Test Strategy

**Defining Project goals**

1. Automate tasks that are highly repetitive.
2. Tasks that carry high risk or probability of failure is high
3. Tasks that require testing over multiple browsers/devices/OS/environment/hardware/configurations
4. Tests with clear pass/fail results
5. Automate tests that require operating through multiple data sets
6. Automate exercises that allow for multiple options, for example, form submission that accepts different combinations.
7. Tests that require a lot of time if done manually; for example, we had a test that required logging in every time we run a new task.
8. Lastly, you should definitely automate tests that require checking of stable features.

## Planning a test approach

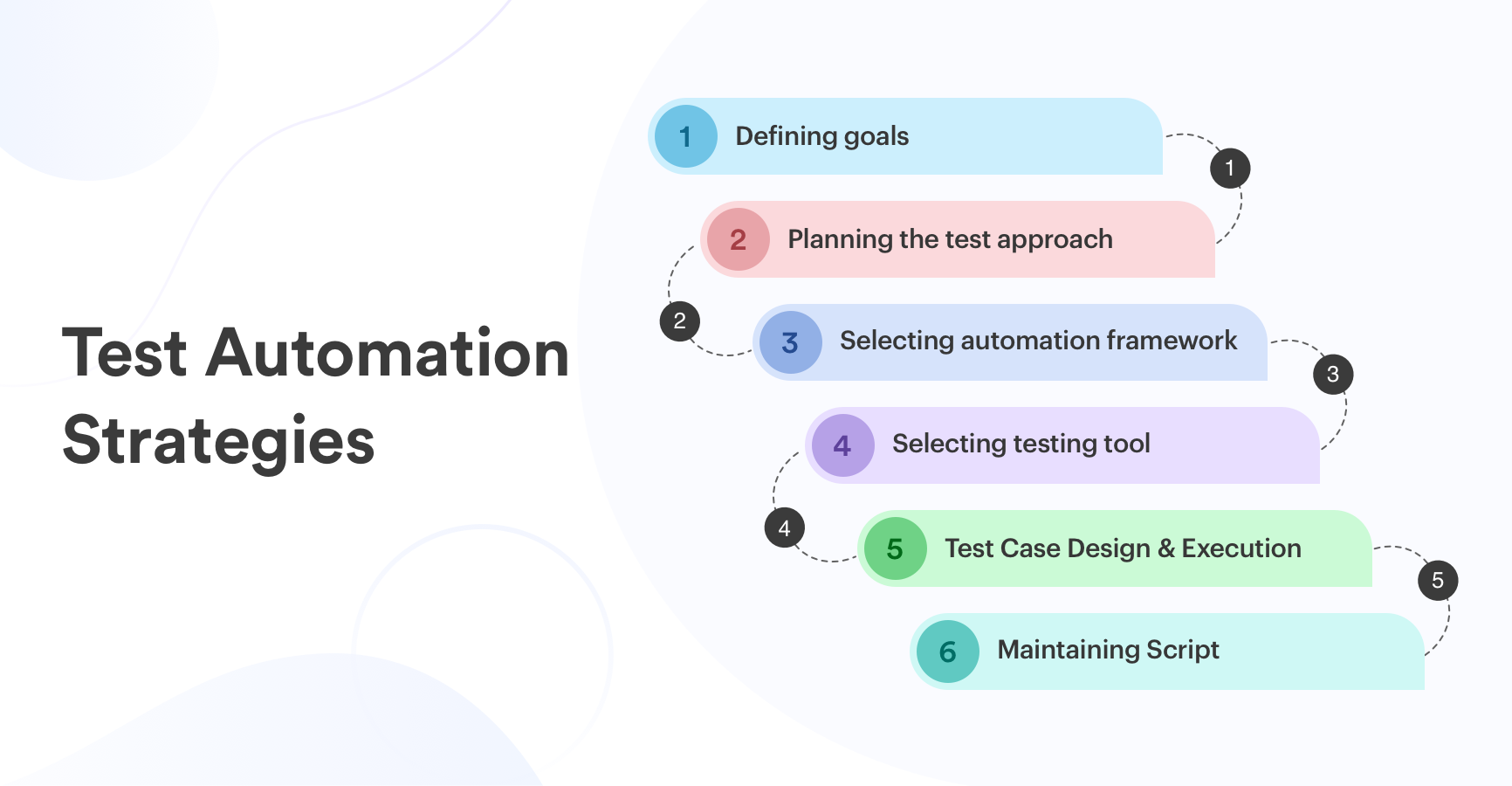
* [CASE STUDIES](https://www.simform.com/case-studies/)
* [HOW IT WORKS](https://www.simform.com/how-it-works/)
* [BLOG](https://www.simform.com/blog/)

[CONTACT US](https://www.simform.com/contact/)

[Simform](https://www.simform.com/)[Blog](https://www.simform.com/blog/)[Software Testing & QA](https://www.simform.com/category/software-testing/)

# How to Build a Test Automation Strategy?

[Software Testing & QA](https://www.simform.com/category/software-testing/)



**Quick Summary :-** From Google to Netflix, everyone's eyeing test automation. But why is it still at 14-18% usage in the world? You can't afford to go wrong with your test automation strategy considering today's competition. Read more to find out how to build a successful one as I share with you the lessons we've learned during the years.

## How to Build a Test Automation Strategy?

Software development is a path full of chaos. Here’s why!

Software projects start with excess enthusiasm. It starts with the design, then comes the architectural pattern, and everything looks fine. Even ahead of the schedule. But then things get weird. Rework happens and you’re now a bit behind schedule.

In the end, your time-to-market is months behind. You cut some corners knowingly acknowledging the technical debt. Because, of course, you’ll take care of that later. Finally, you ship to production. First some bugs trickle in and then it streams in. Now, you’re in the loophole of breaking/fixing things in parallel with new development.

You think [software testing](https://www.simform.com/services/software-testing/) automation can save your time and things can be back on track. You start doing things only to find that everything is haywire. Because you don’t know which tools to use or have any software test automation strategy in place. At the end, it turns out, you’re never going to pay down that technical debt.

Why does implementing test automation seem to be a rescue option, rather than the rule of thumb, in software development? Why, after all these years, does software testing still inevitably seem to work out this way?

Let’s help you out!

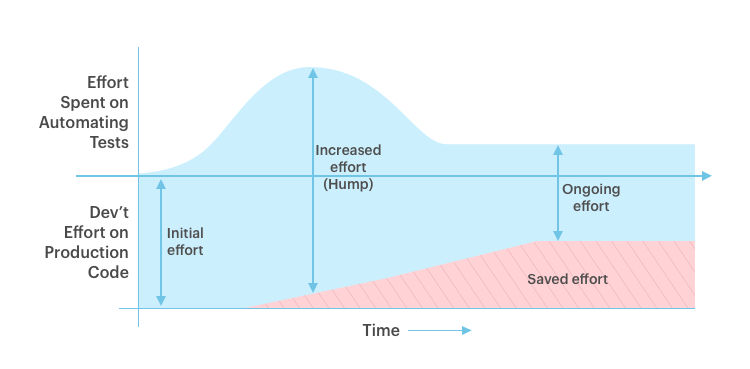
**Quick links:**

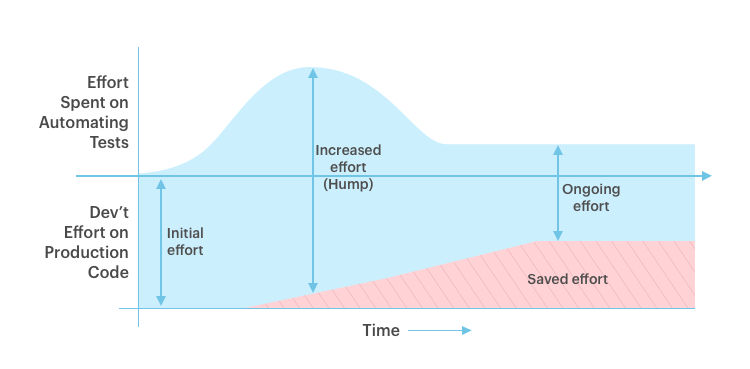
* [Test Automation: How to get started?](https://www.simform.com/test-automation-strategy/#section1)
* [Test Automation Strategy](https://www.simform.com/test-automation-strategy/#section2)
* [Test Automation Strategy Document: Steps to follow](https://www.simform.com/test-automation-strategy/#section9)

## Test Automation: How to Get Started?

One of the missing parts in the software development industry is bug fixing, which leads to delays in projects. This is backed by the [**world quality report**](https://www.sogeti.com/explore/reports/world-quality-report-2019/#download)that suggests **61% respondents** have difficulties in automating their QA, resulting in the biggest disregarded costs. Test automation breaks the conventional pattern of finding and fixing bugs and fixing them back followed by the longer regression tests.

***“Test automation is making software work to test one.“***





*The figure shows the initial time & efforts it takes in learning the best practices of test automation. But we can see how it saves the overall effort and time spent.*

*It’s like, it takes time to learn a programming language. But then, we have today’s world.*

The critical scenario that comes up is at the time of software delivery. And testing turns out as the common “bottleneck.” We started our test automation journey back in 2010, which brought a major change in our processes. Now, why did we need that change? Not because we couldn’t cope with faster deliveries, but our process needed to be more productive.

I didn’t have much experience with test automation back then, but it was more of a learning experience for all of us. We, at Simform, however, applied test automation for a pharmaceutical client (one of our then clients) where the UI was tested first followed by the known areas and eventually, we took off in the direction of fully automated test runs.

At the time of manual testing, we used to allocate half of the time to testing.  Later, if things go well, we deliver the product/software. Otherwise, again we’d get back on phase one of development. Because developers could never come to know the faults until they complete the entire thing.



With test automation, we’ve learned a lot and have started strategizing it with the help of experienced agile coaches. Let’s understand the test automation strategies below.

## Test Automation Strategy

As said in the tweet below, test strategies should be made specific to the project. Let’s walk through the steps that will help you define the strategy for your next project.

[](https://twitter.com/jrosaproenca/status/1191653725884030976)

**Jump to:**

* [Define your goals](https://www.simform.com/test-automation-strategy/#section3)
* [Planning a test approach](https://www.simform.com/test-automation-strategy/#section4)
* [Selecting test automation framework](https://www.simform.com/test-automation-strategy/#section5)
* [Select your tool](https://www.simform.com/test-automation-strategy/#section6)
* [Create and run the tests](https://www.simform.com/test-automation-strategy/#section7)
* [Maintain Scripts](https://www.simform.com/test-automation-strategy/#section8)

## #1. Defining your goals

When I started my career, test automation wasn’t looked up with many aspirations. We have faced many test automation challenges like you might have too. If you ask me today, things have changed dramatically, mainly due to the availability of reliable tools. However, this doesn’t mean we automate everything we can. The first step towards a successful enterprise test automation strategy is to define our goals and figure out which tests to automate.

The deciding factor for any test to automate is linked with how many times that particular test can be repeated. The best example of a test that can be automated is the one that runs often, is a mundane task, time-consuming, and requires a huge amount of data to perform the stipulated task. Here’s the list of potential test cases that can be automated:

1. Automate tasks that are highly repetitive, for example, log-ins.
2. Tasks that carry high risk or probability of failure or high
3. Tasks that require testing over multiple browsers/devices/OS/environment/hardware/configurations
4. Tests with clear pass/fail results
5. Automate tests that require operating through multiple data sets
6. Automate exercises that allow for multiple options, for example, form submission that accepts different combinations.
7. Tests that require a lot of time if done manually; for example, we had a test that required logging in every time we run a new task.
8. Lastly, you should definitely automate tests that require checking of stable features.

[Andrei Mikhailau](https://www.scnsoft.com/experts/andrei-mikhailau), Software Testing Director at ScienceSoft, and his team applies test automation to reduce the meantime to validate new features or fixes, improve regression testing coverage, and eliminate human errors. They also apply automated tests when manual testing is inefficient or impossible, e.g., for testing performance.



However, their biggest challenges in test automation resides in surviving the fast and frequent UI layout and functionality changes. Even the smallest modification may require a rewrite of a substantial number of tests. One of the ways to minimize this inconvenience is to ensure maximum test code reuse. Creating standard high-level application-specific libraries is a suggested option here.

Success in test automation requires you to plan things meticulously. Once you’re done defining your goals and scope of test automation, the next step is to look out for different test approaches.

You may also like: [Microservices Testing Strategies, Types & Tools: A Complete Guide](https://www.simform.com/microservice-testing-strategies/" \t "_blank)

## #2. Planning a test approach

My overall experience in testing has taught me one thing: the overall testing culture in any organization is highly influenced by the prevailing testing approach, apart from how they envision the test automation and how are they planning to collaborate beyond the development teams.

This means retrospecting the current process, finalizing and deciding the new test approach as well as determining the level of testing, roles, and responsibilities of the team members.

But first, let’s decide on the testing method that can provide the maximum value when automated. Different testing methods can be employed for different testing levels.

### **Unit Testing**

Unit testing is the backbone of any test automation strategy in agile that provides the team with the highest ROI. This testing employs a small piece of code (either function or method) that developers can write, execute, and maintain.

For instance, if you’re constructing a building, a unit might refer to the cement. This cement is one functional piece of code that is irrelevant outside the context. You could test this grey powder separately to check if it hardens after mixing with the water.

Likewise, each unit should be tested individually, but this won’t provide you with a house. Unit testing will give you observability at a granular level which is important for the developer, however useful for the product owners. It is recommended to run these tests locally and in the build.

### **Regression Testing**

Imagine yourself updating your product build. Now, you return to your product to check the new functionality only to land in disappointment because the feature that you’ve been using since ages has stopped working.

Each time you introduce new functionality into your product, you need to check whether it breaks or makes the existing features. Since regression tests return investments after each build update, they should be your second most priority after unit tests.

### **Component Testing**

Let’s take our constructing a house metaphor a little further. You’ve tested bricks, cement, tiles, and everything that is needed to construct the house. Will all these ingredients result in a house after they’ve been assembled together? And does it look like a house at all?

Component testing doesn’t care about the quality and quantity of these entities. It only tests the overall business value.

### **End-to-End Testing**

Imagine a house. A house will have all sorts of parts– bathroom, porch, dining room, kitchen, etc. Are each of these areas easily accessible? For example, will a person looking for a bathroom result in him actually finding a way to it? The utmost concern of the end-to-end tests is to see if all the components of the product can interact with each other in the correct order.

As the name suggests, this type of testing is slow since it conducts the test of the entire business flow from beginning to end. It is surely an effective way to verify the system interaction, but only when you don’t have too many systems.

We always run these tests in our release pipeline once the system is deployed. You can also run it locally if your architectural pattern requires.

As said earlier, end-to-end tests are complex by nature. So how do you tackle this? Well, the solution that has worked for us is to select the tools that are easy to use, maintain, and scale so as to ensure the long term success.

### **Integration/API Testing**

Integration and API testing allows you to test how well the system works with the external systems. In other words, how stable your system is when it comes to interacting with APIs.

Since modern software development relies on APIs & SDKs to a larger extent, validating the overall business logic and app’s behavior is critical.

The best way to execute these tests is to run them before deploying the system in your release pipeline. We often automate API tests that are repeatedly executed like regression cases. The common test cases where we run automated API testing are the areas where the changes of the code directly affect the API functionality.

### **Cross-browser Testing**

Checking your app’s functionality over multiple browser/OS/devices/screen resolutions is no brainer. Conducting these tests manually is a time-consuming task. But what if you want to do it at scale with high speed? The answer, automating the cross-browser tests.

Automated cross-browser testing has tremendously increased the test coverage and reduced our testing efforts and time by 80%. Leverage the power of test automation by conducting tests for the most common combinations your target users employs. Another focus area should be testing high-risk features.

### **Smoke Testing**

We conduct smoke tests to verify and get instant feedback about the system builds. It ensures that all the critical functionalities (config, permissions, .net frameworks, etc.) are working correctly.

If these parameters  fail under smoke tests, the product updates are rolled back. However, a series of smoke tests is even more important since it gives us the answer to these three preliminary questions:

* Is the application successfully initiating without an error?
* Is the login functionality working correctly?
* Are the primary buttons and menus working correctly?

**. Selecting Test Automation Framework**

The test automation framework is a set of detailed guidelines that one needs to follow while writing and running tests. For example, coding standards, procedures, test-data reporting, etc. Here is the list of six test automation framework that you can choose from:

**Linear Scripting- Record & Playback**

One of the handiest frameworks of all. Once set, you can play it back for the rest of the time. It allows testers to record the steps sequentially. For example, navigation, inputs, etc.

Pros:

* Coding expertise is not required
* Generates test scripts faster
* Maintains the sequence and hence easy to understand for anyone

Cons:

* Impossible to re-run the test cases with multiple datasets
* Not scalable for the extended project scope
* Rework will require for the changes in the application

**Library Architecture Testing Framework**

Library architecture works on the principle of identifying and dividing. That means the framework easily identifies the common tasks and group them accordingly. The framework saves these similar functions into the library and uses it whenever needed by the test scripts.

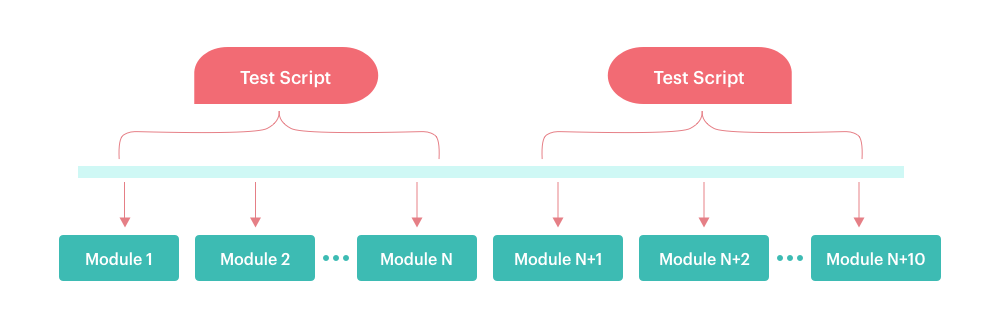
Pros:

* Maintains high-level of modularity
* Cost-effective and scalable
* Easy to run multiple test scripts

Cons:

* Requires changes to the scripts since data is hardcoded
* Technical expertise is required

**Modular Based Testing Framework**



As the name suggests, the framework divides an application into separate units and conducts isolated testing. A separate test script is created for each part and later combined as merged tests.

Pros:

* Modular changes will not affect the entire application
* Test scripts are reusable
* Less efforts and time for creating test scripts

Cons:

* Multiple datasets can’t be used
* Technical expertise is required for setup

**Data-driven Test Framework**

 It overcomes the glitch of the linear or modular-based framework. It doesn’t hardcode the data, but allows to store and access it from the external file such as Excel, CSV, etc. It allows testers to test the same feature with different sets of data.

Pros:

* Tests with multiple datasets are possible
* Less number of scripts
* Future changes in the modules will not affect the entire application

Cons:

* Framework setup is time-consuming
* Expert needed to format the data sources
* Data is hardcoded

**Keyword-driven Test Framework**

Keywords are used to represent the actions performed on GUI. For example, phrases like “click the link to verify” or “click the login button”. It stores the keywords externally and is used for testing the GUI of an application. These keywords are separated from the test logic, and it makes seamless testing of actions by following a set of instructions.

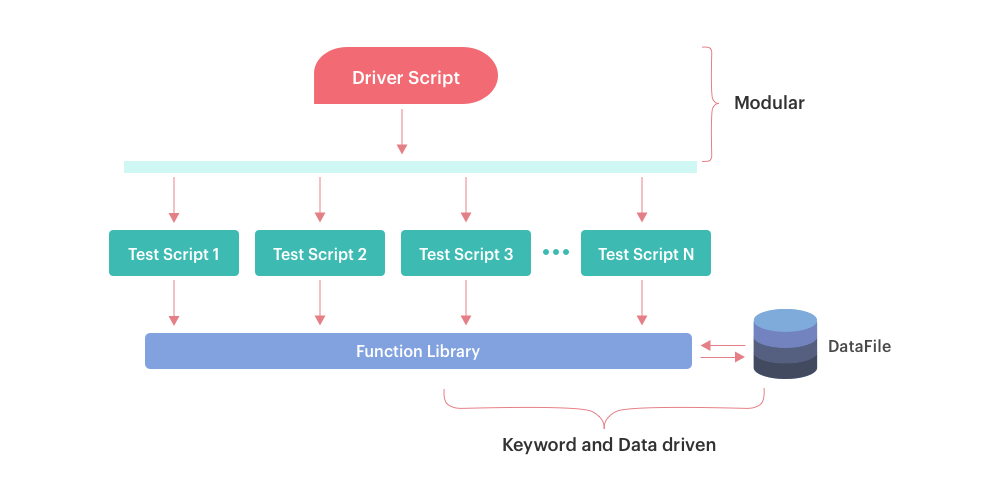
Pros:

* Independent test scripts of AUT is possible
* One keyword for multiple test scripts is used
* Code is reusable

Cons:

* Setup cost is high
* Time-consuming as the object repositories need to be setup
* QA engineer with great automation skills is required

**Hybrid Testing Framework**



It is the combination of previous frameworks that overcomes the flaws and boosts the benefits. The hybrid framework favors the flexibility for different applications; hence, it is quite good at bringing accurate test results.

Pros:

* Flexible, reusable and adaptable
* Wide community of support
* Can be integrated with CI pipeline
* Granular reporting

Cons:

* Test automation expertise is needed

## Select your tool

Automation testing largely depends on the selection of the right testing tools. The first step to selecting the right automation tool is to understand the technology the application is built on and test requirements of the Application Under Test (AUT). One of the main aspects of selecting the right automation tool is the compatibility with the tech stack of the AUT.

There are few [tools](https://www.simform.com/functional-testing-tools/) that are popular among testers; however, they might fail to address the testing demands of certain kinds of applications. One more thing, the tool must support the programming languages and testing environments your testers are most comfortable with.

A specific example is to figure out whether your application needs both functional and performance testing. For mobile apps, if you’re targeting both iOS & Android, opt for a platform that supports both – Selenium, Appium – or you may also use separate testing suits– Robotium for Android, UI Automation for iOS.

The second step is to check whether the tool has the right balance between the supported platform and the ease of use. A growing number of platforms impose the requirement of testing the app deployments on various platforms.

You must note that even in a single variant of the platform there are various versions that need to be supported. For example, if a desktop app claims to run on Windows, then it has to run on Windows 7, 10 (both 32 & 64 bit), etc.

Similarly, a mobile application could be supported on different versions of Android and iOS. Having said that, it is mandatory for the automation tool to test the application on all the platforms with a minimal change to the configuration file once the test script is written.

A must have for test automation tool is to support cross-platform testing for all the required platforms with maximum use of the test scripts.

The third aspect is to find a tool based on popularity. Popularity is proof that the tool has the availability of support, quality documentation, and technical forums. This helps you in finding the relevant testing engineers for executing and maintaining the testing tool.

There are higher chances in the way that some of the other testers might have  exposure of using the tool. For example, if we consider web app automation testing, Selenium is popular and widely used in comparison to Sahi.

The fourth aspect to take into consideration is the licensing cost of the tool. However, this isn’t as simple as comparing the pricing of the shortlisted products. You need to select the tool which has the right balance of cost and fulfillment of your testing requirements.

Firstly, there are two kinds of tools that you can choose from– open-source and commercial. Open-source tools are an attractive option; however, commercial tools offer better support and learning resources.

Yet, while choosing an open-source tool, the license agreement needs to be carefully assessed before purchasing since all come with their own caveats. There are two kinds of license associated with commercial tools:

1. Development license which allows you to write scripts on the limited number of machines
2. Runtime license which allows you to execute tests on a limited number of machines

Commercial tool necessitates you to buy both the licenses while the open-source tool provides them for free. Lastly, the two important aspects that we consider at Simform are integration with project management software & CI suite, and the availability of training resources and customer support.

In my experience, building a custom test automation framework over a free open-source tool has worked best so far. The greatest advantage is control. With the in-house automation path, what you will need is exactly what your team builds.

If you’re still not sure how to get started, you can choose both initially. Most commercial tools come with a free trial period. At the same time, you can start building the MVP of the in-house framework.

After three weeks, it should be clear whether the commercial tool will work or it is within the budget or your in-house tool is a better option. Hope this helps!

## #5. Create and run the tests

Once you’re done with creating your test automation strategy and selecting the right tools, it’s time to write and execute the scripts. From time and again, we’ve observed that making automation testing a part-time job of manual testers decreases team morale and productivity.

These are the two things we’d not suggest you to compromise with. Suggested option is to hire a dedicated automation testing team. If hiring isn’t a feasible option for you, outsourcing software testing services can be a good option.

When you start writing test cases, it is advisable to follow best practices. Here are a few of them that we rigorously followed at Simform.

* Writing test case templates in a way that they could be re-used in multiple projects. Before writing any new test cases, we make sure to check if there are similar test cases written already. This helps us in reducing the redundancy.
* Often times it happens that the one who has designed the test case isn’t the one executing it. This has encouraged us to write test cases in a concise and understandable manner.
* Over time, we’ve learned to prioritize each test case based on the feature or component involved. Practicing this helps us in ensuring that high priority cases are executed first.

## #6. Maintain scripts

Maintaining test scripts involves scrutinizing the testing parameters. For instance, when the product is rich with features, implementing a regression test may take more time and under-exploit the importance of test automation.

For such cases, maintaining test cases plays a crucial role. The test cases should be optimized and categorized in such a way that  it evaluates the subset of test cases and clearly defines the purpose of test automation.

In my experience of working with agile coaches and QA professionals, I’ve always been advised to remain focussed on maintaining the effectiveness of regression tests.

This can be achieved by periodically cleaning older test cases and categorizing the current ones. You should remain clear on your requirements and whether old test cases are effective to keep.

Test scripts should be minimal with more outcomes. If you can make smaller test cases, automation becomes easier.

For example, if you are testing an eCommerce website, you have to check events such as placing an order, selecting an item, completing the transaction, and more. Placing an order requires clicking on the button and it can be separated from the navigation test case.

Automation will not do everything automatically unless you trigger it. You will have to make it work for you by designing the test cases as per the need since it is directly related to your product delivery and ROI. Ask your QA analysts to work on subsets of test cases to save a lot of time.

***The more streamlined the test cases are the more rapidly a software can be ready for production.***

Don’t go for meandering objectives when writing a test case. Keep it concise. Keep narrowing the scope of a test case when it comes to automation. For example, when you write a script for checking whether a user can upload a profile picture or not, the script should not outperform the scope.

It means it should not be checking whether the user can successfully register or not or should not be trying to change the profile picture in order to check the upload process. You need to automate the process of checking functionalities. In order to automate other functions as well, you can better split the test cases separately.

 Over time, while implementing test automation, we have realized that in order to be successful, automation needs to be everyone’s job. We worked hard on changing the way how business analysts and testers collaborate together along with creating and running tests.

By implementing the automation to our testing process, we are able to spend more time on planning, detect more defects faster, and better meet the project demands.

For [Joe Wilson](https://www.linkedin.com/in/joewilson0/) from[Volare Systems,](https://volaresystems.com/) the end goal of automated testing is to have a decoupled software design and confidence in the quality of the software product being built. Automated testing doesn’t mean there is no manual testing. It means the important things the application used to do still work as they did before. However, there were two main challenges that they faced.



To tackle these challenges efficiently, they have used lots of tools over the years for testing. For example, for NET, they’ve used MSTest, NUnit, and XUnit, for JavaScript: Jasmine, Mocha, and Jest, for end-to-end testing: SpecFlow, Cucumber, WatiR, WatinN, Selenium, and Cypress.io.

In the next section, I am sharing the steps and considerations you can follow to create  the test automation strategy document.

[](https://www.simform.com/steps-to-improve-the-software-qa-process/)

## Test Automation Strategy Document: Steps to Follow

Software glitches cause huge disasters. Error could occur is possible while programming, but it shouldn’t be when you’re testing. To avoid these errors, we have test automation. However, to implement this, you will need a thoroughly prepared test automation strategy document.

Let’s discuss the building blocks of this automation test strategy example document.

### **#1. Test Automation Scope**

Before getting started, it’s good to have a clear picture in mind. For each module of the project, there should be timelines and milestones defined from an automation perspective. For any particular project, define a scope of it. It should keep everyone on the same page (project stakeholders, consultants, project managers, tech leads, developers, testers, etc.)

### **#2. Test automation Approach**

This is a broader term to define; hence, it can be broken into three parts: process, roles, and technology. This includes what tests to automate and the decision of when the test cases should be developed during the sprints.

During this part, you should define the roles for designing and reviewing the test cases. A well-followed approach leads you to a successful test automation.

### **#3. Risk analysis**

As the name suggests, what are the risks involved in running the automated test case? If yes, at what percentage it may happen and how to optimize it? This will keep you one step ahead of the unexpected disasters. You should consider conducting it before starting the test automation.

### **#4. Test automation environment**

Make a concrete pipeline to go through before the software is ready for production. This is how the test automation environment is defined. You need to have a clear understanding of where the data will be stored and whether it can be reused or not.

There has to be a stable and predictable test environment. For instance, for a particular project to go under the test automation, you need to plan the test cycles, prepare the test cases, and predict the testing time required.

***Test automation infrastructure should make it easy for anyone to create, maintain, run and consume your tests and test results –***[***Carlos Kidman***](https://twitter.com/CarlosKidman), Founder of QA at the Point

### **#5. Execution plan**

The execution plan is the backbone of “bug-free” testing. Run individual test cases before the regression test suite and ensure they are working great. Avoid false outcomes and test fails. It consumes a lot of additional time.

The execution plan should include day to day tasks and procedures. You can use a tool like Bamboo or Jenkins for scheduling the execution. Ask your QA Analyst to set up a test case in a fashion that brings faster feedback to the development team.

### **#6. Review & analysis**

Filter the test cases that have consumed more time and analyze them for optimization. Review the entire test automation strategy and take feedback from everyone involved i.e., peers, DevOps engineers.

Ask DevOps engineers to plan out the things that hinder the automation environment. Ensure the QA team to acknowledge the bugs in test scripts and they are resolved. Remember to document everything i.e., analysis, results, and suggestions.

[](https://www.simform.com/steps-to-improve-the-software-qa-process/)

[](https://www.simform.com/functional-testing-tools/)

[Mobile App Testing: Challenges, Types and Best Practices](https://www.simform.com/mobile-app-testing/)

[Software Testing Services](https://www.simform.com/services/software-testing/)

## Conclusion

Successful implementation of test automation is an art — the one which requires you to iterate on the go. However, the universal key is to start small and invest over a long period of time.

What if your first experiment fails? A lot of organizations get started with automating tests only to realize that their tests are unreliable or impossible to maintain. In such cases, I’ve observed companies continue to lose money, having fallen for sunk cost fallacy.

If you’re one of them, what should you do? Scrap ineffective tests. Reframe your test automation strategy. Select the new tool. Or hire a [software testing service provider](https://www.simform.com/services/software-testing/) like Simform who can help you analyze your current approach and

## Definition of Regression Packs

Automated regression tests are the core of the Test Automation Strategy.

### Smoke Regression Pack

Regression packs serve as a sanity check that the application can be loaded and accessed. Also, just a few key scenarios should also be run to make sure application is still functional.

The aim of the smoke test pack is to catch the most obvious issues, such as application not loading, or a common user flow cannot be executed; for this reason, the smoke tests should last no longer than **5 minutes** to give quick feedback in case something major is not working.

The smoke test pack runs on every deploy and can be a mixture of API and/or GUI tests.

**Functional Regression Packs**, which is meant to check the functionality of the application in more detail than the smoke test.

Multiple regression packs shall exist for different purposes. If there are multiple teams working on different sections of the application, then ideally there should be different regression packs that can be focused on the area the team is working on.

These packs should be able to run in any environment as and when required, provided the behavior of the features remain consistent throughout the environments. They are executed multiple times a day and should last no longer than 15 to 30 minutes.

As these functional tests are more detailed, then they will take longer to run therefore, it is important to have the majority of functional tests at API layer where tests can be executed faster so we could be within the **15 to 30 minutes** time limit.

**End-to-End Regression Pack,** which tests the whole application as a whole. The aim of these tests is to ensure that various parts of the application which connect to various databases and third-party applications work properly.

The End-to-End tests are not meant to test all of the functionalities as those are already tested in the functional regression packs, however, these tests are “light-weight” which just check the transitions from one state to another and a handful of the most important scenarios or user journeys.

These tests are mainly executed through the GUI, as they are checking how users would use the system. The time taken to execute these can vary from one application to another but they are usually run once a day or night.

### Automated Unit Tests

Test Automation starts at the unit level. Unit tests should be written by developers for any new feature that is developed. These Unit Tests form the foundation of a larger automation practice that spans all the way up to the System GUI Tests.

It is the responsibility of the developers to ensure that for every new feature that is developed, a set of coherent and solid Unit Tests are written to prove that the code works as intended and meets the requirements.

Unit Tests provide the most ROI to the team as they are very quick to run, easy to maintain and modify (as there are no dependencies) and when there are errors in code, it is quickly fed back to the developer.

Unit tests are run on the developer’s machine as well as the CI environment.

“We are planning to automate our Test Suite”  
  
“Provide us a Test Automation feasibility analysis”  
  
 “Initiate or expand our automation testing”  
  
Above are a few standard client requests we receive, and we start off with the usual steps/checklist for Automation feasibility and solution, as listed below :  
  
Application under Test analysis (Compatibility with testing tool, Prerequisites, Complexity)  
Tool Identification (Cost, Feasibility, Usability)  
Automation Design (Framework, Coding language, Test Reporting)  
Automation Development (Coding, Execution, Defect management)  
Delivery, Maintenance and Metrics (Automation Coverage, Test Suite Updates and ROI)  
But the optimum ROI with reduced testing time and effort is not always achieved. Even after a successful implementation of automation, the premise of the client requirement is not addressed. With the increased interoperability of devices, diverse technologies, innumerable Automation Testing Tools (licensed / open source) and various coding languages, the ability to automate everything is possible. The word “Automation Feasibility” has lost its relevance in the current world.  
  
To arrive at the crux of the problem statement, we need to ask the right questions  
  
  
Why do you need Automation?  
What is your objective for automation testing?  
Do you know if you are testing too much or too many times?  
Do you need to automate every single script?  
What is the roadmap for the Application Under Test?  
What is your current skill level?  
How much do you want to automate and how fast?  
In short, assess the Automation Suitability  
  
Evolution of Automation Tools  
  
Over the years test automation tools have become more robust and extensible, frameworks more scalable and expandable. Consider the evolution of the open source tool – Selenium. It has come a long way, from its inception where it targeted the then popular Mercury QTP (Chemical element Selenium reduces the effects of Mercury toxicity). Now Selenium is used to test across multiple platforms, browsers, applications and compatible with multiple languages and frameworks. Mercury QTP matured to HP/Microfocus UFT and its latest release 14 has LeanFT solution, which enables the user to maintain Selenium scripts.  
  
Change from competition to collaboration in the test automation space has increased the ability to integrate and automate applications rapidly in the past decade. Newer tools like TOSCA, Ranorex, etc. and frameworks like TDD, BDD have created more options and are increasing the flexibility of implementing a test automation solution in umpteen ways.  
  
 What is Automation Suitability  
  
Test automation is a vital contributor to testing efficiency. The focus should be on long-term strategic planning as well as maintenance and not just the short-term goal of automating a requirement to meet current project/release needs.  
  
Automation Suitability assessment is determined through a multi-factor analysis of the different aspects of automation testing. Some of the major factors for analysis are:  
  
Effort required to achieve estimated ROI  
Total testing scope and targeted automation coverage  
Time and effort invested in automation  
Planned / targeted cycle time for execution  
Automation tool related cost and skills available/acquirable in team  
Application under test architecture, stability and future roadmap  
Maintenance of test suite vs frequency of execution required  
Reusability in automation framework  
Robustness of automation tool selected and its support online  
Coding language and framework adopted  
Ability to integrate open source tools for extensibility  
Test data and environment challenges  
Cross-platform including mobile testing capability  
By not considering all aspects, a high percentage of automation projects fail or don't result in anticipated ROI. Other reasons can include a lack of central coordination or an operating model, unsuitable processes and improper tool usage.  
  
 World Quality Report 2017-18: WQR provides us more data points on similar problems faced by testing practices around the globe and across sectors.  
  
  
  
Agile and DevOps World:  
  
Few see Agile and DevOps as a way out and attempt in transforming to shift-left practices enabling organizations to find defects earlier. Leveraging automation testing in earlier phases is not always suitable. Agile or DevOps transformations, if done poorly, will increase the budget because of more QA issues and test cycles being needed to reach the right level of quality. TCOE’s should not be eager to move everything to DevOps and must adapt to facilitate the changing QA demands. A proper automation suitability analysis needs to be performed. In WQR 17-18 findings the Difficulty in integrating Test Automation into DevOps Process is cited by 38% of organizations.  
  
  
  
  
 Heavy Automation problems  
  
Manufacturing Industry: Like in IT, if invested heavily on total automation, the need will be to constantly have safety checks on everything and necessary monitoring supervisors. This costs money and makes it less efficient having such facilities deviating from the true purpose of automation.  
  
   
  
Optimal Automation - The Guardian Example  
  
Leading English Daily ‘The Guardian’ faced a similar problem when they realized that they needed to ship more frequently. Instead of investing heavily on Automation, the QA team made a choice of testing much less and allowing to release much more frequently. In order to mitigate the risks of their new approach they created tools that allowed anyone in the team to do push button deploys and rollbacks - which meant fixing things was relatively easy and that enabled team to find bugs very quickly if released to production. After much of technical innovation they ended up in a place where they could release to production at will and soon deploy several hundred times a day.  
  
  
Need for Automation Suitability assessment  
  
As testing technology grows, Automation Testing is always looked as the sure/safe prospect for an increase in speed, quality and reduced cost. There is an increasing need for automation suitability analysis as part of your IT transformation journey before attempting/expanding automation testing as part of any framework/model.  
  
Cigniti’s ATS (Advisory and Transformation Services) brings in deep consulting and industry expertise to assess and re-engineer your QA processes in Waterfall / Agile / DevOps. We analyze and provide an implementation roadmap, optimum tool stack recommendation including a practical operating model to help our clients improve time-to-market and cost of quality while keeping organizational and business goals in mind.

The customer experience begins when a product works. Functional testing is the way to ensure that happens. Since testing affects the overall SDLC, developers must implement functional testing as quickly and efficiently as possible. That means automation. Specifically, automated functional testing tools.  
  
So which tools should you use, and how?  
  
Automated Functional Testing Tools  
Functional testing involves quality assurance on the entire product from end-to-end. You have to replicate the live environment in which it will perform once it hits the market. That ensures that the product carries out the actions that meet your business requirements and is error-free upon release.  
  
This means testing the UI, APIs, database, security and client/server applications. These tests must be performed continually during the SDLC to ensure changes to code do not break the application. This is especially important when working within an Agile environment that prioritizes product evolution through a series of highly-efficient iterations.  
  
To ensure comprehensive coverage, we break functional testing down into four key types:  
  
Smoke testing – A preliminary test that confirms an application is stable enough to permit further testing  
Sanity testing – An end-to-end test that verifies new functionality works and that defects have been fixed  
Regression testing – Test implemented after code changes to verify that the intervention has not negatively affected the rest of the product  
Usability testing – Used to verify the user interface works per business goals  
Each task checks and re-checks the core functionality throughout the SDLC. It is time-consuming and a common source of bottlenecks during production. Automated functional testing tools are the solution.  
  
What to Consider for Functional Testing  
Functional testing ensures that your product works. Automated functional testing tools ensure that you test quickly, using the least resources possible.  
  
The variety of tools available means you can select them based on your unique product development needs. Tool selection should be based on the following three considerations:  
  
Objectives  
Ask:  
  
What are your business goals?  
What solution are you trying to deliver?   
What are your time and resource constraints?  
Your answers will inform your tool selection. One of your first choices will be open-source vs. paid tools.  
  
Selenium, for example, is a highly-popular free option that is flexible, reliable and efficient across a number of languages, platforms and browsers. As with most open-source solutions, the options for support are limited and updates happen infrequently. Selenium is also a web-only tool and has a learning curve and limited report function. That makes it difficult to use across a varied stakeholder team, as is common in Agile approaches.  
  
Paid licensed tools can cost upward of $10,000, but options such as TestComplete, Ranorex and UFT have robust support and advanced features such as AI-enhanced object identification.  
  
There is no right or wrong choice in deciding on open-source vs. paid. You just have to pick the most efficient tool for your product.  
  
Framework  
Your test framework is the foundation of your QA. The framework establishes standards and best practices to ensure comprehensive, efficient testing. The automated functional testing tools you select have to meet the requirements of your test framework. This includes languages used, object location, report conventions and more.  
  
There is enough variety among testing tools to ensure that you do not have to compromise your own test standards.  
  
Environment  
Your test environment defines how you setup and access data for testing. It is where your hardware and software meet, and it is critical to the success of your functional tests. You should look for an automated test tool that conforms to your test environment. The idea is to get a tool that suits your needs, rather than requiring changes on your part to accommodate testing.  
  
Establishing test infrastructure is one of the more expensive and complicated areas of QA testing. If you do not have the internal expertise or resources to fund and maintain such an environment, it is better to go with an outsourced QA solution.  
  
Functional testing is too important to become an afterthought or a compromise. If you do not have the professional-grade infrastructure necessary to comprehensively test your application, partner with someone who does.  
  
Maximizing QA Automation with Expert Insight  
When you partner with an external QA expert, your team gets access to engineering experience, hardware and infrastructure. Outsourcing allows you to scale up and slow production according to your release cycle without the burden of long-term infrastructure and resource liability. It also gives you someone to turn to when you encounter problems or are faced with the dilemma of choice: How do you select from all those automated test tools?  
  
Your QA partner is a tool-agnostic expert that can create an efficient, comprehensive test suite that delivers a better product faster. They have a breadth of experience to match any application with the optimal test tool. They also test for you in an Agile partnership with highly skilled development and testing engineers. \

10-Step Automation Testing Process: How to Start Automation Testing in Your Org  
  
  
Automation Testing Process: Learn how to start automation testing on your project (a step-by-step guide)  
  
  
In many organizations, quality is the first preference. If you are found to be in such an organization and still there is no formal test automation is done, you could be the person to inaugurate it.  
  
It will assist your organization to build more quality products in less time and likewise be able to market it early.  
  
=> In this third piece of the ‘Test automation tutorial series’, I will discuss what is test automation process and how to start test automation in your organization. It is significant to understand that which step is to perform first and why.  
  
Sticking with these steps will help you to introduce automation in a seamless way and allow you to avert common pitfalls which leads to automation failures.  
  
  
  
  
  
What You Will Learn:  
  
10-Step Automation Testing Process to Start Test Automation  
Step #1. Convince the Management  
Step #2. Finding Automation tool experts  
Step #3. Using the correct tool for automation  
Step #4. Analyzing various applications to determine those which are best suited for automation  
Step #5. Training the Team  
Step #6. Creating the test automation Framework  
Step #7. Developing an Execution Plan  
Step #8. Writing Scripts  
Step #9. Reporting  
Step #10. Maintenance of Scripts  
Conclusion  
Recommended Reading  
10-Step Automation Testing Process to Start Test Automation  
Here is a step-by-step test automation process and guide to help you start automation testing.  
  
Let’s start.  
  
Step #1. Convince the Management  
No matter how much you are eager to discover and initiate test automation in your organization, you cannot do anything if your management is not convinced about the benefits test automation offers. It is a universal fact that test automation is expensive. The tools are expensive (HP QTP/UFT license cost around $8K per machine). There is a cost of a test automation architect or engineer (which, by the way, are expensive too). After that, the benefits of test automation cannot be seen immediately. You have to wait 2-3 months before your scripts are prepared, tested, and that can run reliably for you to test the application.  
  
  
You have to convince the management to bear the pain of these expenses and also you have to tell them to be patient before test automation can start giving them results.  
  
So how they will be convinced? You have to tell them the cost-benefit analysis. Like you can ask questions that how much time we take to test the BAT (Build Acceptance Testing) of our application? Then you can say, if it takes a day, with test automation we can test it within 2 hours. The cost is that you have to purchase the tool, train the resource and wait for the results for two months. After two months, we will be able to run a BAT in two hours. That will save 6 hours of manual testing each time whenever a new build releases. If build is released 4 times a month. You will be able to save 24 hours or 3 days of manual testing!  
  
That doesn’t mean that manual testers will not be doing anything. They will use these 6 hours of testing to focus on new and important functionalities of the application, while automation will take care of the regression issues. This setup will overall improve the quality of product a dozen times.  
  
If your management is not willing to pay for the quality of their products, then nobody can force them to do so. They will learn automatically when clients will complain about the products. Quality affects everything. It affects your sales, it affects your relationship with clients, it affects your perception in the minds of consumers. So, intelligent management has always invested in the quality of their products.  
  
  
So five points to remember about convincing your management:  
  
Tell them about the benefits of test automation in detail.  
Tell them, that test automation is expensive and it will cost you money initially but then the cost will be reduced once scripts are prepared and start executing.  
Tell them that they have to wait for around 3 months before expecting any result from test automation.  
Tell them, that test automation is not to replace manual testers, but to aid manual testers as they will able to test more at the same time.  
Test automation does not mean more testing in less time; it means more testing at the same time. (If manual testers used to test the BAT in 8 hours, they will be able to test the BAT plus new functionality plus many other things in the same 8 hours in the presence of automation.)  
Remember, convincing your management is the first and most important step in introducing test automation in your organization. If they are not convinced, forget test automation or change your organization. :)  
  
Step #2. Finding Automation tool experts  
There are two kinds of automation experts.  
  
Automation architects  
Automation engineers  
Automation architects are a rare breed. They are hard to find, extremely expensive and extremely necessary for the success of the automation project. These people are usually responsible to build automation frameworks. (We will discuss automation frameworks in detail in a separate article)  
  
  
Automation architects are experienced in different kinds of tools and they usually know the strengths and weaknesses of each tool. They will also help the management in selecting the right tool for automation by carefully analyzing the application and technologies used in that application. They will also help to build the framework, designing the naming conventions and creating rules for scripting. They will also assist in selecting which test cases to automate first.  
  
If you are able to find a right resource for the post of automation architect, your half work is done in successful automation in your organization  
  
Automation engineers, on the other hand, are the people who will convert manual test cases into automated scripts. They will work under an automation architect and will be responsible for creating and executing scripts.  
  
Some companies hire automation engineers from outside and some companies do in-house hiring by training their existing manual testers. Whatever the case, the resource must be good in programming. He/she has to know especially about object-oriented programming. A combination of 1 automation architect and two automation engineers is great for most of the products.  
  
Step #3. Using the correct tool for automation  
This point deserves its own article (and I will write one on that). This is another difficult step in the process of starting automation. There are various tools in the market, but you have to select those which are best for your application.  
  
To make it short, I will write the most important considerations while selecting the tool. I will explain the tool selection process in detail in a separate article.  
  
The most important things to consider while selecting the right tools are:  
  
The tool must be in your budget. The automation tools are really expensive. So the company should have the budget to purchase the tool.  
The tool must support technologies used in your application. If your application is using flash or Silverlight, the tool must support it. If your application is running on mobile, the tool must be able to execute scripts on mobile. You can purchase a single tool that supports all technologies used in your application or you can purchase separate tools for each technology. For example, you can use selenium for your web applications, Robotium for your Android applications and MS Coded UI for desktop applications. Whatever the decision, this should be in your budget.  
You must have the necessary skilled resources who can use this tool or learn that tool in less time. For example, you have hired the automation architect who has only experienced in QTP, and you are purchasing a license for MS Coded UI, the resource might not be comfortable using it. Tools are like good cars, but you must have good drivers too to drive these good cars.  
The tool must have a good reporting mechanism to show the results to stakeholders after each execution.  
There are various other factors while selecting the right tool and I will cover them in a separate article.  
  
  
Read this guide for the latest top automation tools:  
  
Top 20 Best Automation Testing Tools In 2020 (Comprehensive List)  
  
  
Step #4. Analyzing various applications to determine those which are best suited for automation  
If your organization is working on 5 applications, it is not necessary that each should be automated. We have to see the various factors while selecting any application to automate.  
  
The application which should be automated must have these factors:  
  
The application should not be in the early stages of its development. (The application should have all or some modules which are stable and tested by manual testers)  
The UI of the application must be stable. (The UI must not change frequently)  
The manual test cases of this application should be in written form.  
The main goal of automation is to make sure that if the application is bug-free in one build, it should remain bug-free in the next build. The manual tester should not waste their time in finding regression issues, these issues should be identified in automation.  
  
So to find a regression, we must have an application which is already stable and has some test cases written for it. Automation team will convert these test cases into scripts and will run these scripts on every build to make sure no regression appears.  
  
Also, read => How to Select Correct Test Cases for Automation Testing  
  
Step #5. Training the Team  
After tool selection and resource hiring, the next step is logically the training of the resources.  
  
If manual testers are converted into automation engineers, they have to be trained on automation terminologies and concepts. If automation architect is hired from outside, he must get knowledge about the product to test, the manual testing process and what management is expecting.  
  
  
Give resources some time to try different things until they finally come up with a winning automation strategy. Train them on the tools which organization is already using bug tracking software and requirements management software.  
  
Good training and strong communication between manual testers, developers and automation team is really necessary.  
  
Step #6. Creating the test automation Framework  
The biggest task for the automation architect is to come up with an automation framework that should support automated testing for the long run.  
  
Automation framework is basically the set of rules and careful planning to write the scripts in a manner which results in the least amount of maintenance. If anything changes in the application, the scripts need little or no updating to cope up with that change. That is the beauty of an automation framework.  
  
There are five kinds of automation frameworks, namely linear, modular, data-driven, keyword-driven and hybrid. All of these frameworks will be covered in detail with examples in a separate article in this series.  
  
You can also start reading more on automation frameworks in following tutorials:  
  
=> Why Do We Need Framework for Test Automation?  
=> QTP Framework examples  
=> Selenium Framework examples  
  
Step #7. Developing an Execution Plan  
The execution plan includes selecting which environments the scripts will be executed. The environment includes OS, Browser and different hardware configurations.  
  
For example, if the test case demands that it should check the website in 3 browsers, namely, Chrome, Firefox and IE, then the automation team will write the script in such a manner that it will be able to execute in each browser.  
  
This should always be told before writing the scripts because it will be taken care in scripts if the automation team know it beforehand. The execution plan should also state that who will execute the scripts. Normally the automation team executes the scripts on every build, but it varies from company to company. Some managers ask developers to execute these scripts on their build before release and some companies hire a dedicated resource just for the execution. Even some companies run scripts in unattended mode, which of course requires no additional resource.  
  
Step #8. Writing Scripts  
When the framework is designed, the execution plan is known and resources are trained on the new tool, now it’s the right time to start writing scripts.  
  
Scripts should be written in an organized manner with proper naming convention. The source code should be maintained in a source control to avoid code loss. Version control and history should be maintained. Test automation is just like software development. All best programming practices should be taken care while writing the scripts.  
  
Also, read => How to Translate Manual Test Cases into Automation Scripts  
  
Step #9. Reporting  
The reporting feature is usually provided by the tool. But we can create custom reporting mechanisms like auto-emailing the results to management.  
  
We can create reports at the end of each execution in the form of charts and tables if management needs it. The management should always be informed about the test case coverage, that means which manual test cases are covered in automation and which of them are remaining.  
  
Step #10. Maintenance of Scripts  
If best programming practices are followed and framework is good, then maintenance will not be a problem.  
  
Maintenance usually occurs when there is a change request an application. The scripts should immediately be updated to cope with that change to ensure flawless execution.  
  
For example, if you are writing some text in the textbox through the script and now this text box becomes the drop-down list, we should immediately update the script.  
  
Some other kinds of changes include that your scripts were running on the English version of the application. Now there is a change request that the application should support Chinese. Your framework should allow you to update your scripts with little effort to support execution in Chinese too! That is why Automation architects are expensive. :)  
  
If the framework is not good and best practices are not followed, then maintenance will become a nightmare. Most automation projects fail due to poor maintenance of scripts.  
  
Conclusion  
This article describes what is automation testing process and how to start automation testing in your organization from start to end in a step by step manner. If you follow these steps, I hope your automation will be a success.  
  
There are some parts (like Automation tool selection and Automation Frameworks) which deserve their own articles. We will cover these in upcoming parts of this automation testing tutorial series.  
  
=> Meanwhile click here to check all the tutorials we already posted in this series.  
  
I tried to cover all aspects in a broader view and use my own experience to write this tutorial.  
  
If you feel that I missed something important or some portion of this tutorial needs a little more explanation, please ask me in the comments section. I would love to answer your queries.  
  
PREV Tutorial #2 | NEXT Tutorial #4  
  
Recommended Reading  
Step by Step Guide to Implement Proof of Concept (POC) in Automation Testing  
What is Automation Testing (Ultimate Guide to Start Test Automation)  
Sikuli GUI Automation Testing Tool - Beginner's Guide Part #2  
Best Software Testing Tools 2020 [QA Test Automation Tools]  
Are Testers Losing Their Grip Over Testing Due to Automation?  
Manual and Automation Testing Challenges  
Are You a Manual or Automation Testing Expert? Work Part Time For Us!  
11 Best Automation Tools for Testing Android Applications (Android App Testing Tools)  
Last Updated: September 1, 2020 12:19 pm  
  
Next Read: How to Choose the Best Automation Testing Tool (A Complete Guide) »

## he Test Automation Checklist

A strong automation checklist template in place is key within any successful test automation strategy. From an automation test plan template for Selenium to a solid automation testing checklist, any action that can be clearly defined and measurable should be included within your QA checklist.

Incorporate these [QA automation testing tips for faster product releases](https://blog.qasource.com/qa-automation-testing-tips-for-faster-product-releases) by applying the following test automation assessment checklist to your strategy.

* **Kickoff Meeting**: Schedule a meeting with your automation experts, development engineers and stakeholders to discuss the purpose, needs and plans for test automation. What requirements are essential for your test automation strategy to succeed? [How can automation add value to your testing](https://blog.qasource.com/how-automation-adds-value-to-your-testing)? In what ways can your team [Reduce your automation backlog](https://info.qasource.com/test-automation-checklist-to-reduce-test-automation-backlog) so that the project stays on track?
* **Requirements Collection**: After defining the goals, needs and plan of your test automation strategy, set deadlines for your automation team to gather critical requirements for the project. This will enable your team to implement the most appropriate test automation methods. Requirements that should be collected during this process should include:
  + Technology used for application development
  + Product & domain knowledge (including functional specifications and product roadmap)
  + Product architecture (including any specific back end hooks like APIs, Web Services or DB connectivity hooks)
  + Automation tool preferences
  + List of any challenges faced during any earlier efforts of automation
* **List of Use Cases**: Ask your developers to provide a list of test cases and their priorities. In other words, how critical is each test case in ensuring that the product is ready for consumer use? While every test case provides value during the QA process, specific test cases executed earlier than others allow developers more time to fix any uncovered (and possibly expected) issue. With this prioritized list,  the automation team can decide early on about the framework strategy and tool selection.
* **Test Automation Strategy Methodologies**: Require your automation team to identify how the tests should be prepared, what test language to use, the test interface and the test data (inputs/outputs). Should your test automation strategy include record and replay, functional test automation, data-driven test automation, keyword-driven test automation or hybrid test automation? Defining this ensures a maintainable and portable automation solution.

[**QA Team Project Management Worksheet**](https://blog.qasource.com/cs/c/?cta_guid=b6c75d8b-bee6-4354-9623-6420f7e1f039&signature=AAH58kE79Nr9MPgS5RJ5oDuSNUpAY3Q2cA&pageId=3716952478&placement_guid=a318a785-c369-43a3-8e6f-540669b90c20&click=950b8a96-d8d5-4b43-aed4-5fd91a9e559c&hsutk=bd318c15af5921559dc35a7cb565f1d0&canon=https%3A%2F%2Fblog.qasource.com%2Ftest-automation-checklist%2F&utm_referrer=https%3A%2F%2Fwww.google.com%2F&portal_id=1751973&redirect_url=APefjpE_4axkrWSIh170KCn4_CgFuC7jOk2jYQqUKBpAFVzE4BCSIFZK7cgN1rUMThvZ-weoamHXwSn8h372hdtBem76pXKhrYcys3NoEeExLi9b0nImd7yW5GP-2H2sev-OKuJ_q6hnNpX3FkzZNA9rRaIu8UE0Jw&__hstc=227323074.bd318c15af5921559dc35a7cb565f1d0.1602998336793.1602998336793.1602998336793.1&__hssc=227323074.1.1602998336794&__hsfp=2673779826&contentType=blog-post)

* **Test Automation Tool Evaluation Checklist**: Examine both paid and open-source automation tools to verify which testing tools best suit your application and your QA team. Not sure where to start? For step-by-step guidance on choosing automation tools, you can download our [Guide to selecting the best automation tool](https://info.qasource.com/a-guide-to-selecting-the-best-test-automation-tool).
* **Sign-Off**: Review the use cases so that the automation team can propose an effort estimate for sign-off. Having a timeline (and a deadline) will help all your team members stay on track with their required deliverables. Keep in mind that DevOps upholds the standard that everyone is responsible for any released defects, so DevOps teams should include within their QA checklist everyone required for sign-offs as well as realistic time estimates.
* **Basic Framework**: Develop the basic framework required for automating use cases. In other words, design an integrated system that sets the rules of test automation for your software product so that test results are accurate, consistent and accessible to everyone on the team. To do this, your automation team should ask:
  + Is the code being developed scalable?
  + Is the code being developed logical?
  + Is the code being developed coherent?
  + Is the code being developed reusable?
  + Is the code being developed configurable?
  + Is the code being developed maintainable?
* **Test Script Creation**: Automate the use cases according to the previously established goals, carefully identifying the elements, creating the data, documenting the scripts and verifying.
* **Batch Execution & Analysis**: In conjunction with test script creation, execute the scripts in batch so that the automation team can verify accurate interaction with the application and ensure no unexpected test script failures are related to synchronization issues.
* **Demo & Reporting**: Once the suite is completed, present the final framework, automated tests and reports. This step may include the automation team training outside stakeholders to ensure future set-up and executions will be completed successfully.

During the test automation process, the team leads are updated with daily and weekly status reports. We also hold weekly conference call meetings which are used to discuss the team’s progress and any issues/challenges that need to be addressed before moving on. Consistent communication and structure creates an environment where maintainable, portable and extensible automation suites are built. Say goodbye to endless updates and maintenance. Plan ahead and build a test automation solution that works for the duration of your product development life cycle.

## What Else Should You Include in Your Automation Checklist Template?

During the test automation process, the team leads are updated with daily and weekly status reports. Expect to hold weekly conference call meetings to discuss the team’s progress and any issues/challenges worth addressing. Plan ahead and build a test automation tool evaluation checklist that works for the duration of your product development life cycle.

## How Do I Start Implementing a Test Automation Checklist?

A test automation strategy can only be as successful as the automation script review checklist and QA checklist in place. Without following a structured Selenium code review checklist during your development cycle, your software product is more prone to defects and code issues not detected until market release, especially when you don’t [Choose the right automated functional testing tools for your product](https://blog.qasource.com/resources/choose-the-right-automated-functional-testing-tools-for-your-product).

Ensure your test automation strategy is a success by partnering with a reliable QA service provider like QASource. [Our team of testing experts](https://www.qasource.com/testing-experts) specializes in all areas of [Automation testing](https://www.qasource.com/automation-testing-services) and can help your company structure a test automation assessment checklist within your development practices so that your product is always released error-free to your consumers. [Get a free quote today](https://info.qasource.com/request-a-free-quote).

|  |
| --- |
|  |
| **Test Readiness Review (TRR) Criteria** | **Status** |
| All the Requirements finalized and analyzed | Done |
| [Test Plan](https://www.softwaretestinghelp.com/test-plan-sample-softwaretesting-and-quality-assurance-templates/) created and reviewed | Done |
| [Test Cases preparation](https://www.softwaretestinghelp.com/how-to-write-effective-test-cases-test-cases-procedures-and-definitions/) done |  |
| Test Case review and sign off |  |
| [Test Data](https://www.softwaretestinghelp.com/tips-to-design-test-data-before-executing-your-test-cases/) availability |  |
| [Smoke Testing](https://www.softwaretestinghelp.com/smoke-testing-and-sanity-testing-difference/) |  |
| Is Sanity Testing done? |  |
|  |  |
| Team aware of the roles and responsibilities |  |
| Team aware of the deliverables expected of them |  |
| Team aware of the [Communication protocol](https://www.softwaretestinghelp.com/how-to-write-effective-emails-to-qa-team/) |  |
| Team’s access to the application, version control tools, [Test Management](https://www.softwaretestinghelp.com/category/test-management-tools/) |  |
| Team’s trained |  |
|  |  |
| Technical Aspects- Server1 refreshed or not? |  |
| Defect reporting standards are defined |  |

|  |
| --- |
|  |
| **Exit Criteria** | **Status** |
| 100% Test Scripts executed | Done |
| 95% pass rate of Test Scripts |  |
| No open Critical and High severity defects |  |
| 95% of Medium severity defects have been closed |  |
| All remaining defects are either canceled or documented as Change Requests for a future release |  |
| All expected and actual results are captured and documented with the test script | Done |
| All test metrics are collected based on reports from [HP ALM](https://www.softwaretestinghelp.com/learn-hp-quality-center-qc-in-4-days/) |  |
| All defects are logged in HP ALM | Done |
| Test Closure Memo is completed and signed off |  |

**Testing Checklist:**

1. Create System and Acceptance Tests [ ]
2. Start Acceptance Test Creation [ ]
3. Identify Test team [ ]
4. Create Workplan [ ]
5. Create Test Approach [ ]
6. Link Acceptance Criteria and Requirements to form the basis of Acceptance Test [ ]
7. Use a subset of system test cases to form requirements portion of Acceptance Test [ ]
8. Create scripts for use by the customer to demonstrate that the system meets requirements [ ]
9. Create a Test schedule. Include people and all other resources. [ ]
10. Conduct Acceptance Test [ ]
11. Start System Test Creation [ ]
12. Identify test team members [ ]
13. Create Workplan [ ]
14. Determine Resource Requirements [ ]
15. Identify productivity tools for testing [ ]
16. Determine Data Requirements [ ]
17. Reach an agreement with Data Center [ ]
18. Create Test Approach [ ]
19. Identify any facilities that are needed [ ]
20. Obtain and review existing test material [ ]
21. Create an inventory of test items [ ]
22. Identify Design states, conditions, processes, and procedures [ ]
23. Determine the need for Code-based (white box) testing. Identify conditions. [ ]
24. Identify all functional requirements [ ]
25. End inventory creation [ ]
26. Start Test Case creation [ ]
27. Create Test Cases based on the inventory of test items [ ]
28. Identify logical groups of business function for the new system [ ]
29. Divide test cases into functional groups traced to test item inventory [ ]
30. Design data sets to correspond to test cases [ ]
31. End Test Case creation [ ]
32. Review business functions, test cases, and data sets with users [ ]
33. Get signoff on test design from Project leader and QA [ ]
34. End Test Design [ ]
35. Begin Test Preparation [ ]
36. Obtain Test Support resources [ ]
37. Outline expected results for each test case [ ]
38. Obtain Test Data. Validate and trace to test cases [ ]
39. Prepare detailed Test Scripts for each test case [ ]
40. Prepare & Document environmental setup procedures. Include back up and recovery plans [ ]
41. End Test Preparation phase [ ]
42. Conduct System Test [ ]
43. Execute Test Scripts [ ]
44. Compare the actual result to expected [ ]
45. Document discrepancies and create problem report [ ]
46. Prepare maintenance phase input [ ]
47. Re-execute test group after problem repairs [ ]
48. Create a final test report, include known bugs list [ ]
49. Obtain formal signoff [ ]

### **Automation Checklist**

If you answer yes to any of these questions, then your test should be seriously considered for Automation.

**Q #1) Can the test sequence of actions be defined?**

**Answer:** Is it useful to repeat the sequence of actions many times? Examples of this would be Acceptance tests, Compatibility tests, Performance tests, and regression tests.

**Q #2) Is it possible to Automate the sequence of actions?**

**Answer:** This may determine that automation is not suitable for this sequence of actions.

**Q #3) Is it possible to “semi-automate” a test?**

**Answer:**Automating portions of a test can speed up test execution time.

**Q #4) Is the behavior of the software under test the same with automation as without?**

**Answer:**This is an important concern for Performance Testing.

**Q #5) Are you testing non-UI aspects of the program?**

**Answer:**Almost all non-UI functions can and should be automated tests.

**Q #6) Do you need to run the same tests on multiple hardware configurations?**

**Answer:**Run ad-hoc tests (Note: Ideally every bug should have an associated test case. Ad hoc tests are best done manually. You should try to imagine yourself in real-world situations and use your software as your customer would. As bugs are found during ad-hoc testing, new test cases should be created so that they can be reproduced easily and so that regression tests can be performed when you get to the Zero Bug Build phase.)

The testing stage at the top of the pyramid has the smallest number of test cases. These test cases execute more slowly than the test cases at any other stage, but they also cover the most comprehensive functionalities. They are the hardest to automate and the least resilient.



When we take a look at how the testing pyramid fits into the testing cycle, we can break it down into the following testing stages:

* [Unit Testing](https://www.theactframework.com/under-construction?hsLang=en): Designed to validate the behavior of individual functions, methods, and objects—and the interaction between them—within the application.
* [System Testing](https://www.theactframework.com/under-construction?hsLang=en): Designed to validate the functionality of the application or system in an isolated environment.
* [System Integration Testing](https://www.theactframework.com/under-construction?hsLang=en): Designed to validate the integration between systems.
* [End-to-End Testing](https://www.theactframework.com/under-construction?hsLang=en): Designed to validate complete business processes in a production-like environment.
* [User Acceptance Testing](https://www.theactframework.com/under-construction?hsLang=en): Designed to receive feedback and acceptance from the business on functionality as well as user experience.
* [Production Testing](https://www.theactframework.com/under-construction?hsLang=en): Designed to validate proper behavior in production during release.

The testing cycle as illustrated above reflects the ideal scenario for a large enterprise organizations. In smaller organizations, the [system integration](https://www.theactframework.com/under-construction?hsLang=en) and [end-to-end testing](https://www.theactframework.com/under-construction?hsLang=en) stages may be combined into a single testing stage.

