QualityLogic

WHITE PAPER

The Foundations of Successful Test Automation

Automating the test process applies computational power to what it does best, the repetitive exercise of software systems. And yet, anyone who has been in the software testing field for more than a few years has seen a test automation effort fail.

Learn what you need to know to make your test automation effort a success.

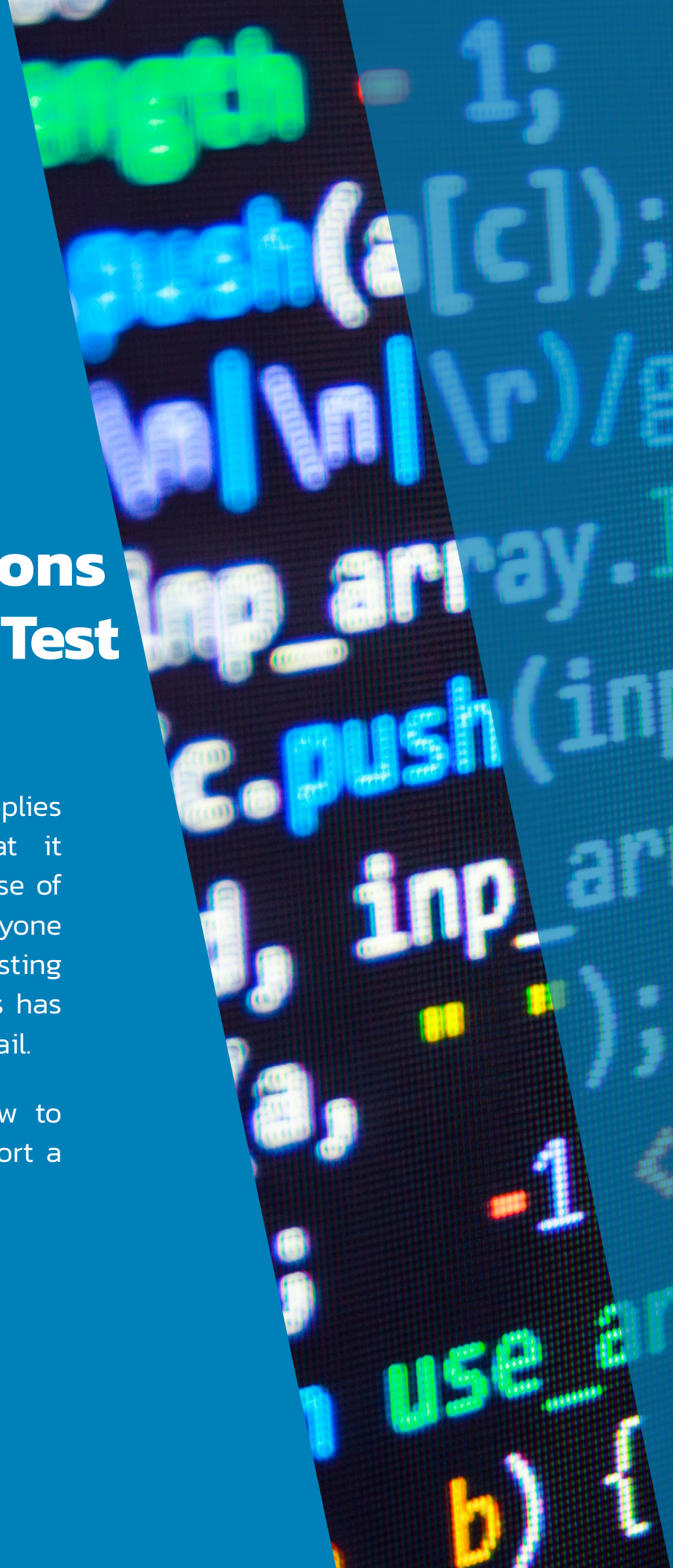


Table of Contents

Overview

Software development has progressed from the Waterfall method that allowed months and even years for creating, assembling, and testing a software release to the Agile and Continuous Integration methodologies that crank out releases in a week or two. As a result, test automation has become an integral component of software development.

In the software marketplace, development efforts must get it right the first time. The modern user community has very little tolerance for content format issues or slow performance, much less functional failures. The immediacy of market requirements that are desperately trying to distinguish a product from dozens of competitors makes little space for a careful measured approach to testing. The rapid release of features and defect fixes in these software products and services means that test verification has had to change dramatically to keep up.

The Choice to Automate

Automating the test process applies computational power to what it does best, the repetitive exercise of software systems. And yet, anyone who has been in the software testing field for more than a few years has seen a test automation effort fail. The project starts out with great optimism that the selected automation tool will work wonders and in short order all the tests will run with the push of a button. However, the tools don't quite work as advertised, the automation is a bit trickier than expected, and everything about automation takes longer and costs substantially more than estimated.

Then the automated tests start generating false failures due to changes in the system. This causes diversion of test automation team members to help fight these false failure issues, further slowing progress. Next, a few test automation team members take their acquired experience to other companies and maintaining their code becomes much more difficult. Management starts to get disillusioned at the slow progress. Developers get tired of the false failures and want additional proof before they will work on bugs caught with automation and the whole thing spirals downward from there.

Despite its obvious benefits, the horror stories of failed automation attempts and massive investments in a technology that didn't provide a positive ROI have left many companies on the fence about test automation. So, what are the considerations for starting a successful test automation initiative?

Developing a Test Automation Strategy

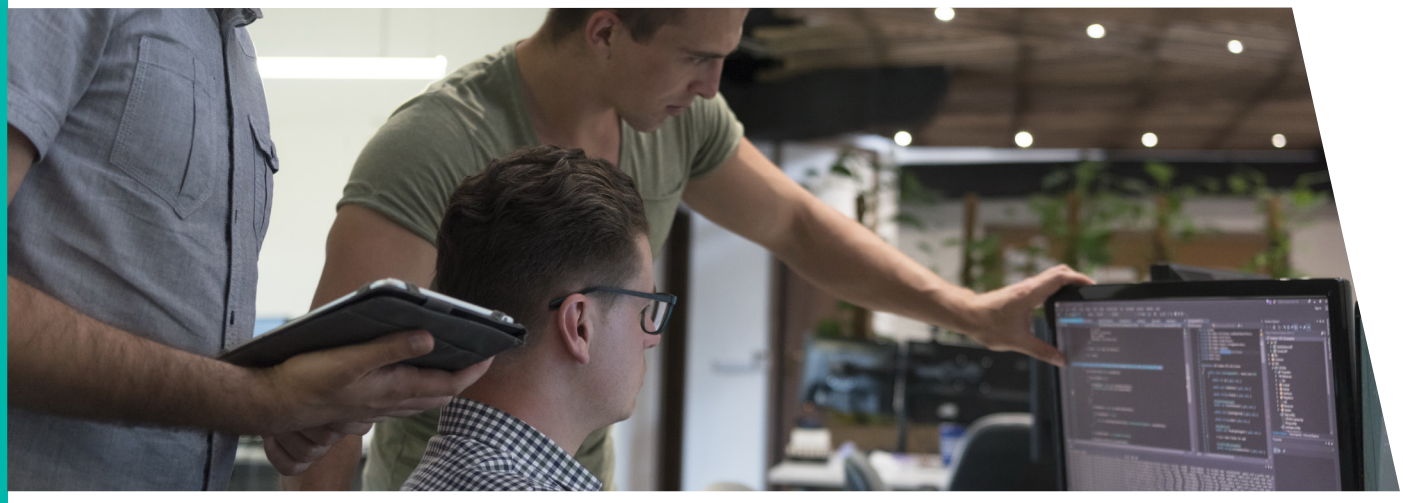
It is critical to develop a test automation strategy that meets your business needs. Test automation can generate a huge ROI, but it takes time. Management must be committed to the effort and their expectations must be carefully calibrated against a well-thought-out strategy. Automation is not a closed ended project, but rather a fundamental change in how the testing aspect of software development is performed.

Test case parsing, automation tool selection, staffing and automation priorities must all be considered in a unified approach to creating as solid foundation for the automation effort. Test automation is no less a programming project than is production code development. The initial sticker shock of test automation can easily dim management's prospects for an actual ROI making strategic plans for implementation absolutely necessary.

Cultivate a Test Automation Culture

Using test automation during the software development life-cycle provides quick feedback to the team after a new feature has been developed. It is a strategy that is especially beneficial for tests that don't change frequently and are executed repeatedly on every new release.

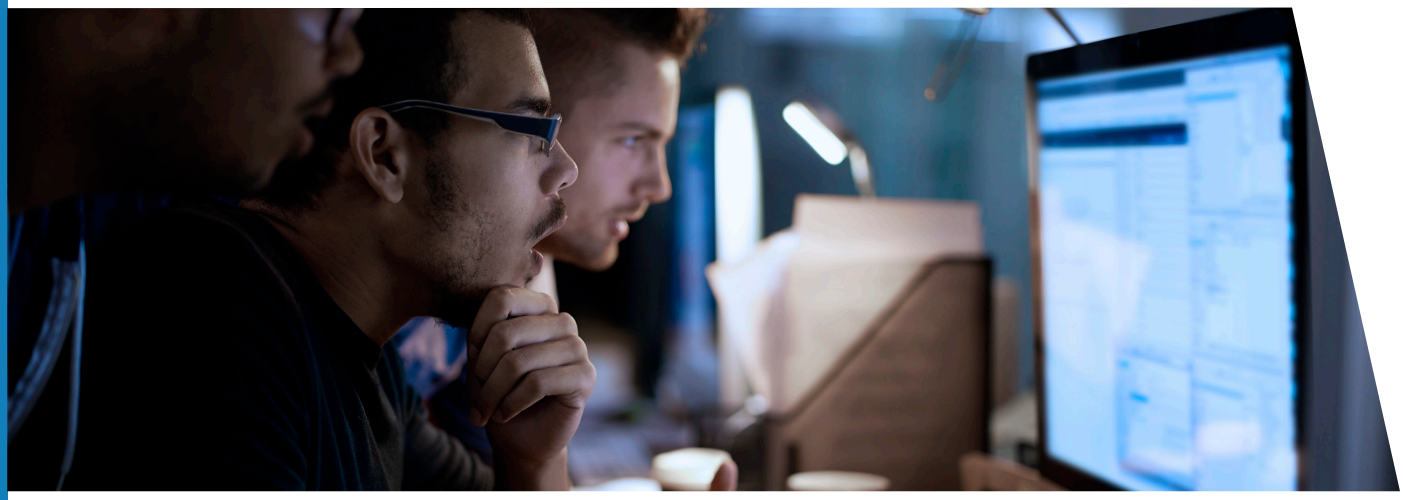
Test automation isn't so much about creating more available time in the test cycle, but instead being more efficient and productive with the time spent testing. This is best supported with a mind-set of looking to see where



automation makes sense and avoiding trying to make it a one size fits all. By automating repetitive tests, you can reallocate that time to running exploratory tests to verify that your application is still in good order as changes are being delivered. The downside of automating those repetitive tests is that many organizations fall into a "set-it-and-forget-it" mentality. Automated test script maintenance is one of the most crucial aspects of test automation. As the development of your product progresses, so must your automation code.

Test automation doesn't eliminate the need for human interaction. If anything, it requires more collaboration if the scripts are to be comprehensive and effective. Writing scripts can't happen in a bubble. The most effective scripts result from an agile collaboration between the business analysts, developers, and QA team. This sharing of ideas is what ultimately leads to a comprehensive suite of scripts that reflect the expanse of system functionality.

Scripts will need to be revised as requirements change. New cases will need to be developed as new features arise. Existing test cases may need to be deactivated or thoroughly rewritten to accommodate system changes. All of this requires a feedback loop between every member of the agile team to keep everything in sync.



How to Prepare for the Test Automation Effort

There are many good reasons why companies want to start automated testing. There are potentially large costs saving over manual testing. Release cycles are getting shorter, so testing must be accelerated. Perhaps the most compelling reason is that test automation improves product quality by catching bugs earlier in the process. A set of manual tests may take a week to run, the same tests automated could be run once or twice a day.

The triaging and maintenance of automated tests is where most organizations tend to trip up. They don't build that time into their initial test automation strategy and the automation project dies as a result.

This is one place where a 3rd-party testing service can be of great use. QualityLogic's services are regularly utilized to review test failures as well as debug and update test automation code. We are also extremely adept at helping companies develop their test automation strategy and writing automated test scripts. Utilizing an outsourced software testing company to ensure the success of your automation project allows your company to focus on developing revenue-generating code for new features and critical updates.

There are two primary types of tests that are typically automated. Unit tests that focus on individual source code methods and functional tests

that ensure all aspects of the software program are working correctly. The task of unit test automation is usually the domain of the developer, while functional test automation typically rests with the software test team.

A good starting point for test automation is to create a sanity check that serves as a touch on each major system feature to verify that it is in place and elementally functional. Note that a sanity check does not test a feature in depth. Its purpose is to assure that some change in the system, be it a new feature or a defect fix, has not damaged a supposedly unrelated portion of the system code.

Test automation is an investment in increasing test throughput and should be viewed as such. Its main payoffs are its ability to increase test case execution, facilitate high repetition, and detailed regression testing.

How to Optimize and Scale Your Automation Project

Automated and manual testing go hand-in-hand. Neither is inherently better than the other. Both strategies complement each other in ways that allow for the most comprehensive and effective test strategy. That said, you will need to employ both at some point in your development cycles. The key is knowing when and how to implement each.

As mentioned previously, continuous integration environments require a testing approach that can keep pace. Test automation makes it possible

to repeat an entire test suite as often as needed. Not only that, automated scripts can simulate multiple concurrent users to evaluate the system's performance.

Data-driven functions are another prime area for automation. There may be cases where the same tasks need to be validated with different inputs. Doing so manually is time-consuming and a waste of QA resources. Automating these types of tasks ensures that all possible input scenarios are accurately covered in depth.

Lastly, static and repetitive tasks are ideal for automated scripts. Why waste time having someone manually verify things that remain relatively unchanged from one cycle to the next? Letting automated test scripts handle these tasks frees your team to work on more important items.

That said, there are scenarios where human observation is more appropriate than an automated script. For example, when the goal is testing usability, a manual approach is best. Humans learn more about user perspective during their evaluation. They can then use this knowledge to make recommendations on how to improve the user experience. Also, people are good at catching things the system missed. It is not uncommon for someone to find things that were never addressed as a part of the automated scripts. Lastly, and perhaps most importantly, there is no substitute for the analytical observation skills required to evaluate complex systems. It may not be possible to evaluate certain features using automation. In those cases, a manual approach is necessary.

In the end, the real question isn't which is better than the other. The real question is, how can you capitalize on both to get the most effective results.

Coding Best Practices for Test Automation

Showing automation results quickly is a huge confidence builder for both the automation team and management. A result of this is that code developers are pressed into automation script development at the outset of most automation projects.

They will bring their coding practices with them and good production code shares its foundations with good automation script code.

Coding best practices for test automation include templates that guide the test script developers' efforts. Key elements of an effective set of best practices include:

- Test case naming
- Object location strategies (use more than one)
- Hard or soft asserts
- Wait handling
- Page object pattern usage
- Data driven test inputs

- Minimal dependencies with other test cases

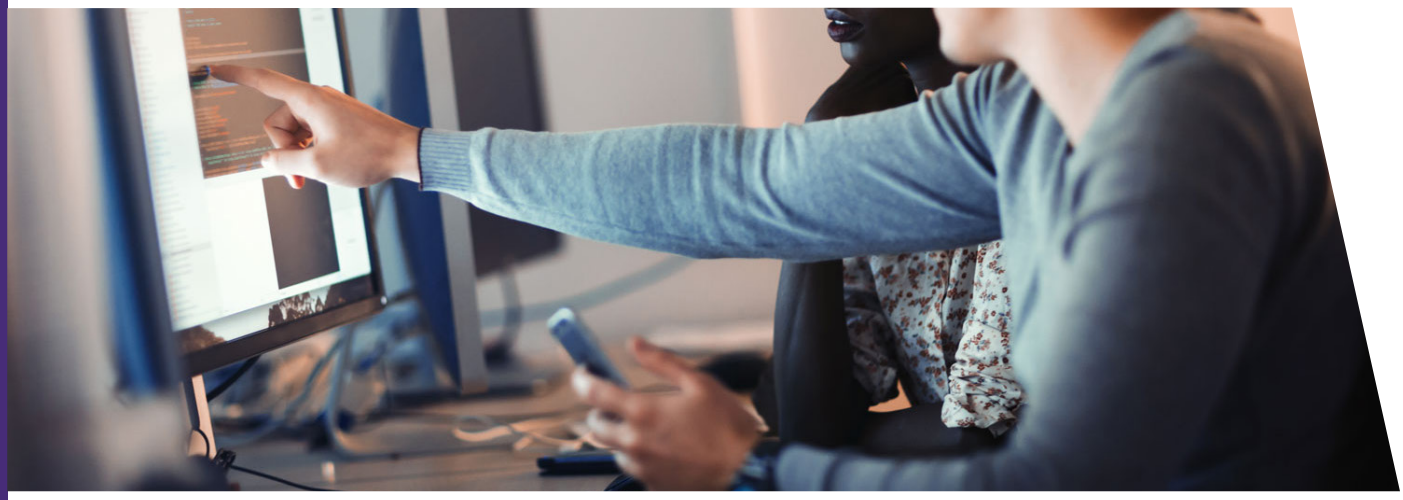
At QualityLogic we use AI-based similarity analysis of manual test cases to identify opportunities to build libraries for common user interactions and to identify the order in which to approach coding of test cases to maximize development progress. The techniques can be grouped into the following approaches:

- Isolation of globally common test sequences whose functionality can be automated as part of a common code library
- Clustering similar test cases for assignment to the same programmer
- Predictive ordering of test cases for development to maximize code sharing between similar test cases

Best Practices for Automated Test Creation and Maintenance

Automated tests should focus on clear and narrow objectives, typically replicating a specific typical user action. Tests with a narrower scope are easier to code, easier to debug, and easier to maintain. Remember, test automation is software development. Robust source code management is a must using platforms like GitHub.

Manual test operations tend to rapidly cull test cases due to the continuous review they receive from testers. But automated tests tend to raft up like old clothes stuffed into the back of a large closet. The relatively recent advent



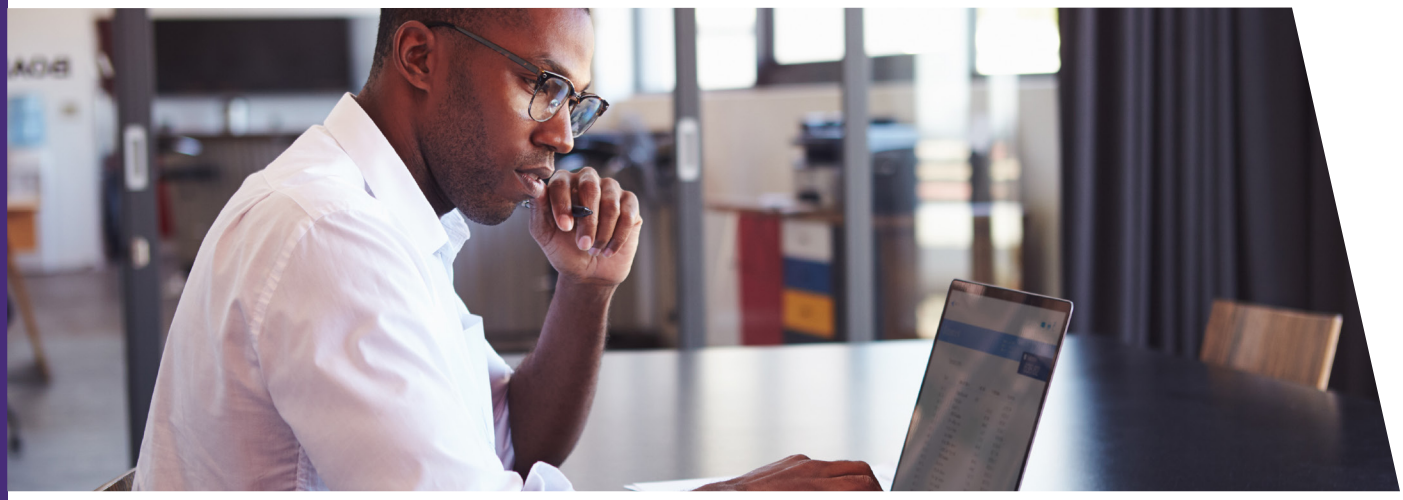
of test automation tools that generate test scripts by recording processes and GUI activity has aggravated the issue of test script management. Advertised as engines for quick, accurate test script generation, the code produced by these tools usually must be altered before useful testing can occur.

This tends to consume exactly the maintenance resources they are intended to free up. If used extensively, test recorders can cause a great many marginally useful and very inflexible test scripts to be amassed in a very short period.

Take the output of an auto-generating test tool, combine it with the test scripts created specifically by code developers, then add in tests written by QA engineers to plug the gaps and the body of test scripts can get out of hand quickly. Now salt heavily with functional changes brought about by defect fixes with undesired feature interactions and you have a test array that rapidly goes from unmanageable to unusable.

Living with false failures is a fatal mistake for automation efforts. If a test is flakey, pull it out of the daily automation runs until it is fixed. If development is not going to fix certain bugs, pull the test cases that trigger those errors from daily runs. You must preserve an environment where an automation run failure is a red flag to everyone on the team. False failures or failures that are ignored poison that assumption.

Test creation should be predicated on clear criteria for why the test is



necessary and when it is to be implemented. At the other end of the process, formal test retirement criteria should govern a regular audit of the existing tests. With the pressures to create them, test retirement becomes critical. A firmly grounded retirement plan keeps test suites from growing out of control.

Regression tests are another source of rapid test suite growth. Retirement of obsolete scripts will help keep regression suites under control. Understand that full regression testing and the data sets it entails may simply take longer than the development schedules permit. “Regression test everything” sounds impressive but it doesn’t scale and can undermine your automated testing effort.

A test automation framework should be put in place. This is more than just the selected automation tool. It includes common resources and libraries that test cases leverage; integration with build processes; issue trackers and other parts of the software development infrastructure. This allows the test developer to focus on coding test cases knowing the framework will take care of the execution and reporting details.

Choosing the Right Test Automation Tools

There are a wide variety of test automation tools available with varying degrees of effectiveness and requiring differing skills. The first order selection criteria are whether to go with a commercial tool offering or

open source tools. Commercial test automation tools from big players like Tricentis are robust, but very expensive. Commercial tools tend to be relatively easy to use, simplify the test creation process, have training and support for their products, and tend to be less buggy than their Open Source counterparts.

Open Source automation tools are free, have supportive user groups, and some, like Selenium, have become de facto standard test tools. Multiple Open Source tools may need to be used in concert for a given automation solution and integration can be challenging. Customer motivations vary, but few like the idea of being locked into a single vendor for a critical part of their development infrastructure.

Automation tools can be roughly categorized in to the following general areas:

Dumb Record & Playback Test Tools

A brute force recording of user interactions with the application. Test scripts are very fragile and break with the slightest change in the application.

Smart Record & Playback Test Tools

More adaptive recording of user interactions with the application, storing multiple object identifiers and leveraging machine learning. They can adapt in a limited fashion to application changes without breaking the test script.

AI Assisted Auto Discovery & Playback

Self-discovery of paths through the application using reinforcement

learning, with the ability to playback any of the discovered paths. Able to adapt in a limited fashion to application changes without breaking the test script.

Abstract Syntax Test Tools

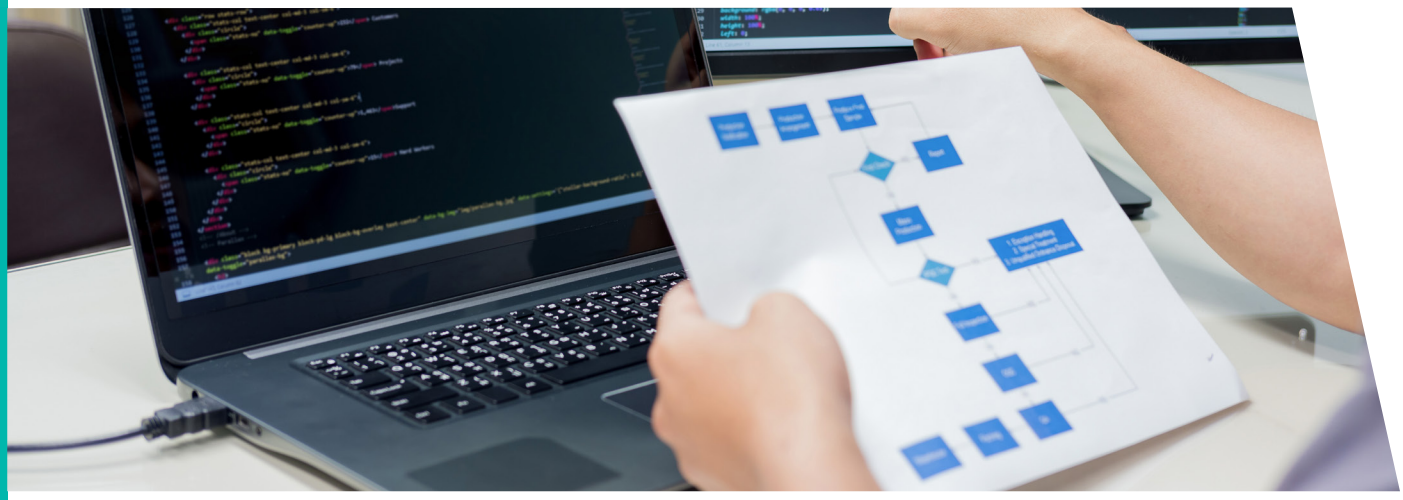
Use of natural language, keywords, or procedural text (think Cucumber/Gherkin) to define test cases, with the underlying automation code driven by the abstract test definitions. In some tools the automation code triggered by the abstract test definitions must be hand coded while other tools have some helper routines deal with more common scenarios that can be inferred from the application objects.

Hand Coded Test Development

Use of common programming languages to define automated test cases using an underlying automation API such as those supported by Selenium, Appium, or mobile device operating systems.

While there is a lot of excitement around the AI enabled testing tools, particularly those that can auto-generate test scripts, these tools work best with applications whose application logic is relatively simple. In our experience, applications with more complex application logic require hand coding of automated test cases to fully test that application logic.

Programming skills are needed for most test automation efforts. Senior test developers can code reference tests for various test classes, then more junior staff can use those reference tests as a guide for derivative tests.



Most organizations select a specific language for test case development and it is prudent to have potential test developers demonstrate their skills in the selected language before being added to the team.

Industry-wide, Java and JavaScript are the most popular for test automation projects, however QualityLogic's customers have been more frequently using C# or Python. In theory, Python is a friendlier language for more junior developers which may be a consideration when selecting a language.

Quantifying Your Automation ROI

Quality has been moved onto the front lines. Testers and code developers are expected to work in tandem fixing bugs as quickly as they are found. The Agile SCRUM has stand-up meetings daily in which the 'stories' selected for the current two-week sprint are discussed in terms of the day's progress in both code development and operational verification. Once a fix or feature is integrated into the system, it is expected to be verified and released in a day or two at the most.

This pushes hard against the reality that failing to perform adequate testing costs more than the initial investment. A study by the National Institute of Standards and Technology (NSIT) found that software defects cost the economy nearly $60 billion annually.

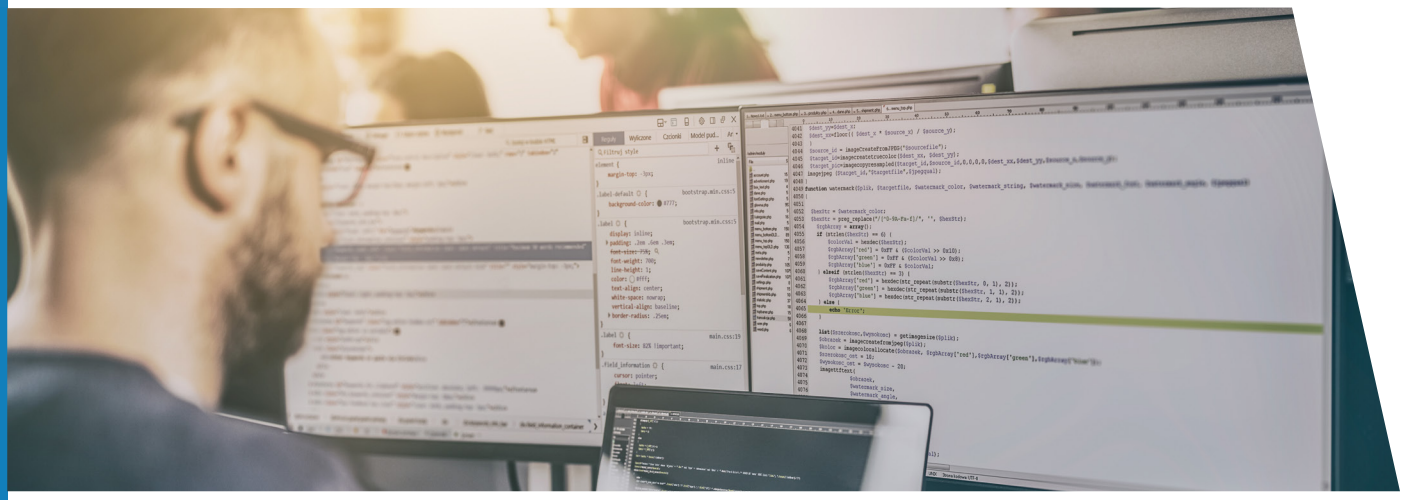
Think you're exempt? Think again. Imagine, just one post-deployment issue

could leave an entire function of your system out of commission costing you money and customers. With that said, test automation is about much more than speed to market. It's about catching bugs pre-deployment when they are less expensive to fix.

Quantitative measurement of test automation results is a problematic proposition. You need to be able to numerically analyze which defects didn't show up in the released product. One step to doing this is to tag defect reports with the specific QA phase in which they were discovered. This will produce a readily parsed data array of which defects were caught by automation and whether the effectiveness trend is increasing for the number of these detections.

One of the most telling aspects of defect tracking is the escape rate. Microsoft coined this term to label those defects that passed all the way through the QA process and were found only in the released product. These are obviously the most damaging bugs and improving automation implementation should measurably drive this number down.

Aside from the cost benefits, there are several intangible benefits that more than justify the cost. Automating repetitive tests frees up time that can allow a team to more deeply reflect on the needs and wants of the customer.



Helping You Achieve Success in Your Test Automation Initiative

Production management is caught between two fires in that they need skilled engineers to crank out the code that fulfills their marketing needs, and that code must be fully tested before it goes out the door. All of which leaves management with the unenviable task of parsing out scarce, expensive resources, that they never have enough of, to project leads who begin to regard each other as competitors rather than colleagues.

QualityLogic has recognized this problem from many years of test project work with clients who must deal with it constantly. We offer skilled test script development engineers who have that keen perspective on quality that translates into excellent coverage on a fixed commitment basis. We work across an array of different test systems, both open source and proprietary, developing test specifications and their attendant scripts.

We maintain and update test scripts to keep them current, and we guarantee the quality of the test scripts and their interdependence in systems or re-work them at no cost to the client. We can create scripts on existing customer platforms, or we can create an entire test automation facility from scratch, delivering a working platform and training in-house resources on its use and maintenance.

Summary

We realize that not all projects are created equal. Our team will help you identify aspects of your system that lend themselves to automation and which do not. Based on that, we will devise a strategy and determine a framework that best suits your needs. With this approach, you won't waste time and money on a ton of bulky features that you'll never use. Our unique approach to test automation uses the latest in virtualization and cloud computing. As a result, we can scale our testing environment based on your needs limiting the impact to your project schedule and budget.

There is finally a solution to the shortage of skilled engineers that doesn't rob development of badly needed resources to create test scripts. QualityLogic brings those skills to your project at affordable rates so that you can create the automation platform that assures the high quality of each end user's experience.

For More Information

Visit www.QualityLogic.com or call +1 208-424-1905