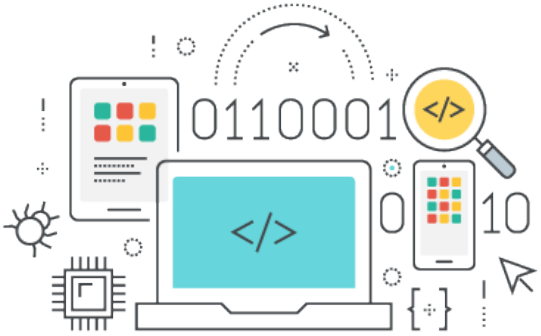
TestComplete

SMARTBEAR



Ensuring Software Success℠

Six Tips for Getting Started with Automated Testing

A SmartBear White Paper

Automated testing will shorten your development cycle, help you avoid cumbersome repetitive tasks, and improve software quality. So how do you get started? These six tips provide a successful foundation to start improving your software quality.

Introduction

1. Decide What Test Cases to Automate

It's impractical to automate all testing. The first step to successful test automation is to determine which test cases should be automated and which tests need to be conducted manually.

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, or require intuition, are better left for manual testing. Good test cases for automation are those that are run frequently, have many steps, or require large amounts of data to perform. You'll benefit the most by automating the following types of tests:

- Repetitive tests that run for multiple builds
- Tests that require multiple data sets
- Frequently-used functions that introduce high risk conditions
- Tests that run on several different hardware or software platforms and configurations
- Tests that require a significant amount of time and effort when conducted manually

A successful automated testing process requires a lot of planning and design work. Start out by creating an automation plan. This plan will help you define your goals, determine which tests or test sets should be automated, and will serve as a guide for future tests. There are a few different types of testing and each has its own place in the delivery lifecycle. For instance, unit testing is used to test small individual parts of the application to ensure those pieces are performing as expected. Load testing is conducted when an application is nearing completion and is used to determine how well your piece of software can

handle the actual demands of your end users. The goal is to demonstrate its behavior under various conditions, such as a heavy workload. To test a certain piece of the application's UI, you would use functional or graphical user interface (GUI) testing.

After determining your goals and which tests to automate, you should decide what actions your automated tests will perform. You don't want to just create steps that test various aspects of the application's behavior at one time. Large, complex automated tests are difficult to edit and debug. It's best to divide your tests into several logical and 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 have the opportunity to update your automated tests by adding new small segments that address new functions and you'll want to test the functionality of your application with each new addition, rather than waiting until the entire 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 set.

Once you create several simple automated tests, you can group your tests into one, larger automated workflow. You can even organize automated tests by the application's functional areas, user stories, modular or reusable test pieces, different applications or by a set of test data – whatever makes sense for your team.

2. Test Early and Test Often

To get the most out of your automated tests, you should start as early as possible in the development cycle and run tests whenever there is an update to the application. The earlier testers get involved in the life cycle of the project the better, and the earlier you test, the easier it is to find those bugs. 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 the right automated testing tool is essential for a successful test automation process. There are a lot of automated testing tools on the market and it is important to choose the one that best suits your overall requirements. Consider these key features when selecting an automated testing tool:

- **Support for your platforms and technology**. Are you testing .Net, Java, Angular, or Android applications? On what operating systems?
- **Flexibility for testers of all skill levels.** Can your QA department write automated test scripts from scratch or is there a need for keyword-driven testing or a record-and-replay tool?
- **Test result validation.** Does it include features for implementing checkpoints to verify values, databases, or key functions of your application?

- **Ability to create stable and scalable tests.** You'll want to be able to create automated tests that are reusable, maintainable, and resistant to changes in the application's UI. Will your automated tests break if your UI changes?

4. Divide Your Automated Testing Efforts

Usually, the complexity of different tests is based on the skill level of the QA engineers creating those tests. Therefore it's important to identify the level of experience and skill sets for each of your team members and divide your automated testing efforts accordingly. For example, writing automated test scripts requires in-depth knowledge of scripting languages or development experience. If you plan on having your team write scripts from scratch, you'll want to have QA engineers who know the scripting language provided by the automated testing tool.

Alternatively, some of your team members may not be versed in scripting languages. These QA engineers may be better at writing test cases or building codeless tests. If this is the case, you'll want your automated testing tool to have codeless testing capabilities, such as TestComplete's keyword driven testing and record-and-replay features. These will allow you to create automated tests without in-depth knowledge of any scripting languages.

Keyword tests are a simple series of keywords with a specified action. With keyword tests, you can simulate keystrokes, click buttons, select menu items, or call object methods and properties. 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 to create robust and powerful automated tests. With record and replay capabilities, you can track those simulated user actions and play them back in order to test use cases multiple times. This is particularly beneficial for regression testing and will help speed up your entire test automation process, which is crucial for any agile environment.

Different team members can contribute to automated testing in different ways. Developers and automation engineers can create scripts and unit tests, while QA testers can use a record-and-replay tool to create automated tests as well. Testing performed by a team is more effective at finding defects and the right automated testing tool should allow you to share your projects with several testers.

5. Create High Quality Test Data

Many testers will spend a significant amount of time creating tests for different data sets. This is incredibly inefficient and can slow down your testing process. Data-driven testing is a test execution strategy that alleviates this problem by allowing you to run the same test using varied data inputs, usually from an outside source such as a CSV file, Excel file, or database table. A good automated testing tool will understand the content of the data files and iterate 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 can be boring, but you should invest time and effort into creating data that is well structured. A robust automated testing tool will take it even a step further and provide a built-in data generator to help you create those complex data sets (e.g., social security numbers, e-mail addresses). With good test data, writing automated tests becomes much easier and less time-consuming. The earlier you create high-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, meaning your automated tests may no longer work with future versions of the application.

The problem is that many automated testing tools will rely on either location coordinates or object properties, such as an ID, to find an object in the UI. When screen elements shift or the object's ID changes, your automated tests will no longer be able to find the object and the tests will fail. For the test to run successfully, you would then need to spend the time updating the location coordinates or properties, such as replacing old object names with new ones. One way to avoid this challenge is to provide unique names for your screen elements. This will make your automated tests resistant to the UI changes and ensure that your automated tests work without having to make underlying changes. This will also prevent your automated testing tool from relying on location coordinates to find the object, which is a less stable and easily breakable method.

The best way to overcome this challenge is to find an automated testing tool that can handle these types of changes with minimal maintenance. Comprehensive automated testing tools will provide a centralized object repository - making object updates easy and ensuring those updates will work across all scripts, tests, and test suites.

7. Automated Testing with TestComplete

The best practices described in this article will help you successfully implement an automated testing process. Our own automated testing tool, TestComplete, has several out-of-the-box features that make it easy for you to follow these best practices and will help you 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
- Mobile Testing
- Cross-Browser Testing

- TestComplete allows you to divide each test into individual test parts, call test items, and organize them in a tree-like structure. The tool lets you repeatedly use individual tests and run them in a certain order.
- Go beyond record and replay with TestComplete. Non-programmers can create complex testing scenarios quickly and easily without scripting by dragging and dropping built-in actions for conditional logic, setting up delays, or logging test results.
- TestComplete also supports multiple scripting languages that can be used for creating automated test scripts, including JavaScript, Python and VBScript.
- With integrations to test management platforms like QAComplete, CI systems like Git or Mercurial, and IDEs like Visual Studio, TestComplete allows QA engineers to share their test projects with their team and developers.
- TestComplete has a powerful object identification engine that can accurately recognize over 50,000 object properties across more than 500 control types, including all major frameworks and third-party control sets. This feature allows you to build more stable UI tests and will help you find the right controls quickly and easily.
- Environment Manager, TestComplete's on-demand cloud lab, will give you real-time access to the latest devices, resolutions, browsers and operating systems. With this feature, you can run and report on tests across thousands of remote environments to improve test speed and coverage - all without setup or configuration.
- TestComplete addresses a full range of software testing challenges facing corporate IT departments, product developers, QA engineers, and consultants. The tool enhances the software testing process by increasing efficiency, reducing complexity, and lowering costs.

Conclusion

Adopting the top six 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-quality software. 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. TestComplete will help you achieve thorough quality assurance in development, from the first line of code all the way through delivery and maintenance, with no surprises along the way. Ship superior applications and ship them on time with TestComplete.



TestComplete

Download your free trial of TestComplete today and see why testers choose SmartBear for automated UI testing

TRY TESTCOMPLETE FREE

FOR 30 DAYS

About SmartBear Software

Supporting more than five million software professionals and over 20,000 companies in 194 countries, SmartBear is the leader in software quality tools for teams. The company's products help deliver the highest quality and best performing software possible while helping teams ship code at nearly impossible velocities. With products for API testing, UI testing, code review and performance monitoring across mobile, web and desktop applications, SmartBear equips every development, testing and operations team member with the tools to ensure quality at every stage of the software cycle.

For more information, visit: http://smartbear.com, or for the SmartBear community, go to: Facebook, Twitter, LinkedIn.

SMARTBEAR

SmartBear Software, Inc.
450 Artisan Way
Somerville, MA 02145