

Automated Testing Best Practices

This document includes best practices to consider before implementing automated software testing. These best practices are strategic and are applicable regardless of the automation tool you use.

Preparing for Automation

Set realistic expectations

It is impossible to automate all testing, so it is important to determine what tests will produce the most benefit. The point of automation is not to eliminate testers but to make better use of their time. Tests that require large amounts of data to be input and tests that are run frequently, such as regression tests, are good candidates for automation. Tests that require human intervention are not. Any test that has predictable results and meets one of the following criteria is a candidate for automation:

- The test is repetitive (for example, choose every item on a Web page or repeat the test for each release).
- The test evaluates high risk conditions.
- The test is impossible or costly to perform manually.
- The test requires multiple data values to perform the same action (i.e., a data-driven test).
- The test is a baseline test run on several different configurations.

Automating the repetitive and boring tasks frees up testers to perform more interesting and in-depth tests.

Keep an eye out for tests or areas of an application that are frequently modified. These are barriers to automation. If there is an area of your Web site that changes daily, it will be difficult to define exactly what constitutes a passing result.

Prepare your applications for automated testing

Automation tools use a series of properties to identify and locate controls. For example, when searching for a button, the script might use the button label to locate the object. If the label changes from OK to Submit, scripts will no longer be able to find the button when they run and will fail. Developers should immediately notify testers of any application changes so the necessary updates can be made to scripts.

If possible, have developers create unique names for controls. Providing a unique name makes it easier for both the script and the tester to identify controls. This reduces the chance that the automation tool will have to rely on location coordinates to find the control.

Automation tools can reveal duplicate control names and non-standard development practices.

Identify the correct search criteria

If the controls do not have a unique identifier, consult with development to determine a set of control properties that you can combine to create solid search criteria. For example, combining the control type and location coordinates can help distinguish between two combo boxes with similar properties.

Keep in mind that some applications have dynamic properties. For example, a Web page can have dynamic hyperlinks that include a session ID. Since that ID changes each time the site is accessed, the hyperlink reference does not make effective search criteria and will eventually cause the script to fail.

Define a standard display resolution

In some cases, the test tool will need to use relative coordinates to locate controls. All mouse actions also rely on coordinates, so changing the display setting will affect script execution.

Use a standard display setting. If testing requires more than one display setting, create display-specific subroutines that can be conditionally called to perform mouse actions.

Prepare Your Automation Team and Processes

Divide your efforts

Identify the skills of your team members. Some team members are better at writing test cases than writing scripts. Have a tester with a programming background create any subroutines and

functions needed to perform specific actions. Then document the functions and how to use them so other testers can easily include them in scripts.

Create an automation plan

To decide which tests to automate, it is helpful to write an automation plan. Begin with your goal for automated testing. Following are examples of possible goals:

- Speed up testing to allow for accelerated releases
- Allow testing to occur more frequently
- Improve test coverage
- Ensure consistency
- Improve the reliability of testing
- Define the testing process

Document the types of tests or processes to automate. This could include the following:

- Regression testing – Tests that are run for every build
- Data-driven testing – Tests that require multiple sets of data over many iterations of the test
- Configuration tests – Test that run on many different platforms
- Test bed setup – Setup/configurations required on a group of test machines before testing can begin
- Backend/Non-GUI testing – Tests that verify the data collected by the application

Your automation plan allows you to identify the initial set of tests to automate, and also serves as a guide for future tests.

Define measurements for success

Your automated testing goal should not be automation for automation's sake. In the automation plan, define how you will tell if your test automation is effective. Following are potential measures for successful automation:

- Same number of features are tested in less time
- More features are tested in the same time
- Current testing costs are less than previous testing costs

Develop test cases

Creating and debugging automated scripts is time consuming. Therefore, you should avoid ad-hoc tests and tests that are only performed a few times. Automated tests should be repeatable and should have clear results. After you determine the types of tests to automate, write test cases that clearly document the goal of the test, the prerequisites, the expected outcomes, and what determines a failure. Ensure that each test has a specific purpose and identifiable results. It is helpful to begin by automating previously documented, manual test cases.

When developing tests identify checkpoints you can use to verify test results. Ask yourself what changes in the application to indicate a successful test. The system might update a database or present a verification number. Use a spreadsheet or databases to verify the expected changes.

Choose the right tool

When reviewing automation tools, make sure the tool supports your application's technology. Is the tool continually updated with latest technologies? Automation tools can be difficult to learn so do not underestimate the value of training. Is training available for the tool?

Analyze your team's strengths and determine if the tool is a good match for your team's skill set. Does the tool require users to be programmers or does it offer flexibility for testers of different skill levels?

Implementing Automated Testing

Document automated tests

In addition to writing an automation plan, create an automation testing standards document. Decide on how to organize the scripts after they are recorded and define a structure for the scripts.

Before recording any scripts, establish naming conventions for scripts and variables. Also, create a separate document that lists all automated tests and the purpose for each test. Use comments throughout scripts to explain the logic of the test.

Create reusable and maintainable automated tests

It is best to create small scripts then combine them to create more complex tests. Long, complex scripts are difficult to debug and update. When creating scripts, strive to keep them reusable,

understandable, maintainable, and modular. You can use the acronym RUMM to remember the characteristics of an effective script. The following describes how to achieve each of the characteristics.

Reusable – Keep scripts small and focused on a single task. For example, if you are testing the functionality for setting user options, you can create one script that opens the user options and one that closes it. This allows you to use these steps repeatedly without including them in every script that accesses the user options window. You can also use variables and external data to make scripts reusable. When values for input fields are not hard coded in the script, you can use the same script to test for a variety of different conditions.

Understandable – Maintaining a document that contains detailed descriptions your scripts' functionality and purpose makes it easy to understand what types of scripts are available and how to combine them into larger tests. In addition to a list of scripts, comments in the individual scripts can go a long way to making them easy to use. As a general rule, add comments to variables, functions, subroutines, and branching structures so it is clear how the individual components work.

Maintainable – Using external data not only makes scripts reusable, it also makes them easy to maintain. To add different test scenarios, you can simply add more data to a spreadsheet or database without needing to edit the script. Also, using an automation tool that includes a text-based scripting interface or allows you to edit a script in a text editor can reduce maintenance time.

Modular – Creating scripts to perform specific tasks gives you the flexibility to build complex tests from small scripts that are

easy to troubleshoot. If your code base changes, it is easier to add small, modular scripts that address the new functionality. You can organize scripts by the application interface, major/minor division in application, or common functions. For example, when testing an email client, you might have small scripts that perform the following actions:

- Open application
- Log in
- Create an email message
- Select a recipient
- Send the email message
- Go to user options
- Edit user options
- Save user options
- Close user options
- Log out
- Close application

Conclusion

These best practices form the basis for effective test automation. Implementing them can help you avoid common mistakes and improve your testing process regardless of the automation tool you use.

Quality-Centric ALM Solutions

Designed for the most demanding software development environments, Seapine's application lifecycle management (ALM) solutions are scalable, feature-rich, team-based tools that can be used separately for superior issue tracking, software configuration management, automated software testing, or test case management—or seamlessly integrated for more efficient control of the application development process.



TestTrack Pro

Development Workflow and Issue Management

Tracking defects, issues, and feature requests is a critical component of any software development and quality control process. The earlier and quicker bugs are resolved, the lower your development cost and the higher your product quality. TestTrack Pro puts improved quality, communication, and reporting within reach. Use TestTrack Pro and create better software in less time.



TestTrack TCM

Test Case Planning and Tracking

Testing software applications requires hundreds to thousands of unique test cases, the time to execute them, and the ability to efficiently manage the results. At the same time, software applications are becoming increasingly more complex and development schedules more aggressive, but quality cannot be compromised. TestTrack TCM manages all facets of the software testing process including test case creation, scheduling, execution, measurement and reporting. With TestTrack TCM, you will test more in less time, while seeing a measurable improvement in product quality.



Surround SCM

Software Configuration Management

Measurable, enforceable, and repeatable software configuration management and version control techniques are critical to delivering quality software products on time. Surround SCM with change automation, custom metadata, and virtual branching delivers complete control over the software change process, a clearer view of the state of files, and streamlines parallel development.



QA Wizard Pro

Automated Functional and Regression Testing

Automated testing is a critical part of developing and deploying quality software applications. QA Wizard Pro automates the functional and regression testing of Web and Windows applications, helping your quality assurance team test more of an application in less time.



TestTrack Studio

Software Test Planning and Tracking

From the creation of test cases through the resolution of defects, TestTrack Studio tracks and manages all the details of your testing effort. TestTrack Studio seamlessly blends the award-winning defect tracking features of TestTrack Pro and the time saving test case management features of TestTrack TCM into an integrated test environment. In TestTrack Studio, your QA managers, product managers, developers, and testers have a collaboration solution to help your company deliver higher quality products, faster.



Seapine CM

Complete Software Change and Issue Management

Seapine CM brings together comprehensive configuration management and flexible issue tracking to offer a complete change management solution. Featuring a seamless two-way integration between our advanced configuration management tool, Surround SCM, and our award-winning issue tracking tool, TestTrack Pro, Seapine CM allows users to access bugs, feature requests, change requests and source code files and digital assets from within either tool. Work smarter not harder with Seapine CM.