WHITE PAPER

HOW TO GET THE MOST OUT OF YOUR CI/CD WORKFLOW USING AUTOMATED TESTING

This paper is aimed at Test and QA Executives as well as Project Managers who are considering adopting automated testing, but are unsure of how to get started. It highlights the benefits of automated testing, the recommended technical approach to take, and suggests tools that enable teams to successfully adopt automated testing as part of a healthy continuous integration and delivery process. It also examines which tests to automate and which to continue to do manually.

SAUCELABS

TABLE OF CONTENTS

EXECUTIVE SUMMARY

In today's hyper-competitive cloud economy, it's important to be first to market to gain a competitive edge. This makes organizations prefer agile development techniques like continuous integration and continuous delivery (CI and CD) to the traditional waterfall approach to building software.

Automated testing is an integral part of the continuous delivery pipeline. However, despite the acknowledged benefits of automated testing, ground reality is that most organizations still use outdated manual testing processes. The initial effort required to setup an automated testing process makes teams want to avoid the pain, and make do with manual testing. However, to benefit from automated testing, it's important to use the right tool, the right framework, and the right technical approach.

The right approach involves knowing which tests to automate, and which to continue manually. Having a testing framework enables the process to be scaled to larger projects, and better cope with changes along the way. When selecting the right automated testing tool, organizations are faced with three options - build one in-house, leverage an open source tool, or buy a commercial tool. Among open source frameworks, Selenium and Appium have emerged as the ideal way to automate testing for web apps and mobile apps respectively. However, they can be resource intensive to setup and maintain in-house. Thus, the ideal automated testing tool should be based on Selenium and Appium, but avoid the pain of manual maintenance.

AUTOMATED TESTING AS PART OF THE BROADER CI/CD PIPELINE

In their groundbreaking book, 'Continuous Delivery', Jez Humble and David Farley begin by declaring that "The most important problem that we face as software professionals is this: If somebody thinks of a good idea, how do we deliver it to users as quickly as possible?"

Realizing that the traditional waterfall technique of software development is inadequate to meet this goal, organizations are slowly but surely adopting agile development techniques like continuous integration and continuous delivery.

- Definition of Continuous Integration (CI): "A development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early." - ThoughtWorks

- Definition of Continuous Delivery (CD): "The practice of releasing every good build to users" - Jez Humble

Humble and Farley define what a typical deployment pipeline looks like using the following illustration:

They describe this pipeline as "an automated implementation of your application's build, deploy, test, and release process." They go on to recommend that as much of this pipeline should be automated as possible.

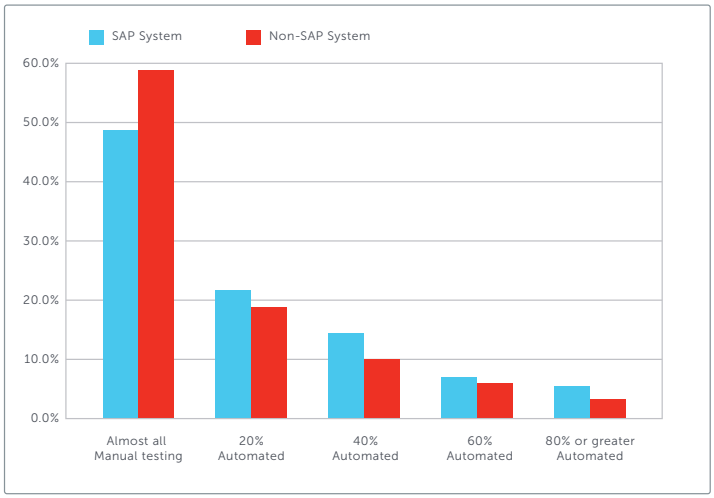
Commenting on this pipeline, renowned expert and ThoughtWorks Chief Scientist, Martin Fowler says, "A deployment pipeline breaks up your build into stages. Each stage provides increasing confidence, usually at the cost of extra time. Early stages can find most problems yielding faster feedback, while later stages provide slower and more thorough probing. Deployment pipelines are a central part of Continuous Delivery."

As seen from the pipeline, automated testing is a vital component of the continuous delivery process.

REALITY CHECK - THE MAJORITY OF TESTING IS STILL MANUAL

Automated testing is an integral part of the broader CI/CD methodology. But surprisingly, most organizations still do the bulk of their testing manually. In one survey by XBOsoft, an automated testing consultancy, only 28% of the respondents automate more than 50% of their test cases.

In another survey by WorkSoft, an SAP partner, 60% of survey respondents, mostly SAP customers, noted that their testing was almost fully manual.



WHAT IS AUTOMATED TESTING?

Manual testing involves a human tester using a computer or mobile device, trying various usage and input combinations, comparing the results to the expected behavior and recording their observations. Automated testing, on the other hand, uses tools to execute pre-scripted tests on a web or mobile app, and records the test data in detailed reports that can be reviewed by a human tester after the tests are run.

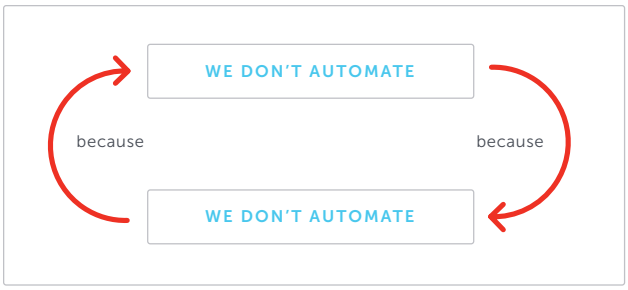
- Definition of automated testing: "It is the use of special software (separate from the software being tested) to control the execution of tests and the comparison of actual outcomes with predicted outcomes." - Wikipedia

MANUAL TESTING VS AUTOMATED TESTING - WEIGHING THE BENEFITS

CRITERIA	MANUAL TEST	AUTOMATED TESTING
Faster time to market		✓
Ideal for cross-platform tests		✓
Accurate		✓
Performs complex tests		✓
Increases frequency of feedback		✓
Supports agile methodology		✓
Frees up testers for their best work		✓
Scales to larger projects		✓
Is repeatable		✓
Enables after-hours testing		✓
Improves documentation & traceability		✓
Cost effective		✓
Improves over time		✓
Ideal for exploratory tests	✓	
Ideal for one-off tests	✓	
Easy to adopt	✓	
Easy to maintain after adoption		✓
Allows (massive) parallelization		✓

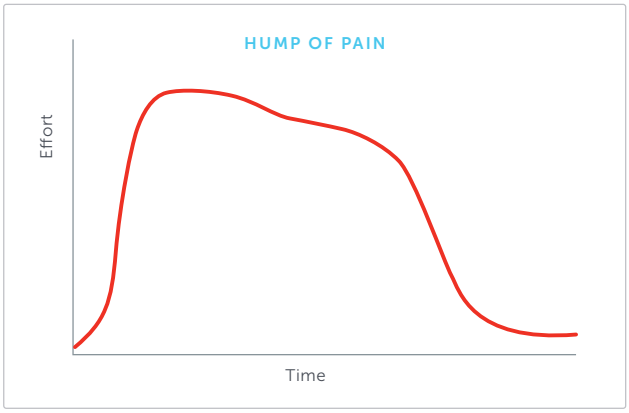
OBSTACLES TO ADOPTING AUTOMATED TESTING

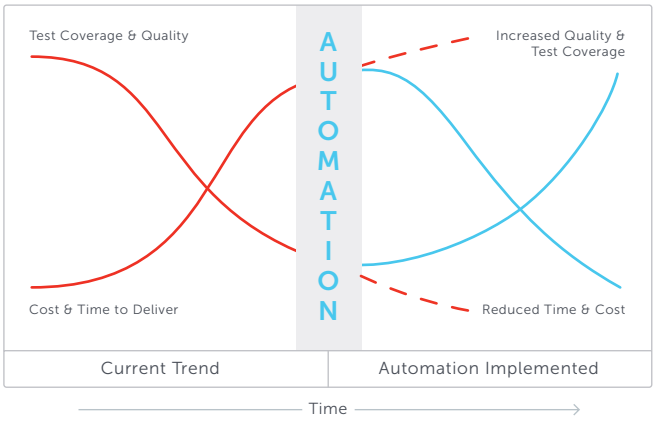
In the landmark book "Implementing Automated Software Testing," Elfriede Dustin refers to an IDT survey in which 37% of respondents that were not users of automated testing claimed a lack of time to be the main reason. Considering that one of the key benefits of automated testing is quicker time to market, this leads to the common 'chicken and egg' paradox:



While it is necessary to spend adequate time testing an app to maintain quality, the goal of a testing project should be to continually reduce the time spent in manual testing, and automate as much of the tests as possible.

However, when they take their first steps towards this goal, test and QA teams find that there is normally a "hump of pain" (a phrase first attributed to Brian Marick) associated with the change in workflow, and mindset of the team members. This is when teams are most likely to give up on automated testing.



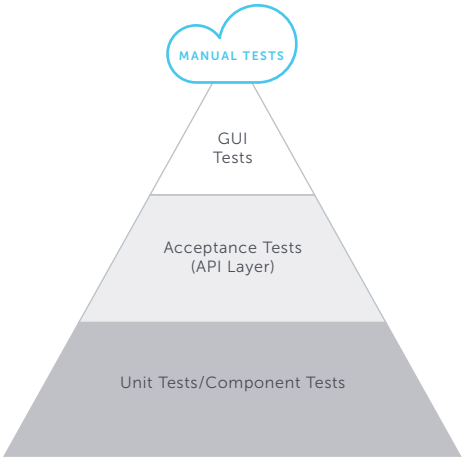


However, persisting with the automated testing strategy will pay off. If done right, the rewards can be worth the temporary pain. Over time, the effectiveness of manual testing reduces, while that of automated testing only increases.

When adopting automated testing, being prepared for the "hump of pain" makes all the difference. As Igor Khrol, a automated testing specialist, says, "Select the right tool, the right framework, and the right technical approach".

THE RIGHT APPROACH TO AUTOMATED TESTING

The right technical approach involves knowing which tests to automate and which to continue manually. Tasks that are repeated most often and are the most labor-intensive are often the best candidates for automation. Elfriede Dustin, in his book on automated testing, says "Our experience has been that 40% to 60% of the tests for most projects can and should be automated."



[The Appendix includes a checklist with questions for deciding which tests to automate.]

The automated testing pyramid developed by Mike Cohn suggests to automate more low-level unit tests than high level end-to-end tests running through a GUI.

Unit and Component Testing

Unit tests are a way to test individual units of the source code to determine whether they are fit for use. They are a series of low-level tests that pinpoint exactly where to search for bugs very early on in the cycle. The vast majority of commit tests should be comprised of unit tests.

Because of the need for quick feedback, unit tests should be very fast to execute, and be run in memory. They should cover a large proportion of the codebase (around 80%) to give a good level of confidence that when they pass, the application is fairly likely to be working. As Humble & Farley caution, "If your team is not automating unit tests, its chances of long-term success are slim. Make unit-level automated testing and continuous integration your first priority."

API or Web Services Testing

API testing operates at the business logic layer of the software. Instead of using standard user inputs (keyboard) and outputs, in API Testing, you use software to send calls to the API, get output, and note down the system's response.

One challenge with API testing is setting up the test environment. The database and server should be configured as per the application requirements. Once installation is done, the API function should be called to check whether that API is working.

GUI Testing

The goal of GUI testing is to ensure the frontend interface of the application works as expected in all browsers and platforms. In the early days, this used to be done by testers manually rendering the app on every possible combination of browsers and platforms. When it became impossible to test every combination of browser version and system on a separate machine, virtual machines (VMs) came to the rescue, enabling teams to test multiple combinations simultaneously on a single system. This helped somewhat, but testers still had to grapple with the manual configuration of VMs, and ever increasing hardware resources. This is a prime area for writing automated business-facing tests, understandable to both customers and developers that drive development.

Regression Testing

Regression testing seeks to uncover new bugs, or regressions, after modifications such as enhancements and configuration changes have been made to the application. Before a new version of the application is released, the old test cases are run against the new version to make sure that all the old capabilities still work.

With more frequent releases, teams typically tend to compromise on regression testing to save time, and in the process let bugs slip through the cracks. To keep up with faster development times, regression tests can be made faster using automation. Changes in the application may require the regression test scripts to be updated as well, which may require some manual intervention. However, automating regression tests saves time as teams don't need to run the same tests over and over again.

Functional Testing

Functional testing starts with a list of specifications, or a design document. Based on the specifications, various functionalities of the app is tested by feeding them input and examining the output against expected output. Functional testing doesn't consider the internal structure of the app being tested, and is purely focused on the functionality of the app from an outcomes point-of-view.

For a web application, a functional test could be simply that a user manually navigates through the application to verify the application behaves as expected. But since automating a test is the best way to make sure it is run often, functional tests should also be automated whenever possible.

Mobile App & Browser Testing

While much of what's been discussed so far applies to both web and mobile app testing, automating mobile app tests is more complex than doing the same on web apps. When automating mobile tests, it is imperative to use a combination of emulators and real devices to optimize costs and get your apps to market faster. The unlimited availability of emulators make them easy to spin up and use to test your app logic. On the other hand, the accuracy of real devices makes them ideal for functional testing later in the cycle. Going exclusively one way will mean you lose out on the benefits of the other.

Here are some of the differences to keep in mind when automating mobile app tests:

- Cross platform & device testing is even more complex with mobile. Mobile apps can be either native, web, or hybrid, further increasing the complexity. Cross-platform and device compatibility is one of the main reasons mobile app testing requires automation rather than manual testing.
- Emulators allow you to test your business logic and general display responses at scale, but may not give you the pixel-perfect resolution you might need or allow you to see how your app functions with the quirks of real-life phone hardware.
- Real devices allow you to test rendering of your app as well as hardware dependencies. However, these are generally slower and more expensive to test against than emulators. The breadth of mobile devices you need to test against along with the need to provide a consistent base configuration and a way to share results across groups make these suitable for automated testing in a cloud.

WHICH TESTS TO CONTINUE MANUALLY

Usability Tests

Usability testing is done to discover how easy it is for users to accomplish their goals with your software. There are several different approaches to usability

testing, from contextual enquiry to sitting users down in front of your application and filming them performing common tasks.

Usability testers gather metrics, noting how long it takes users to finish their tasks, watching out for people pressing the wrong buttons, noting how long it takes them to find the right text field, and getting them to record their level of satisfaction at the end. Usability, consistency of look and feel, and so on are difficult things to verify in automated tests, and are ideal for manual testing

One-off Tests

Tests that are run only one time or infrequently will not be high-payoff tests to automate. They are better done manually. If the one-off tests keep coming back after a point, it makes sense to consider automating them too. Thus, it makes sense to have a threshold for how much one-off testing you'd like to keep manual.

THE RIGHT AUTOMATED TESTING FRAMEWORK

A automated testing framework is a layer on top of the tools and processes that makes it easy to extend the automated tests to larger projects. While a detailed discussion of automated testing frameworks is beyond the scope of this paper, here is a brief description of the most popular types of frameworks:

- Linear Scripting: Writing all the steps in your action one after the other in a linear form.
- Data-Driven Testing: Test scripts are built in such a way that they work for different sets of data without any changes to the test script.
- Keyword-Driven or Table-Driven Testing: First identify a set of Keywords and then associate an action (or function) which each of these keywords.
- Hybrid Testing: Combines the features of the above types of frameworks.

Whichever framework you end up choosing, it's important to keep the framework separate from test script creation so it is easy to extend or modify later.

SELECTING THE RIGHT AUTOMATED TESTING TOOL

This is an important step that can have a big impact on the success of your automation efforts. Choosing the wrong tool can complicate the "hump of pain" problem, and increase chances of giving up on automated testing. Because of this, it's important to select a tool that is suitable for your needs.

If your app targets only a few platforms and browsers, a niche testing tool may just do the job, but most of today's web and mobile apps are built to run on a large number of possible combinations of operating systems and browsers, and you will most likely need a tool that covers all required platform and browser combinations. Additionally, the tool should be compatible with your existing technology stack.

For today's web and mobile apps, cloud-based testing tools are gaining popularity because of their ease of configuration, low maintenance, and tremendous scalability.

The Crucial Decision - In-house, Open Source, or Commercial

Considering the varying needs of a test and QA team, it's most likely that you'll need a combination of tools to support your entire software testing lifecycle (STLC). That said, it helps to focus on one platform that meets most of your automated testing needs first, and then find tools that perform specific tasks that the primary tool doesn't perform.

When selecting the primary testing tool, the important decision is to choose between in-house, open source, and commercial tools. In-house tools give you the most control, but require expertise to build, are hard to maintain and can lock you into one vendor. They also turn out to be expensive if you factor in all internal resources required.

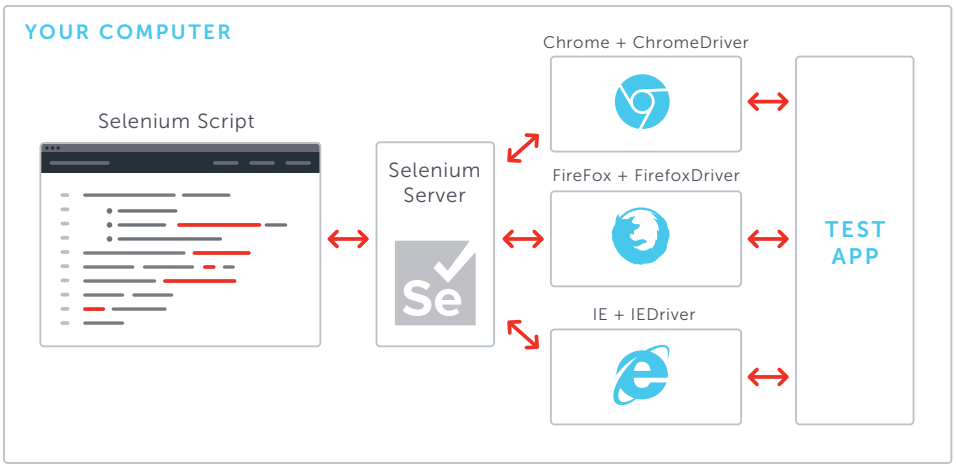
Open source tools have proven to be very capable in handling the most advanced automation projects, and are being preferred by more and more organizations. While they are easier to setup than building an in-house tool from scratch, they can become cumbersome to maintain. They require experts who specialize in open source tools. With no formal support available, the project can be delayed when complex issues arise.

There are plenty of commercial tools available, and they require careful examination before adopting one. They are typically easier to setup and maintain as the vendor would take on the load of keeping the tool up-to-date. They can be expensive compared to open source tools, but if saving time is a priority, they could be worth the extra dollar. Commercial tools also come with support, which gives organizations that are new to automated testing a fair level of confidence.

Considering the importance of open source tools in automated testing, let's look at two of the most popular open source automated testing, and consider how they may fit into your testing workflow.

Selenium - The Leading Automated Testing Framework for Web Apps

Selenium is the most widely used open source automated testing tool. It gives you the freedom to test your web app across browsers automatically, through test scripts. It is a technology that allows programmers to send commands to web browsers to make them perform tasks as though they were being used by humans. In this sense it is like a robot for web browsing.



Jason Huggins, creator of Selenium, says on LinuxInsider that "Selenium's success came from its ability to test Firefox and IE on Windows or Mac or Linux and be able to drive it from Ruby or Python." When thinking of automated testing for web apps, a Selenium-based solution is the ideal choice.

Appium - The Leading Automated Testing Framework for Mobile Apps

Appium is the leading open source automated testing framework for use with native, hybrid and mobile web apps. It drives iOS and Android apps using WebDriver, the same API that controls the behavior of browsers with Selenium.

Investing in WebDriver means you are betting on a single, free, and open standard for testing. It also keeps you from being locked into proprietary tools and technologies.

Appium allows automated tests to be written in most modern programming languages. This is a big advantage over Apple's UIAutomation library, which lets you write tests using only JavaScript, or Google's UiAutomator that supports only Java. Appium opens up the possibility of true cross-platform native mobile automation, and has become an indispensable tool for automated testing.

OPEN SOURCE TOOLS REQUIRE EXPERTISE TO RUN IN-HOUSE

While Selenium and Appium are powerful tools for browser-based automated testing, and mobile automated testing, these tools require a team with prior experience to be able to setup and maintain them.

Builds with Selenium tests can take long to run, and need to be optimized regularly. It can be hard to cover all the relevant browsers and platforms which will need to be manually installed, and configured frequently. This adds manual effort, and defeats the purpose of automated testing. Also, scaling a testing project can be complex and will require the Selenium Grid to be deployed, and tests to be rewritten to support multi-threading.

Unless there's an in-house team with adequate experience in both frameworks, organizations looking to leverage Selenium and Appium on their own may be setting themselves up for failure. Because of this, it makes sense to evaluate a commercial option that leverages both Selenium and Appium.

THE IDEAL SOLUTION SHOULD COMBINE THE BEST OF BOTH WORLDS - SELENIUM AND APPIUM

The ideal automated testing solution would need to be cloud based, eliminating the need to maintain a test grid yourself, and allow you to test apps and access data securely behind firewalls. It would use open source standard frameworks (like Selenium and Appium) and should start a pristine new VM for every browser instance to make sure your tests aren't polluted by data from previous activity.

Most commercial solutions today run their VMs on public cloud infrastructure, and can result in false positives associated with browsers that don't close completely. Solutions that capture screenshots and videos of tests can make debugging a lot easier, and would be preferred over solutions that don't offer these features.

Other desirable features include enabling massive amounts of parallel tests, supporting tests written in any programming language, and integration with your CI server. Finally, the tool would need to be easy to configure, have a large existing user base, and be backed by prompt customer support.

A tool that meets these requirements would make the ideal automated testing tool, resulting in a smoother transition from manual testing, a more effective CI/CD workflow, and eventually, higher quality software that reaches the market much faster.

CONCLUSION

Organizations making the transition to continuous delivery should simultaneously evolve their testing efforts from being predominantly manual to being increasingly automated. Despite the initial pains, the benefits of automated testing make it worthwhile to invest in the right tool, the right framework, and the right technical approach

Selenium and Appium have emerged as the best tools for testing web and mobile apps, but they are tedious to maintain with in-house resources. The ideal automated testing solution would be based on Selenium and Appium without the burden of manual configuration and maintenance. It would deliver pristine VMs for each browser instance, and a secure path to test apps behind a firewall, among other important features.

Organizations that invest in their automated testing efforts are poised to get the most out of their CI/CD workflows. They will eventually be the ones to ship higher quality products faster, and in doing so, will put their competition to the test.

APPENDIX

Ten things to consider when moving to automated testing.

AUTOMATED TESTING CRITERIA
Is the test executed more than once?
Is the test run on a regular basis (i.e. often reused, such as part of regression or build testing)?
Is the test impossible or prohibitively expensive to perform manually, such as concurrency, soak/endurance testing, performance, and memory leak detection testing?
Are there timing-critical components that are a must to automate?
Does the test cover the most complex area (often the most error-prone area)?
Does the test require many data combinations using the same test steps (i.e. multiple data inputs for the same feature)?
Are the expected results constant (i.e. do not change or vary with each test)? Even if the results vary, is there a percentage tolerance that could be measured as expected results?
Is the test very time-consuming, such as expected results analysis of hundreds of outputs?
Is the test run on a stable application (i.e. the features of the application are not in constant flux)?
Does the test need to be verified on multiple software and hardware configurations?

ABOUT SAUCE LABS

Sauce Labs ensures the world's leading apps and websites work flawlessly on every browser, OS and device. Its award-winning Continuous Testing Cloud provides development and quality teams with instant access to the test coverage, scalability, and analytics they need to deliver a flawless digital experience. Sauce Labs is a privately held company funded by Toba Capital, Salesforce Ventures, Centerview Capital Technology, IVP and Adams Street Partners. For more information, please visit saucelabs.com.

SAUCELABS

SAUCE LABS INC. - HQ
116 NEW MONTGOMERY STREET, 3RD FL
SAN FRANCISCO, CA 94105 USA

SAUCE LABS EUROPE GMBH
NEUENDORFSTR. 18B
16761 HENNIGSDORF GERMANY

SAUCE LABS INC. - CANADA
134 ABBOTT ST #501
VANCOUVER, BC V6B 2K4 CANADA