

# Effective Automation Test Script Design

*Have you ever automated a manual test script and then two days later thought of a more efficient way of creating it? Or maybe completed 10 test scripts over a two-week period and then realized that your automation design only allowed you to cover 50% of the scenarios you have to script?*

*Do you then:*

- Only complete the remaining scripts with the more effective design, thus having coded two different approaches?*
- Complete the remaining scripts with the inefficient design used on the initial scripts, only spreading the bad design to other scripts?*
- Go back and rework your completed scripts, adding more time to your already tight deadline?*

*Those of us who are automation specialists commonly find ourselves under a time constraint to complete automated test cases. In our haste to create automated scripts, we often forget the bigger picture of automation test case design.*

For automation projects that evolve past the start-up and installation phases, poor test case design leads to more automation project failures than any other cause. In our rush to show progress and results to our customers, we tend to start scripting without putting enough thought into what we want to automate and developing a formal approach. As QA professionals, we all know what happens when a developer starts coding without requirements. How can we expect to deliver quality code without applying some type of methodology to our approach?

## Test Automation Using a Methodology

Eventually, most seasoned automation specialists realize that an automation project should be treated just like a development project. To be successful with test automation one must look at an automation project from a broader scope. It is necessary to use an automation methodology to design robust automation scripts around specific functional areas of an application. Using the basic scripts, you can generate scripts for regression testing and for testing new functionality of subsequent releases.

A recommended methodology would include the following phases:

- Analysis & strategy
- Design
- Construction
- Execution

## Analysis & Strategy

Analysis and strategy is probably the most important part of your methodology. During this phase, the automation specialist gathers requirements for the automation project. It is important to remember that these are not the requirements of the application. These are the items that are critical for the completion of the automation project. During the analysis and strategy phase:

- All project documentation should be reviewed, (including business requirements).
- Test plans should be analyzed so the automation specialist understands what is being tested. Combining this understanding with the tool capabilities will allow the automation specialist to determine what can be automated.
- Specific application controls should be analyzed with the automation tool for compatibility.
- If the application under test (AUT) is available, the navigational flow of the windows should be mapped. If the AUT is still under construction, then window snapshots and window design documents can be sufficient. The goal is to obtain a general feel for how the application flows from window to window and how data flows through the application.

Other automation requirements might suggest that the test suite run in unattended fashion, or run on only specific operating systems. Once all of the requirements are gathered, it is important to gain consensus from all of the stakeholders. These requirements will allow the project to meet the desired goals of automation.

The business scenario is another important component of analysis and strategy. Scenarios are the building blocks for automated scripts. The validation matrix below contains the functional areas of a CD purchasing application in the left column. Each area has been ranked according to its importance to be tested. The functional areas are marked under column headings TS1, TS2, etc. to reflect the flow of a test case. Combining a flow through the functional areas to imitate what a user would perform creates business scenarios.

The matrix also allows you to know what to automate based on risk. The scenarios ranked as “H” (High) should be automated first. Once you have created a script to handle one flow, you should make it generic enough to handle different data inputs so you can create any combination on this matrix. This is an example of designing scripts that are robust enough to handle all combinations. The purpose of a generic script is to avoid the duplication of code across the application. The generic script may contain functions to perform specific tasks including data-driven activity or error checking.

Our analysis of this example reveals that there are three functional areas related to the application: “Order Processing,” “Music Types” and “Credit Card.” These may be three different windows, or one main window with many input options. The objective is to make one generic automated script (or in some cases, several generic scripts, one for each functional area) that is robust enough to handle any of these combinations of inputs.

Here is the issue: if we made a generic script based on business scenario TS1, it would not include code to handle the functionality for “Order Not Approved,” (a functional subset of the “Credit Card” functional area). Later when we were ready to code TS4, we would have to then modify our generic script to allow us to handle that functionality. We don’t find out until TS5 that we are using the check for “CD Quantity >= 4”, (a functional subset of the “Order Processing” functional area), thus allowing a customer to receive a discount. If we had haphazardly chosen to automate our test cases, we may not have chosen the correct test case to give us the best example of a generic script. Using this table allows us to see all of the functional areas in the application and build one automated script that can handle different data inputs or logic control to cover any combination in the matrix. This cuts down on rework of automated scripts.

Once you have a generic script that covers your functional areas, this script (or series of scripts) can be used to quickly create new automated test cases for subsequent releases. If a new music type is added in the next release, such as Jazz, we just add it as a new data choice for our input files and we have a new test case. Let’s say our company decides to start accepting checks by mail as a form of payment. A new payment type, “Check,” is added to our application. This is inside a new window, so it becomes a new functional area. However, nothing changes with the “Order Processing” and “Music Type” code. We only need to code for the new piece of functionality and incorporate this into our generic script. We have 75% of the script done before we even start. This greatly increases the return on investment of our automated scripts. Without the analysis and strategy done at the beginning of the project though, we would not have had enough information to create such dynamic scripts.

Function Name	Rank	TS1	TS2	TS3	TS4	TS5
Order Processing						
Order >= \$100	H	X	X	X	X	
CD Quantity >= 4	M					X
Music Types						
R & R	L					X
R & B	H	X				
Pop	H		X		X	
Rap	M					
Credit Card						
Visa	H	X				
MasterCard	H		X			X
American Express	L					
Order not Approved	M				X	

## Design

Once we have completed the analysis and strategy, we must start to design our scripts. The design phase starts with determining a high-level design approach. You want to examine the different architectural approaches available. These include data-driven, keyword-driven, event-driven, and dialog-driven.

- **Data-driven:** The data used by the automated test suite is pulled out into a separate file, which the test scripts read.
- **Keyword-driven:** This is similar to data-driven, except keywords are used to indicate what steps the automated test suite should follow. Keywords are stored in a separate file.
- **Event-driven:** The automated test scripts respond to events such as windows appearing, user input or data changing.
- **Dialog-driven:** The test scripts present one or more dialogs at the beginning of the test run that allow the test executor to choose what functions the automated test suite should execute.

Each of these design approaches has its strengths and weaknesses. The goal of each approach is to allow the automation specialist to create a framework of code that will be reusable and easily maintainable. Any one of these approaches can be used to implement the scenarios developed during the analysis and strategy phase. The correct approach is the one that provides the best balance between the automation requirements met and acceptable level of risk. It is important to note that most automation specialists choose an architecture before scripting, but without much thought or comparison. Here we want to emphasize that a carefully thought-out architecture that matches the automation requirements will reap more benefits than just choosing an architecture with which the automation specialist happens to be efficient.

Once we have chosen an architecture, we need to create the low-level architectural designs. These will become the basis for our suite of scripts. Data flows are tracked and error-handling code is considered. All of the interactions between modules and routines can be defined to ensure that the interfaces are understood and will not need to be redone before the test suite works together. From this we can create a low-level design document consisting of identified data sources, logic flow diagrams and specific routines (functions) to perform special tasks. After these tasks are complete, all that is left is the actual construction of the code. Now the automator has a blueprint to follow when building the automated scripts.

Good analysis in the beginning of an automation project will allow you to determine what to automate and how to group functional areas within your application to take advantage of reusable code. Once that code is created, it can be reused for future releases. You will create fewer scripts that are easier to maintain. These scripts can be used for regression, as well as for creating scripts for a new release. Don't sacrifice analysis for speed when creating automated scripts. A well-designed script will save you time on rework and provide reuse over the life of a project.

Visit us at [www.spherion.com](http://www.spherion.com) to learn how your entire organization can benefit from applications that are delivered on schedule and within budget.