorten your development cycles, avoid cumbersome repetitive tasks and help improve software quality but how do you get started? These best practices a successful foundation to start improving your software quality.

Introduction

Thorough testing is crucial to the success of a software product. If your software doesn't work properly, chances are strong that most people won't buy or use it…at least not for long. But testing to find defects – or bugs – is time-consuming, expensive, often repetitive, and subject to human error. Automated testing, in which Quality Assurance teams use software tools to run detailed, repetitive, and data-intensive tests automatically, helps teams improve software quality and make the most of their always-limited testing resources. Automated testing helps teams test faster, allows them to test substantially more code, improves test accuracy, and frees up QA engineers so they can focus on tests that require manual attention and their unique human skills.

Use these best practices to ensure that your testing is successful and you get the maximum return on investment (ROI).

1. Decide what Test Cases to Automate

2. Test Early and Test Often

3. Select the Right Automated Testing Tool

4. Divide your Automated Testing Efforts

5. Create Good, Quality Test Data

6. Create Automated Tests that are Resistant to Changes in the UI

1. Decide What Test Cases to Automate

It is impossible to automate all testing, the first step to successful automation is to determine what test cases should be automated first.

The benefit of automated testing is correlated with how many times a given test can be repeated. Tests that are only performed a few times are better left for manual testing. Good test cases for automation are those that are run frequently and require large amounts of data to perform the same action.

You can get the most benefit out of your automated testing efforts by automating:

- Repetitive tests that run for multiple builds
- Tests that are highly subject to human error
- Tests that require multiple data sets
- Frequently-used functionality that introduces high risk conditions
- Tests that are impossible to perform manually
- Tests that run on several different hardware or software platforms and configurations
- Tests that take a lot of effort and time when doing manual testing

Success in test automation requires careful planning and design work. Start out by creating an automation plan. This plan allows you to identify the initial set of tests to automate, and serve as a guide for future tests. First, you should define your goal for automated testing and determine which types of tests to automate. There are a few different types of testing, and each has its place in the testing process. For instance, unit testing is used to test a small part of the intended application. Load testing is performed when you need to know how a web service responds under a heavy workload. To test a certain piece of the application's UI, you would use functional or GUI testing.

After determining your goal and which types of tests to automate, you should decide what actions your automated tests will perform. Don't just create test steps that test various aspects of the application's behavior at one time. Large, complex automated tests are difficult to edit and debug. It is best to divide your tests into several logical, smaller tests. This structure makes your test environment more coherent and manageable and allows you to share test code, test data and processes. You will get more opportunities to update your automated tests just by adding small tests that address new functionality. Test the functionality of your application as you add it, rather than waiting until the whole feature is implemented.

When creating tests, try to keep them small and focused on one objective. For example, use separate tests for read-only versus read/write tests. This separation allows you to use these individual tests repeatedly without including them in every automated test.

Once you create several simple automated tests, you can group your tests into one, larger automated test. You can organize automated tests by the application's functional area, major/minor division in the application, common functions or a base set of test data. If an automated test refers to other tests, you may need to create a test tree, where you can run tests in a specific order.

2. Test Early and Test Often

To get the most out of your automated testing, testing should be started as early as possible in the development cycle and run as often as needed. The earlier testers get involved in the life cycle of the project the better, and the more you test, the more bugs you find. You can implement automated unit testing on day one and then you can gradually build your automated test suite. Bugs detected early are a lot cheaper to fix than those discovered later in production or deployment.

3. Select the Right Automated Testing Tool

Selecting an automated testing tool is essential for test automation. There are a lot of automated testing tools on the market, and it is important to choose the tool that best suits your overall requirements.

Consider these key points when selecting an automated testing tool:

- Support for your platforms and technology. Are you testing .Net, C# or WPF applications and on what operating systems?
- Flexibility for testers of all skill levels. Can your QA department write automated test scripts or is there a need for keyword testing?
- Feature-rich but also easy to create automated tests. Does the automated testing tool support record-and-playback test creation as well as manual creation of automated tests? Does it include features for implementing checkpoints to verify values, databases, or key functionality of your application?
- Create automated tests that are reusable, maintainable and resistant to changes in the applications UI. Will your automated tests break if your UI changes?

For detailed information about selecting automated testing tools for automated testing, see our Whitepaper *"Selecting Automated Testing Tools"*.

4. Divide Your Automated Testing Efforts

Usually, the creation of different tests is based on the skill level of the QA engineers. It is important to identify the level of experience and skills for each of your team members and divide your automated testing efforts accordingly. For instance, writing automated test scripts requires expert knowledge of scripting languages. Thus, in order to perform that task, you should have QA engineers that know the script language provided by the automated testing tool.

Some team members may not be versed in writing automated test scripts. These QA engineers may be better at writing test cases. It is better when an automated testing tool has a way to create automated tests that does not require an in-depth knowledge of scripting languages, like TestComplete's "keyword tests" capability. A keyword test (also known as keyword-driven testing) is a simple series of keywords with a specified action. With keyword tests, you can simulate keystrokes, click buttons, select menu items, call object methods and properties, and do a lot more. Keyword tests are often seen as an alternative to automated test scripts. Unlike scripts, they can be easily used by technical and non-technical users and allow users of all levels to create robust and powerful automated tests.

You should also collaborate on your automated testing project with other QA engineers in your department. Testing performed by a team is more effective for finding defects and the right automated testing tool should allow you to share your projects with several testers.

5. Create Good, Quality Test Data

Good test data is extremely useful for data-driven testing. The data that should be entered into input fields during an automated test is usually stored in an external file. This data might be read from a database or any other data source like text or XML files, Excel sheets, and database tables. A good automated testing tool actually understands the contents of the data files and

iterates over the contents in the automated test. Using external data makes your automated tests reusable and easier to maintain. To add different testing scenarios, the data files can be easily extended with new data without needing to edit the actual automated test.

Creating test data for your automated tests is boring, but you should invest time and effort into creating data that is well structured. With good test data available, writing automated tests becomes a lot easier. The earlier you create good-quality data, the easier it is to extend existing automated tests along with the application's development.

6. Create Automated Tests that are Resistant to Changes in the UI

Automated tests created with scripts or keyword tests are dependent on the application under test. The user interface of the application may change between builds, especially in the early stages. These changes may affect the test results, or your automated tests may no longer work with future versions of the application.

The problem is that automated testing tools use a series of properties to identify and locate an object. Sometimes a testing tool relies on location coordinates to find the object. For instance, if the control caption or its location has changed, the automated test will no longer be able to find the object when it runs and will fail. To run the automated test successfully, you may need to replace old names with new ones in the entire project, before running the test against the new version of the application. However, if you provide unique names for your controls, it makes your automated tests resistant to these UI changes and ensures that your automated tests work without having to make changes to the test itself. This best practice also prevents the automated testing tool from relying on location coordinates to find the control, which is less stable and breaks easily.

Automated Testing with TestComplete

The best practices described in this article will help you successfully implement test automation. Our own automated testing tool – TestComplete - includes a number of features that make it easy for you to follow these best practices:

- With TestComplete you can perform different types of software testing:
    - Functional Testing
    - Unit Testing
    - Load Testing
    - Keyword-Driven Testing
    - Data-Driven Testing
    - Regression Testing
    - Distributed Testing
    - Coverage Testing
    - Object-Driven Testing
    - Web Testing
    - Manual Testing

- TestComplete allows you to divide each test into individual test parts, called test items, and organize them in a tree-like structure. It lets you repeatedly use individual tests and run them in a certain order.

- TestComplete supports keyword-driven testing. These automated tests can be easily created by inexperienced TestComplete users, and they are also a good option when a simple test needs to be created quickly.

- TestComplete supports five scripting languages that can be used for creating automated test scripts: VBScript, JScript, DelphiScript C++Script and C#Script.

- With TestComplete, QA engineers can share a test project with their team.

- TestComplete offers a **Name Mapping** feature that allows you to create unique names for processes, windows, controls and other objects. It makes your object names and tests clearer and easier to understand, as well as, independent of all object properties and less prone to errors if the UI changes. This feature allows you to test your application successfully even in the early stages of the applications life cycle when the GUI changes often.

- TestComplete provides many other features that help you get started quickly with your automated testing.

TestComplete addresses a full range of software testing challenges facing corporate IT departments, product developers, QA engineers, and consultants. The software enhances the software testing process by increasing efficiency, removing complexity and lowering costs.

Summary

Adopting the 6 recommended best practices outlined here – and taking advantage of TestComplete's many features – will help you build a solid foundation for your automated testing process that promotes high software quality. You will be able to run tests faster, test more code, improve the accuracy of your tests, and focus your testing team's attention on more important tasks that take advantage of their human capabilities.

TestComplete will help you achieve thorough Quality Assurance in development from the first line of code right through delivery and maintenance, with no surprises along the way. Ship superior applications, and ship them on time.

Download now to try TestComplete for free

You may also enjoy these other resources in the SmartBear Software Quality Series:

- [11 Best Practices for Peer Code Review](#)

- [Uniting Your Automated and Manual Test Efforts](#)

Be Smart and join our growing community of over 100,000 development, QA and IT professionals in 90 countries at (www.smartbear.com/community/resources/).

About SmartBear Software

SmartBear Software provides enterprise-class yet affordable tools for development teams that care about software quality and performance. Our collaboration, performance profiling, and testing tools help more than 100,000 developers and testers build some of the best software applications and websites in the world. Our users can be found in small businesses, Fortune 100 companies, and government agencies.

SmartBear Software
+ 1 978.236.7900

www.smartbear.com