

Automation Testing Test Plan Sections

**Section #1: Scope**

* Choose the Test cases/scenarios that are to be regressed over and over across multiple cycles.
* Sometimes the simplest of Test cases need lots of complicated solutions to be automated. If these are just for a one time use, it obviously does not make sense. Reusability should be your focus.
* Automation Testing does not/cannot perform Exploratory testing.

**Section #2:**[**Test strategy**](https://www.softwaretestinghelp.com/writing-test-strategy-document-template/)

* This section is referred to as the Framework in the Automation world.  Some frameworks are extremely challenging to create and also are effective – but time, effort and competency wise they are demanding.  Always look for a middle ground and do the best you can without jeopardizing overutilization of resources.
* Decide on coding best practices to be used, naming conventions, locations for test assets to be stored, the format of test results , etc. to maintain uniformity and increase productivity.

**Section #3: Resources/Roles and Responsibilities**

* The first step in this direction is to understand the team’s capabilities and anticipate ahead of the scope of Automation coming into the picture. This will help choose a team that suits both the Automation and Manual testing needs. Also, pick people who have the right attitude – those do not think that Manual testing is beneath their stature.
* Choose a team well versed with AUT, Test Management, Defect Management, and other SDLC activities
* Section #1: Scope

**Section #4: Tools**

Pick Automation tools based on the following rules:

* Does the company already have licenses for a certain tool, try and see if you can use it
* Look for open-source(but reliable) tools
* Do the team members know the tool already or do we need to bring in someone new? Or train the existing ones?

**Section #5: Schedules**

* Include time for code-walkthroughs and inspection of the Automation scripts
* Maintain the scripts on a timely basis. If you create a piece of code that you are not going to use for the next 6 months or so, make sure to periodically maintain it to lessen its chances of failure.

**Section #6: Environment**

* The target environment that your AUT is going to run and the Automation tool that you want to use should be compatible. This is one of the factors to be considered pre-licensing for the tool.
* Also, analyze if the rest of the [Management Tools](https://www.softwaretestinghelp.com/category/test-management-tools/) in place and the Automation tool you are trying to bring in are inter-connectible for additional benefit.

**Section #7: Deliverables**

* Your Test scripts are your deliverables. However, not everyone is automation/programming language savvy. So, plan on creating a “How-to” document that will help the current users and future team members to be able to understand this script even when you are not around.
* Include comments in your script too.

**Section #8:**[**Risks**](https://www.softwaretestinghelp.com/types-of-risks-in-software-projects/)

If you are going to propose an automation solution, be sure to choose cost-effective tools and solutions to make sure that the Automation endeavor does not burden the project.

It is important to set the expectation that ROI for an Automation project cannot be positive immediately but can be clearly seen over long periods of time.

Therefore, if you propose automating a system, pick the one that is

* Stable and not too much maintenance
* Has scope for huge regression suites
* Does not have too much of manual intervention or does not depend on a human’s intuition

**Section #9: Test data**

* Take into consideration the security aspects of the data
* Do not hard code any test data into the scripts. This just leads to too much script maintenance and might induce errors during modification.
* Be very specific. For a Manual test step – ‘enter the first name’, you can say enter any 5 character name. While testing, a tester can type “Swati” or “Seela” or anything else. But for a tool, it can’t make such suppositions. Therefore, provide exact values.

**Section #10: Reports/Results**

* Script execution results are also technical and might not be easily understood by the rest of the teams. Plan on writing detailed results to notepad or excel sheets as an additional measure.
* Detailed framework documents, review results, defect reports, execution status reports are also expected.

We, as Automation enthusiasts might think that clients/management don’t easily buy the automation proposals.

However, when our ultimate goal is to maximize ROI through Automation, we are in perfect harmony with the management/client’s goals too. **This will ensure that we not only get to Automate our project but will be able to do so, with lots of consent, co-operation, and excitement.**

Planning and thorough analysis of all the factors listed above can be our ally through this journey. Again, ROI means everything.

**Best Practices**

Use these top tips to ensure that your software testing is successful and you get the maximum return on investment (ROI):

1. Decide what Test Cases to Automate
2. Test Early and Test Often
3. Select the Right Automated Testing Tool
4. Divide your Automated Testing Efforts
5. Create Good, Quality Test Data
6. Create Automated Tests that are Resistant to Changes in the UI

## **Decide What Test Cases to Automate**

It is impossible to automate all testing, so it is important to determine what test cases should be automated first.

The benefit of automated testing is linked to how many times a given test can be repeated. Tests that are only performed a few times are better left for manual testing. Good test cases for automation are ones that are run frequently and require large amounts of data to perform the same action.

You can get the most benefit out of your automated testing efforts by automating:

* Repetitive tests that run for multiple builds.
* Tests that tend to cause human error.
* Tests that require multiple data sets.
* Frequently used functionality that introduces high risk conditions.
* Tests that are impossible to perform manually.
* Tests that run on several different hardware or software platforms and configurations.
* Tests that take a lot of effort and time when manual testing.

Success in test automation requires careful planning and design work. Start out by creating an automation plan. This allows you to identify the initial set of tests to automate, and serve as a guide for future tests. First, you should define your goal for automated testing and determine which types of tests to automate. There are a few different [types of testing](https://smartbear.com/learn/automated-testing/), and each has its place in the testing process. For instance, unit testing is used to test a small part of the intended application. To test a certain piece of the application’s UI, you would use functional or [GUI testing](http://smartbear.com/product/testcomplete/features/object-based-recording).

After determining your goal and which types of tests to automate, you should decide what actions your automated tests will perform. Don’t just create test steps that test various aspects of the application’s behavior at one time. Large, complex automated tests are difficult to edit and debug. It is best to divide your tests into several logical, smaller tests. It makes your test environment more coherent and manageable and allows you to share test code, test data and processes. You will get more opportunities to update your automated tests just by adding small tests that address new functionality. Test the functionality of your application as you add it, rather than waiting until the whole feature is implemented.

When creating tests, try to keep them small and focused on one objective. For example, separate tests for read-only versus read/write tests. This allows you to use these individual tests repeatedly without including them in every automated test.

Once you [create several simple automated tests](https://support.smartbear.com/testcomplete/docs/tutorials/getting-started/first-test/index.html), you can group your tests into one, larger automated test. You can organize automated tests by the application’s functional area, major/minor division in the application, common functions or a base set of test data. If an automated test refers to other tests, you may need to create a test tree, where you can run tests in a specific order.

## **Test Early and Test Often**

To get the most out of your automated testing, testing should be started as early as possible and ran as often as needed. The earlier testers get involved in the life cycle of the project the better, and the more you test, the more bugs you find. Automated [unit testing](https://smartbear.com/learn/automated-testing/what-is-unit-testing/) can be implemented on day one and then you can gradually build your automated test suite. Bugs detected early are a lot cheaper to fix than those discovered later in production or deployment.

With the [shift left movement](https://smartbear.com/learn/automated-testing/shifting-left-in-testing/), developers and advanced testers are now empowered to build and run tests. Tools such as TestLeft allows users to run functional UI tests for web and desktop applications from within their favorite IDEs. With support for Visual Studio and Java IDEs such as IntelliJ and Eclipse, developers never have to leave the comfort of their ecosystem to validate application quality - meaning teams can quickly and easily shift left to deliver software faster.

### Start Shifting Left and Automate now with ****TestLeft****

[Start Free Trial](https://smartbear.com/product/testleft/free-trial)

## **Select the Right Automated Testing Tool**

Selecting an automated testing tool is essential for test automation. There are a lot of automated testing tools on the market, and it is important to choose the automated testing tool that best suits your overall requirements.

Consider these key points when selecting an automated testing tool:

* Support for your platforms and technology. Are you testing .Net, C# or WPF applications and on what operating systems? Are you going to test web applications? Do you need support for mobile application testing? Do you work with Android or iOS, or do you work with both operating systems?
* Flexibility for testers of all skill levels. Can your QA department write [automated test scripts](http://smartbear.com/product/testcomplete/features/script-editor-and-debugger/) or is there a need for [keyword testing](http://smartbear.com/product/testcomplete/features/keyword-testing/)?
* Feature rich but also easy to create automated tests. Does the automated testing tool[support record-and-playback test creation](http://smartbear.com/product/testcomplete/features/object-based-recording/) as well as manual creation of automated tests; does it include features for [implementing checkpoints](http://smartbear.com/product/testcomplete/features/checkpoints/) to verify values, databases, or key functionality of your application?
* Create automated tests that are reusable, maintainable and resistant to changes in the applications UI. Will my automated tests break if my UI changes?

For detailed information about selecting automated testing tools for automated testing, see [Selecting Automated Testing Tools](https://smartbear.com/learn/automated-testing/selecting-automated-testing-tools/).

## **Divide Your Automated Testing Efforts**

Usually, the creation of different tests is based on the QA engineers’ skill levels. It is important to identify the level of experience and skills for each of your team members and divide your automated testing efforts accordingly. For instance, writing automated test scripts requires expert knowledge of scripting languages. Thus, in order to perform these tasks, you should have QA engineers that know the script language provided by the automated testing tool.

Some team members may not be versed in writing automated test scripts. These QA engineers may be better at writing test cases. It is better when an automated testing tool has a way to create automated tests that do not require an in-depth knowledge of scripting languages, like TestComplete’s keyword tests feature. A keyword test (also known as keyword-driven testing) is a simple series of keywords with a specified action. With keyword tests, you can simulate keystrokes, click buttons, select menu items, call object methods and properties, and do a lot more. Keyword tests are often seen as an alternative to automated test scripts. Unlike scripts, they can be easily used by technical and non-technical users and allow users of all levels to create robust and powerful automated tests.

You should also collaborate on your automated testing project with other QA engineers in your department. Testing performed by a team is more effective for finding defects and the right automated testing tool allows you to share your projects with several testers.

## **Create Good, Quality Test Data**

Good test data is extremely useful for data-driven testing. The data that should be entered into input fields during an automated test is usually stored in an external file. This data might be read from a database or any other data source like text or XML files, Excel sheets, and database tables. A good automated testing tool actually understands the contents of the data files and iterates over the contents in the automated test. Using external data makes your automated tests reusable and easier to maintain. To add different testing scenarios, the data files can be easily extended with new data without needing to edit the actual automated test.

Typically, you create test data manually and then save it to the desired data storage. However, TestComplete provides you with the Data Generator that assists you in creating Table variables and Excel files that store test data. This approach lets you generate data of the desired type (integer numbers, strings, boolean values and so on) and automatically save this data to the specified variable or file. Using this feature, you decrease the time spent on preparing test data for data-driven tests. For more information on generating test data with TestComplete, see the Using Data Generators section in TestComplete’s help.

Creating test data for your automated tests is boring, but you should invest time and effort into creating data that is well structured. With good test data available, writing automated tests becomes a lot easier. The earlier you create good-quality data, the easier it is to extend existing automated tests along with the application's development.

## **Create Automated Tests That Are Resistant to Changes in the UI**

Automated tests created with scripts or keyword tests are dependent on the application under test. The user interface of the application may change between builds, especially in the early stages. These changes may affect the test results, or your automated tests may no longer work with future versions of the application. The problem is automated testing tools use a series of properties to identify and locate an object. Sometimes a testing tool relies on location coordinates to find the object. For instance, if the control caption or its location has changed, the automated test will no longer be able to find the object when it runs and will fail. To run the automated test successfully, you may need to replace old names with new ones in the entire project, before running the test against the new version of the application. However, if you provide unique names for your controls, it makes your automated tests resistant to these UI changes and ensures that your automated tests work without having to make changes to the test itself. This also eliminates the automated testing tool from relying on location coordinates to find the control, which is less stable and breaks easily.

### Create an Automated Test Now with ****TestComplete****

[Start Free Trial](https://smartbear.com/product/testcomplete/free-trial)

## **Conclusion**

The best practices described in this article are the path to successful test automation implementation. TestComplete includes a number of features that help you follow these best practices:

* With [TestComplete](http://smartbear.com/product/testcomplete/overview) you can perform different types of software testing:
  + [Functional Testing](https://smartbear.com/learn/automated-testing/introduction-to-functional-testing/)
  + [Unit Testing](https://smartbear.com/learn/automated-testing/what-is-unit-testing/)
  + [Keyword-Driven Testing](https://smartbear.com/learn/automated-testing/benefits-of-keyword-testing/)
  + [Data-Driven Testing](https://smartbear.com/learn/automated-testing/introduction-to-data-driven-testing/)
  + [Regression Testing](https://smartbear.com/learn/automated-testing/what-is-regression-testing/)
  + [Distributed Testing](https://smartbear.com/products/qa-tools/automated-testing/supported-testing-types/distributed-testing/)
  + [Coverage Testing](https://smartbear.com/products/qa-tools/automated-testing/supported-testing-types/coverage-testing/)
  + [Object-Driven Testing](https://smartbear.com/products/qa-tools/automated-testing/supported-testing-types/object-driven-testing/)
  + [Web Testing](https://smartbear.com/products/qa-tools/automated-testing/supported-testing-types/web-testing/)
  + [Manual Testing](https://smartbear.com/products/qa-tools/automated-testing/supported-testing-types/manual-testing/)
* TestComplete allows you to divide your test into individual test parts, called test items test items, and organize them in a tree-like structure. It lets you repeatedly use individual tests and run them in a certain order.
* TestComplete supports keyword-driven testing. These automated tests can be easily created by inexperienced TestComplete users or when a simple test needs to be created quickly.
* TestComplete supports five scripting languages that can be used for creating automated test scripts: VBScript, JScript, DelphiScript C++Script and C#Script.
* With TestComplete, QA engineers can share a test project with their team.
* TestComplete offers a **Name Mapping** feature that allows you to create unique names for processes, windows, controls and other objects. It makes your object names and tests clearer and easier to understand, as well as, independent of all object properties and less prone to errors if the UI changes. This feature allows you to test your application successfully even in the early stages of the applications life cycle when the GUI changes often.
* There are a lot of other features that TestComplete provides to help you get started quickly with your automated testing.

Adapting these recommended best practices and using TestComplete’s features can help you avoid common mistakes and improve your automated testing process. This helps you test faster, save money and get your products released on time. If you haven’t tried TestComplete, [download and try it today for 30 days](http://smartbear.com/product/testcomplete/free-trial/).

Automation Life Cycle

In automation testing life cycle methodology, test design is constructed to portray test efforts, to give project and test team a framework on the scope of the test program.

* Determining The Scope Of Test Automation
* Selecting The Right Tool For Automation
* Test Plan + Test Design + Test Strategy
* Setting Up The Test Environment
* Automation Test Script Development + Execution
* Analysis + Generation Of Test Reports

### Determining The Scope Of Test Automation

It is the first stage of automation testing life cycle and it aims to identify the feasibility of automation. Every aspect should be considered while analyzing the feasibility.

Also, it is essential to perform a feasibility analysis on the manual test case pack that allows automation engineers to design the test scripts.

In this particular stage, the following things should be taken care of without a failure.

1. Which modules of the applications can be automated and which not?
2. Which test can be automated and how to automate them?
3. Factors like cost, team size and expertise should also be considered.

Following feasibility checks should be done before starting the test automation:

* Test Case Automation Feasibility
* AUT Automation Feasibility

Navigate through the application screens under test and mention all UI components of the application to perform feasibility analysis in an effective way. Identify a percentage of UI components to be automated via an automation testing tool.

Try to find out automation testing tools that can help to automate UI components with few alterations. Which brings us to our next phase!

### Selecting The Right Automation Tool

Automation testing is highly tool dependent. That is why finding the right automation testing tool is a critical phase for an automation testing life cycle. When you are looking for an automation tool you need to keep in mind the budget, technologies being used in the project, familiarity of the tool with resources on-board, intuitiveness, flexibility and more. Choose a tool that provides you with a support team who can take care of queries or issues.

For example, if you are looking for an automated [browser compatibility testing tool](https://www.lambdatest.com/feature) then you need to keep in mind the variety of browsers offered. Capability of capturing video logs, metadata of automation scripts among different browsers and devices. A mechanism for highlighting and tracking bugs.

LambdaTest comes as the right fit for automation testing, bringing more than 2000 browsers for users to test upon. LambdaTest offers a [cloud-based Selenium Grid](https://www.lambdatest.com/selenium-automation) which is compatible with every test automation framework that supports Selenium. Automation dashboard offered by LambdaTest captures even the minute details of your test execution.Ultimately providing you with step-by-step screenshots, video logs of your test execution, along with the metadata. LambdaTest experts offer 24×7 chat support to resolve any of your concern ASAP.

### Test Plan + Test Design + Test Strategy

It is the most critical phase of automation testing life cycle methodology that defines how to approach and accomplish the goal of test automation. Selecting a test automation framework is the first and foremost thing to do in the Test Strategy phase of Automation Testing Life Cycle.

Selecting a tool depends on the technologies used in the application. Understand your product completely before starting the automation test.

For example, if it is a desktop application, find which language it is built upon. Or, if you want to test a web application, know about the deprecated feature your used which may not be compatible across various browsers.

During the test planning phase, the testing team decides the test procedure creation standards and guidelines; hardware; software and network to support test environment; a preliminary test schedule; test data requirements; defect tracking procedure and associated tracking tool and a procedure to control test configuration and staging environments.

The team of test engineers develops a test architecture to describe the test program structure and the way test procedures are managed after the test program model is designed.

Post designing comes the test architecture where the structure of the entire test program is described along with the management of test procedure.

Make sure to consider the following things when planning a [test management strategy](https://www.lambdatest.com/blog/why-you-need-to-understand-test-management-strategy-to-become-pro/):

1. Gather all manual test cases from the test management tool to identify which test case needs to be automated.
2. Identify which framework to be used after understanding the pros and cons of the testing tool.
3. Build a test suite for Automation test case in the tool for test management.
4. Ensure to mention background, risk, background and dependency between the tool and application in the test plan.
5. Seek approval on test strategy from clients or stakeholders.

### Setting Up The Test Environment

As the name indicates, this stage of Automation Testing Life Cycle involves setting up a machine or remote machine where test cases will be executed. Why would we need remote machines? Because unless we live in an ideal world, your users would be using different machines to access your website or web-app on the internet.

Keeping a check on different devices is one thing, but we also need to be cautious about various browsers and browser versions. As your website may render differently from one browser to another. Cross browser compatibility testing also known as [cross browser testing](https://www.lambdatest.com/) is a procedure where we test a website or a web-app across multiple browser versions to make sure we deliver a seamless user experience to all our customers.

Environment setup phase needs a thorough planning, you need to ensure that you are able to maximize your test coverage across as many different scenarios as possible. It is the responsibility of test team to schedule and track environment setup activities; install test environment software, network resources and hardware; refine test databases and develop test bed scripts and environment setup scripts.

When it comes to cross browser testing, setting up hundreds of browsers & browser versions over numerous devices could be very challenging and buying a device lab is not an affordable option for all. This is where cloud-based tools such as [LambdaTest](https://www.lambdatest.com/) come into play, offering more than 2000+ browsers and browser versions which are hosted by VM for numerous desktop and mobile devices.

#### Key Areas For Test Environment Setup

* **Test data** – Many times test environment setup are not populated with similar data with comparison to Production, this makes your product very brittle once the code changes are put into production environment
* **Front-End Running Environment** – Make sure your have a front-end running environment to perform load testing for analyzing the capability of handling web traffic.
* **Checklist of all the Systems, modules and applications** to be put under test.
* An isolated **Database server** for staging environment.
* Test across various **client operating systems**.
* Test across maximum **browsers and browser versions**.
* Make sure you test your website on **low and high network** to realize the difference between rendering time and overall appearance of the website or web-app.
* **Documentation is key** – Make sure you cover all the **Configuration guides/Installation guides/User manuals** and so on in a central repository.

Test environment setup involves the following tasks:

1. Tool licenses.
2. Setup utilities like advanced text editors and comparison tools.
3. Automation framework implementation
4. AUT access and valid credentials
5. Add-ins licenses

Various organizations utilize a staging environment to test the software. The best approach is to copy production data to test. It helps the test engineer to uncover the issues without corrupting the production data. Here are [13 reasons that leads to failure of staging environments](https://www.lambdatest.com/blog/13-reasons-why-staging-environment-is-failing-for-your-organization/).

#### Best Practices To Setup A Test Environment Management:

* Gather and understand the test environments thoroughly and train the testing team members.
* Check for the required software, licenses and hardware.
* Maintain a checklist of automation tools and their configurations.
* Maintain a cross browser testing matrix to ensure you cover the tests on numerous browsers and versions with respect to priority and market share.
* Make sure to test using real-time traffic to ensure your changes are more sustainable.
* Planning the scheduled use of the test environment.

### Automation Test Script Development + Execution

Once you install the test environment, it is the time to execute the test script. So, this phase of automation testing life cycle is dedicated to the execution of all test scripts.

To perform script execution, signed-off and unit-tested test scripts are delivered to automation testing team.

t is essential to ensure that all test scripts are running correctly. So, there is a need to take care of the following things before developing a test script:

1. Creation of test scripts based on actual requirements.
2. Create a common method of function that can be used throughout the testing process.
3. Make sure to create a reusable, structured and easy script so that a third person can understand it clearly. Here are [8 actionable insights for writing better automation code](https://www.lambdatest.com/blog/8-actionable-insights-to-write-better-automation-code/).
4. Perform [code reviewing of test scripts for better quality assurance](https://www.lambdatest.com/blog/how-code-reviewing-can-help-with-quality-assurance/).
5. Make use of better reporting.

Once the test script is developed successfully, it should be executed by keeping the following things in mind:

1. A test script should include all functional aspects according to the test case.
2. Ensure to run test scripts in multiple environments and across multiple platforms.
3. If possible, batch execution can be done to save time and efforts.
4. If the failure occurs due to some functionality, write a bug report.

To execute test scripts and procedures, the test team has to comply with a schedule decided for procedure execution. Evaluations for test outcomes are executed and test results documentation is prepared during this phase.

Test outcome evaluations are executed and documentation for test results is prepared.

Plans designed for the unit, system, user acceptance and integration testing are run to test the system as a whole. Code profiling is done at the time of unit testing. Profiling discovers instances when there is inappropriate scaling of algorithms, resource utilization and instantiations.

### Analysis + Generation Of Test Results & Test Reports

After all types of testing are performed, the testing team analyzes to identify particular functionality or components that experience a relative number of problem reports.

The result of the analysis suggests that if it requires additional test efforts and procedures or not.

Test results generated from the analysis can confirm whether executed test scripts/procedures can identify errors.

It is the last phase of automation testing life cycle and the test reports are shared with all involved stakeholders at this stage. This is why, test reports are crucial for analyzing how well your web-app responds to adversity. You can use an old school excel sheet, however, LambdaTest Automation provides an in-app report of all test cases executed through your automation script on cloud-based Selenium Grid. Intriguing right? Know everything about LambdaTest [Selenium Grid for automated cross browser testing](https://www.lambdatest.com/blog/lambdatest-now-live-with-an-online-selenium-grid-for-automated-cross-browser-testing/).

## **Wrapping Up**

Automating testing is an effective way to meet the testing goals within the appropriate timelines and with adequate resources. However, make sure you implement the complete automation testing life cycle to get the expected results and test the application in the best possible way. Automating tests without a plan or a sequence can lead to massive scripts which might often fail and involve manual intervention too.