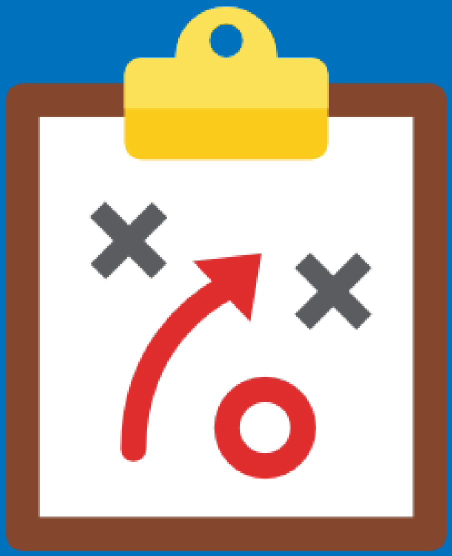
The Test Automation Playbook



A collection of recommended practices, tools & resources to help you achieve automation awesomeness

The Powerr Play Framework

P - Planning

Decide what tools, best practices and technique your team will use at a high level. This is the time to define a list of goals and a set of some guiding principles for your automation framework.

O - Ownership

Define who will be responsible for what. It should be a whole team approach. When a test fail team should care and fix ASAP. If your Agile make sure to add automation to your DOD

W - Writing

When ever possible use unique element ids. Use explicit not implicit waits. Have a test data management plan in place.

E - Execution

Test need to be reliable. Flag any flaky test and fix ASAP or add to tech debt

R - Reporting

Make sure all test results are visible to whole team. Track trends/metrics

R- Refactoring

Refactoring is good. Always work on improving test and process

Automation Testing Resources & Best Practices

What is Automation Testing?

Automation testing usually revolves around improving the execution speed of manual testing. During regression testing, a manual tester will take an existing test case procedure and execute it step by step. This can be time consuming since it is a manual process done by hand.

Because of this, to save time, many companies try to take their manual test cases and convert them to an automated test case. An automated test tool then executes the test steps automatically without human intervention.

Also, automated tools use a programming approach to emulate a user interacting with an application and verifying test steps using programming assertions.

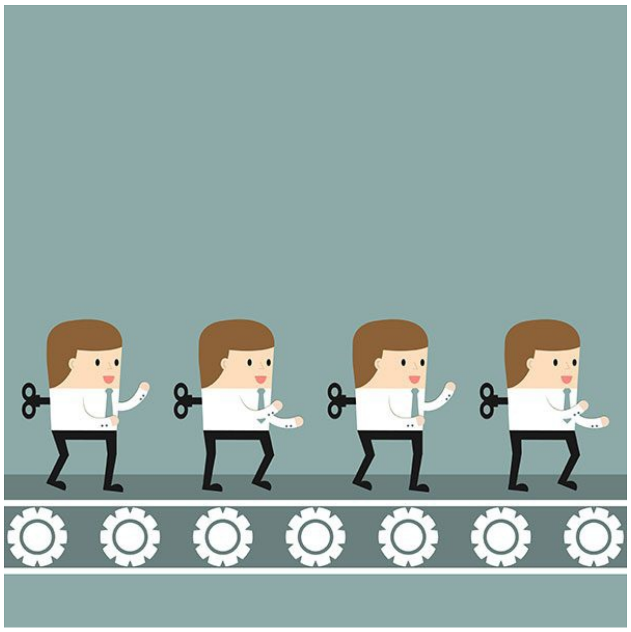
In any case, before we take a look at automated testing, let's touch on some issues with manual testing.

Pitfalls of Manual Testing

There are a few reasons why manual testing may be problematic:

- It uses a lot of resources
- It's time consuming
- It sometimes lacks proper coverage
- Due to their receptiveness, testers may get bored and miss steps when executing manually, leading to possible inconsistencies.

Automation testing should be used to help with your testing efforts. But does automation replace all your testers?



Automation Testing Does Not Replace Testers

Some people assume it automation means to replace human testers.

In actual fact, however, it's the opposite. Automated tests are great for running tests precisely and quickly but they in no way replace human testers.

Automation is also great for running the same steps over and over again, but they don't actually *think*.

Although we can agree that *automation testing* does not replace other testing activities, with today's software development environment and continuous integration practices it is critical. With the increased speed in which we develop software, we need automation testing.

Another reason is that Agile and DevOps practices demand more automation. Practices like continuous integration and delivery require automated tests that run quickly and reliably. I'd actually go so far as to say that in today's modern development environment, we can't succeed without automation.

So what are some reasons for using automated tests?

Why Use Automated Testing?

The theory is that an automated test will save the company both time and money.

But it seems that many folks fail to factor in the amount of time and money it takes to maintain automated test suites. Some other reasons for automated testing are:

- Verify newer versions of software
- Free testers up to focus on more exploratory-type testing
- Automated tests are more repeatable
- Data population
- Accurate benchmarking
- Less false failure due to human error
- Greater test coverage
- Reusability
- Quicker release of software
- Get fast feedback to your developer on failing checked in software
- Saves time
- Ability to leverage programming capabilities

So are there any downsides to creating automated tests?

Pitfalls of Automation Testing

Since automated tests rely on programming languages for their creation, automation becomes a full-blown development effort.

As a matter of fact, what you are doing is developing a piece of software to test another piece of software.

Treat your automated code just like your development code. Follow the same processes and best practices you would use for any other software development project.

Automation testing is difficult and complicated, just like most other development software projects. It also presents many of the same issues other software programs do. So best practices for development software apply to automation as well.

Treating your automation as a second-class development project will cause maintenance and reliability issues in the long run.

Other Automated Testing Problem Areas

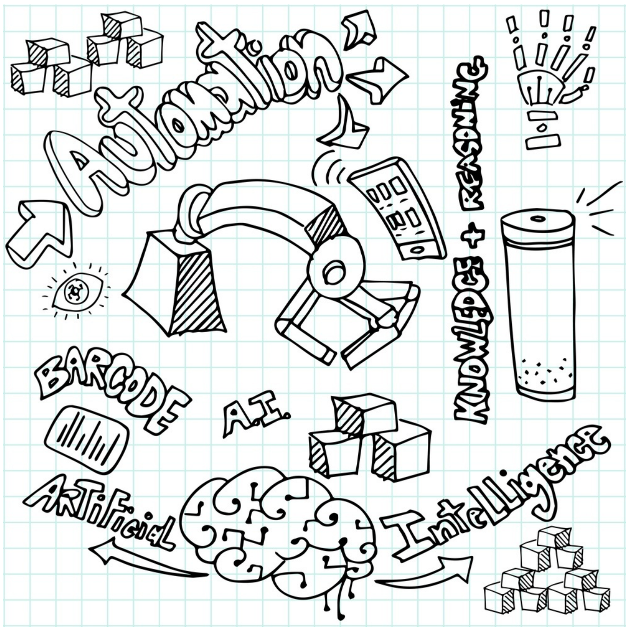
Some other automation testing pitfalls are:

- Setting unrealistic goals
- Believing that automation tests will find more new defects
- A false sense of security
- Underestimating the amount of time it takes to maintain automation
- The creation of large, end-to-end tests should be avoided. Tests should be atomic so that when they fail, you know why
- Focusing on UI automation only
- Not having a controlled test environment

- Ignoring failing tests
- Not having a test data strategy in place
- Not reusing automation code
- Developers not making their code automatable
- Not using proper synchronization in your tests
- Not making your automated test readable
- Creating automated tests that add no value
- Hard coding test data
- Not using a whole team's collaborative automation efforts
- Expecting automation testing to replace manual testers

Because of the above reasons, teams often claim that automation testing "doesn't work."

Another common question I'm frequently asked is, "Which tests should be automated?"



What should be Automated?

You shouldn't try to automate everything. In fact, not everything *is* automatable. When planning what test cases to automate here are some things to look for:

- Test that are deterministic.
- Tests that don't need human interaction
- Test that need to run more than once
- Any manual process that will save engineers time (not necessarily an official "testing" process)
- Test that focus on the money areas of you application
- Test that focus on the risk areas of your application
- Unit tests
- Test that need to run against different data sets
- Test that are hard to test manually.
- Focus on critical paths of your application
- Test that need to run against multiple builds and browsers
- Tests used for load/stress testing

The more repetitive the execution is, the better candidate a test is for automation testing. However, every situation is different.

Ultimately, you should consider using automation for any activity that saves your team time. It doesn't have to be a pure testing activity; you can leverage automation to help reduce the length of time-consuming activity throughout the entire development life cycle.

At this point, some of you may be asking, “What is the ROI of test automation?”

ROI - What is the Cost of Test Automation

Determining the ROI of your automation testing efforts can be tricky. Here is a common calculation some folks use to get a rough estimate of their test automation costs. This can also help you decide whether a test case is even worth automating as opposed to running testing it manually.

***Automation Cost** = how much the tools cost + how much the labor cost to create an automated tests + how much it cost to maintain the automate tests*

Consequently, if your automation cost calculation is lower than the manual execution cost of the test, it’s an indicator that automation is a good choice.

Moreover, ROI quickly adds up with each re-run of your automated test suite.

Because it’s critical that you get a good return on your automation investment, there are some things you *shouldn’t* automate.

What Not to Automate

There are exceptions to everything, of course, but in general you may not want to automate the following test case scenarios:

- One time test
- Ad hoc based testing
- Test that don't have predictable results
- Usability testing
- Application not developed to be testable

In addition to what *not* to automate, another element of a successful automation project is having an automation framework.

What is an Automation Framework?

An automation framework is a common set of tools, guidelines and principles for your tests. Having a framework helps to minimize test script maintenance. In my opinion, common pieces and best practices of a framework are:

- Set your manager's and team's expectations

- Break your automation framework into abstraction layers
- Use proper synchronization methods
- Determine a strategy for training and retraining your framework users
- Use version control
- Look for existing libraries/tools before inventing your own
- Do code reviews on all automated tests
- Include reporting and logging that make debugging easy
- Follow a naming convention
- All elements should have unique IDs
- Avoid coordinate-based automation
- Create reusable methods
- Create reusable utilities
- Refactor code often
- Use Page Objects
- Make tests readable. (Your tests should read like English)
- Avoid code duplication
- Have a test data management strategy
- Handle running test in parallel
- Support mocking and stubbing
- Include automation on your sprint teams definition of done (DOD)
- Remember that automation is a collaborative, “whole team” effort
- Separate your tests from your framework

Start Your Automated Testing Efforts Off Right

It's a given that your applications are going to change over time. And since you know change is going to happen, you should start off right from beginning using best practices or design patterns, like using page objects, doing so will make your automation more repeatable and maintainable.

Test Automation Process

- First, prepare - understand the functional testing objectives. Understand what test data is needed. Know what needs to be verified.
- Write – turn the requirements into an automated solution. Know what the start and end conditions are for each test. Tests should be completely independent of other tests.
- Add proper assertion checks to ensure your application is behaving according to your specifications.
- Each test should have a particular purpose.
- Execute – your tests should be reliable. Run each test at least three times in a row before checking in code.

- Evaluate - verify that the automated test is doing what you expect it to do. Have manual tests verify that it is working as expected. Remember -- if it’s not asserted, it’s not checked.
- Communicate – be sure that everyone on your team is aware of the results. Flaky tests should be fixed ASAP, or you'll risk your team ignoring your test results.
- Repeat/refactor - if you notice a flaky test, refactor it to make it more reliable. Most importantly, deletes any tests that are not reliable and haven’t been fixed within a given time frame.

Test Size Matters

Because tests need to run quickly, test size matters.

At this point, many people visualize a traditional test pyramid, which has unit tests as its base.

Integration tests are in the middle, and GUI tests are at the top.

But I think more in terms of test size. By test size I'm referring to tests that are faster than others. While I understand the need to run UI tests, if you have to create one, make it as fast as possible.

- Don't do all UI end-to-end journeys.

In addition, when an automated test fails you need to know *why*. Keep test small, fast and limited in scope as well as having a readable name will go a long way in helping troubleshoot issues when a test fails. Furthermore, you should endeavor to get feedback to your developers as quickly as possible, and the best way to do that is with a fast, well-named test.

- Unit test, integration test tend to be faster.

If you're into Star Wars, you notice that Yoda in the image above is the smallest test size, and they tend to be more powerful too.

As you move up the pyramid, UI tests can be evil and can drive you crazy, so although you need them, just be careful.

How to pick a Test Tool

There is no "correct" test tool for automation testing. Ultimately, it all depends on your team's unique needs and skill set.

Also, you sometimes need a combination of different tools to get the coverage your application requires.

Due to this there are a few things to look for when trying to select a test tool:

- Look at the product road map and make sure the tools you select will handle future features and technologies.
- Evaluate the cost, including maintenance.
- Use a tool that a leverages the same tools and languages your developers use.
- Don't just assume a tool will work for you. Create small POC for each tool and get team feedback before committing to anything.
- Is the tool extensible?
- How easy is it to use and get started?
- Does it provide reporting and debugging capabilities?
- Does it recognize all the objects in your application?
- Can it integrate with other tools like version control, test management tools, continuous integration tools?
- Find out if the tool has an active user base.
- Select tools that other companies are using.
- How much training will it take to get your teams up to speed with the tool?
- Finally, determine how easy is it to hire folks that have the skills needed to create your automated tests

Automation Testing Tools List

Open Source Tools

Selenium – Has arguably become the de facto test tool standard for browser-based testing. Please remember: you cannot use selenium for non-browser applications; not everything can be an automation testing Selenium script.

Appium – Automation for apps. Appium seems to be the winner in the mobile testing space so far.

Watir - is an open source Ruby library for automating tests. Watir interacts with a browser the same way people do: clicking links, filling out forms and validating text.

Sikuli - What's cool about SikuliX is that it allows you to automate anything you see on your screen using image-based testing.

WinAppDriver - Windows Application Driver is a service to support UI Test Automation of Windows Applications

White Framework - White is a framework for automating rich client applications based on Win32, WinForms, WPF, Silverlight and SWT (Java) platforms. It's .NET based and doesn't require the use of any proprietary scripting languages.

In fact, test automation programs using White support your writing with whatever .NET language, IDE and tools you are already using. White also provides a consistent, object-oriented API, hiding the complexity of Microsoft's UIAutomation library (on which White is based) and Windows messages.

AutoIt - AutoIt v3 is a freeware BASIC-like scripting language designed for automating the Windows GUI and general scripting. Many teams integrate AutoIT with Selenium to work around non browser windows that appear in a automated test workflow.

Javascript Test Automation - Top 8 Essential Javascript Automation Frameworks

Visual Validation Tools - The Top 21 FREE Visual Validation Tools for Testers

Serenity - One of my favorite automation frameworks around. Serenity is a great open-source tool because it acts like a wrapper over Selenium and BDD tools like jBehave and Cucumber JVM. That means there's a lot of built-in functionality available to you in Serenity that takes care of many things you would normally have to code from scratch if you had to create your own BDD framework. What Serenity is *really* awesome at is creating unbelievable reports. Out-of-the-box Serenity creates living documentation that can be used not only to view your Selenium BDD test results but also as documentation for your application.

Gauge - Gauge is a test automation solution that's being built by ThoughtWorks; it's cross platform and open-source. It supports multiple languages including Ruby, Java, C#, Python and Javascript, and it has upcoming support for other languages — like Golang — as well.

Sahi - The first thing you need to know is that *Sahi* comes in two flavors: open source and a pro version. Sahi Pro is the enterprise version of the open-source project. It includes lots of features coveted by larger organizations.

Robot Framework - If you want to use Python for your test automation efforts, you can't go wrong using the

Robot Framework. The Robot Framework is a mature solution that was created for testers and uses a keyword-driven approach to make tests readable and easy to create. It also has many test libraries and other tools you can use

RedwoodHQ takes a little bit of a different approach from the other tools on this list. It creates a website interface that allows multiple testers to work together and run their tests from one web-accessible location.

Galen - If your automation efforts are focused on User Experience Design (UX)/Layout testing, Galen Framework might be a perfect fit for your needs.

Vendor-based Automation Test tools

Applitools - Actually Applitools integrates with both Vendor and Open-source solutions. If anyone has tried doing any sort of visual testing using tools like Eggplant and UFT Insight, you know how hard it is to make these types of tests reliable.

Sometimes the tests are so fragile you can only run them on the same machine they were developed on to avoid flaky tests.

There are many reasons for this, but it's mostly due to pixels being slightly off from one browser or OS. Applitools is different in that it was developed from the ground up for visual validation and its sophisticated algorithm was designed to handle many pixel issues that most other image-based testing tools have a hard time handling. Applitools allows you to find and automatically detect all the visual bugs to validate the visual correctness of your application.

UFT LeanFT - Essentially combines the best of both the vendor-based and open-source worlds by morphing Selenium with some key functionality currently found in UFT (QTP).

Microsoft CodedUI – Actually uses Selenium to help test Chrome and Firefox browsers. But unlike Selenium, which is only for web-based testing, CodedUI is unique in that it allows you to automate a bunch of different technologies and is not limited to the browsers.

SmartBear Testcomplete - Allows you to automate web, desktop and mobile applications. Best of all, you can choose from script-free, drag & drop functionality or JavaScript, Python, VBScript, JScript, DelphiScript, C++Script or C#Script as a scripting language.

IBM Rational Functional Tester - Like most companies, IBM's test portfolio has grown with the acquisition of tools like Rational and Green Hat. It appears that much of the strength of its functional test tools comes from its support of numerous technologies including Windows, Mac, and mobile platforms.

Tricentis - Self-billed as "the continuous testing company," which is in line with Gartner's finding that one of its strengths is its extensive efforts to support Agile testing and continuous improvement processes.

Worksoft - Worksoft is well known for its ERP business end-to-end solutions.

TestPlant - One of the few test automation tools listed that has strong support for Apple's platform. In fact, because of its unique, image-based recognition approach, it has the ability to test hard-to-automate applications -- especially those with object recognition issues. Unfortunately, anyone who has done image-based, functional test automation knows how difficult these types of tests can be to maintain, and some customers noted that as an issue.

Ranorex - Supports a ton of technologies across all kinds of platforms — all from one tool.

Noteworthy, however, is that it lacks a full, end-to-end solution and focuses mainly on functional test automation.

Progress - For those of you who may not be familiar with this company, Progress recently acquired Telerik, which is the home of the popular free debugging tool Fiddler. Also I know a few test engineers who actually use Progress's Test Studio as a front end for their Selenium test automation

efforts. Strengths of Progress are its integration with Visual Studio and its supported languages.

Automation Anywhere - Does not support testing for packaged applications like SAP. Also lacks support for native mobile apps testing.

API Automation Test Tools

Open source API tools

Rest-Assured - Rest-Assured is an open-source Java Domain-specific language (DSL) that makes testing REST service simple. It simplifies things by eliminating the need to use boiler-plate code to test and validate complex responses. It also supports XML and JSON Request/Responses.

RestSharp - Simple REST and HTTP API Client for .NET

Postman - Postman is a rest client that started off as a Chrome browser plugin but recently came out with native versions for both Mac and Windows.

SoapUI - is the world leading Open Source Functional Testing tool for API Testing. It supports multiple protocols such as SOAP, REST, HTTP, JMS, AMF

Fiddler - Fiddler is a tool that allows you to monitor, manipulate and reuse HTTP requests.Fiddler does many things that allow you to debug website issues, and with one of its many extensions you can accomplish even more. Check out my article on how to get started with Fiddler

Karate - Since Karate is built on top of Cucumber-JVM, you can run tests and generate reports like any standard Java project. But instead of Java - you write tests in a language designed to make dealing with HTTP, JSON or XML - **simple**.

Vendor API Tools

SoapUI Pro - Since the free version is open-source, you can actually gain access to the full source code and modify as needed. The pro version is more user-friendly, and has additional functionality including a form editor, an assertion wizard for xpath, and SQL query builder.

UFT API - In previous releases HP had separate products for functional testing. QuickTest Professional (QTP) was used for testing GUI applications, and Service Test was for testing non-GUI technologies. HP's latest test tool release — Unified functional Testing (UFT) — combines both products and features a frontend that merges the separate tools into one common user interface.

Test Execution Report Tools

Allure - an open-source framework designed to create test execution reports clear to everyone in the team.

Run Your Automated Test in the Cloud or On Mobile Devices

Here are some vendors that allow you to save a ton of time by running your test in the cloud and on multiple OS, devices, configuration. Get rid of the headache of having to maintain your own in house lab/grid.

Sauce Labs

Perfecto

Browserstack

Automation Test Management Tools

Zephyr - Manage all aspects of software quality; integrate with JIRA and various test tools, foster collaboration and gain real-time visibility.

QaSymphony - Has a platform called qTest for Software testing and QA tools built for Agile

Automation Testing Courses

Finally, here's a list of my favorite automation testing courses:

Automated Web Testing with Selenium - John Sonmez

Creating an Automated Testing Framework with Selenium - John Sonmez

Quick Guide to API Testing with HP's Unified Functional Testing - Joe Colantonio

Test Automation with CodedUI

Selenium WebDriver Basics with Java - Alan Richardson

Complete Selenium Webdriver with C# building a Framework - Nikolay Advolodkin

Robot Framework

The Java Selenium Guidebook - Dave Haeffner

Test Automation Conferences

AutomationGuild - Automation Guild is the *first ever* event of its kind, 100% online conference that is dedicated to helping YOU perfect the craft of creating automation awesomeness and accelerate your automation career. If you missed the LIVE Automation Guild event? No worries! Due to demand I decided to keep registration open. So you can still get all pre-recorded sessions and recorded Q&A now!

SeleniumConf - SeConf brings together Selenium developers & enthusiasts from around the world to share ideas, socialize, and work together on advancing the present and future success of the project.

STPCON - The Software Test Professionals Conference is the leading event where test leadership, management and strategy converge

Want More?

I wanted to make a guide that anyone could use to quickly get started improving their test automation efforts. For more in depth hands-on automation awesomeness check out:

THE AUTOMATION GUILD CONFERENCE