# How manual testers can break into automation without programming skills

**Jim Trentadue**

Enterprise Account Manager at Ranorex: jtrentadue@ranorex.com

## Abstract

Test automation is seen as the way to make testers more efficient and productive for doing their job. It's also known that you can expand your test coverage greatly by automating nearly all permutations of a given scenario. For large project teams and large releases, automating your testing is the only way project schedules can be met.

Test automation solutions come with a misperception though. A number of solutions require you to have a programming and development background or skillset. But this is not a skillset most manual testers possess nor want to. Often there is a divide between the manual testers and automated testers as these are now two different jobs.

So what kind of bridge can be built across this gap of the two different roles? How can the automated testers get more useful information from the manual testers? How can the manual testers be more productive with what they can contribute to the automation effort? It starts with understanding the automation structure and framework.

Whether they are writing the actual test automation test case or not, manual testers can be productive immediately by understanding a few key points: Where was the Test Automation industry before and where is it headed? What are the available frameworks in the industry? What are the key categories that a manual tester can be effective immediately with test automation?

This paper will focus on three main categories of work for test automation. These are Preparation, Execution, and Analysis. Within each of these three categories, there are three key points in each that cover the full category, which will be explored in depth.

## Biography

*Jim Trentadue has over 16 years of experience as a coordinator/manager in the software testing field. He has filled various roles in testing over his career, focusing on test execution, automation, management, environment management, standards deployment, and test tool implementation. In the area of offshore testing, Jim has worked with multiple large firms on developing and coordinating cohesive relationships. Jim has presented at numerous industry conferences including the SQE STAR East, STAR West and Better Software Conference East conferences, Software Test Professional's STP Conference, IBM Rational Software Development Conference, IIST's SQTM conference, QAI's Quest conference and QAI chapter meetings. Jim has been invited as a guest speaker at the University of South Florida's software testing class, mentoring students on the testing industry and trends for establishing future job searches and continued training.*

# 1. Introduction

This paper has been developed with the premise that there may be elements of test automation existing in your environment. Test automation largely has been considered to be a development activity without a clearly understanding of where a manual tester can contribute. This manual tester may have technical skills, but not a development or programming skillset. But as you'll see within this paper, there are many ways where a manual tester can add value to the test automation process. For example, there are challenges from the test automation engineer's perspective to get better requirements or test documents from the functional manual tester and there are challenges from the functional manual tester to have the automated test case meet the same criteria as a manual test case.

Some other key assumptions is that the majority of test analysts are still performing a manual testing role. But with the points below, they would participate in the automation effort without being fully dedicated to the test automation team.

Based on over 16 years of Testing/QA experience in the industry and also working for a test automation vendor, this paper is based off of my recommendations from industry experience. If a manual tester was starting an automation initiative, many of the same principles below would serve as requirements for selecting a solution.

# 2. Categories for test automation

Below you will find an outline for the three different categories for manual testers to get into test automation: Preparation, Execution and Analysis. Within each of these categories have specific sections where manual testers can provide immediate contributions.

1. Preparation
   1.1. Planning test automation activities into your testing process
   1.2. Creating data sheets or SQL statements for data driving your tests
   1.3. Editing existing manual test cases for automation
2. Execution
   2.1. Learning how objects are recognized within test automation
   2.2. Working with the object repository and organizing this
   2.3. Building test automation actions from the ground up
3. Analysis
   3.1. Adding error handling in your testing flow
   3.2. Knowing how to step into test cases and debug
   3.3. Reading report results and interpreting this

## *2.1 Preparation:*

### 2.1.1 Planning test automation activities into your test process

QA organizations have defined testing processes, without the incorporation of test automation. Methodologies stem from an Agile, Waterfall or other SDLC methodologies. Largely, this serves the need for the manual testers. In most cases, there is not an integration of the test automation activities into the manual process. To illustrate how these activities could be mapped, a simple V-Model is used, regardless of an Agile or Waterfall SDLC methodology.

#### 2.1.1.1 Project Initiation ⟶ Test Strategy

It's recommended that QA release leads own the inclusion of test automation activities within each of their releases. It's unrealistic to think that test automation activities can occur for longer periods of time without impact to project schedules. If you list out the test automation activities for creating or executing automation test cases within the Release / Master Test Plan or Test Strategy document, your Project Manager or Product Owner will have visibility on planned automation within the given release.

#### 2.1.1.2 Analysis ⟶ Test Scenarios

As manual test analysts are drafting their test scenarios or test objectives during the analysis phase of their project, they should divide the scenarios between manual and automated. Everyone will know their objectives and can plan for good coverage on each.

#### 2.1.1.3 Design ⟶ Test Cases

Typically, manual test cases are not written for certain error-handling conditions that automation requires. Considering this during test case development will prepare the manual test cases for automation.

#### 2.1.1.4 Develop ⟶ Test Scripts

Record your automation test cases with the same principles as you would write your manual test cases. Just ensure that you are going slow enough to recognize all objects you are interacting with.

#### 2.1.1.5 Testing ⟶ Test Results

Unlike manual test case execution, automated test execution may require replaying the test several times and making several tweaks and modifications to make test run as expected.

#### 2.1.1.6 Deploy ⟶ Test Summary

Similar to the Test Scenarios document, list out what was executed with automated test cases vs. manual test cases. This information is listed in the Test Summary document prior to deployment.

### 2.1.2 Creating data sheets or SQL statement for data-driving your automation

In many cases, the test automation associate is not the subject matter expert (SME) in the application under test (AUT). The functional test automation analyst will know the UI, as well as the underlying database tables and fields. It would be in the best interest from all team members on the testing team to utilize the knowledge the functional test analyst has for using dynamic data within the automated tests.

It's also important for the functional test analyst to know a typical process that a test automation analyst would follow. Knowing this helps provide a common understanding of how the test automation test case is formed and when the dynamic data element is needed. This is a standard three-step process:

#### 2.1.2.1 Create a variable on what you want to data-drive

There are values where the automation test case will either select or input data to the respective field. This data is static or hard-coded unless driven from a source. The test automation associate will create a variable instead of using the original value so static data can be substituted out for dynamic data ongoing.

#### 2.1.2.2 Link to your data source where the data resides

Test data should reside in a central, secure location and the test automation analysts can link data from various formats: Relational Databases (RDBMS), Excel spreadsheets or CSV files. The functional analyst should select what's easiest for them, but try to use the same data source for all of their data-driven tests.

This will reside on different data sheets or tables, but the connector the test automation associate uses will be the same.

#### 2.1.2.3 Map your data source columns to the defined variables

Once the variables have been defined for the fields and the data source, connecting the data source fields to the variable fields should be straight-forward and essential. This must be done to have the correct mapping on data elements.

### 2.1.3 Editing existing manual test cases for automation

Most QA organizations have hundreds, if not thousands of manual test cases that they would like to be automated. But the chances are that many of these have not been written in a format suitable for automation. The functional test analyst should not worry too much about the overall test case as there are just two areas that should be focused on for editing: Test Steps and Test Data, but not Test Results.

#### 2.1.3.1 Test Steps

The test steps are the items that map directly to either the recorded steps, or to the created steps from the test automation analyst usually through code. Every step must be accounted for; no implication that the automation will know what step to do next. Also, the wording used in a common manual test case is not needed for the automation test cases. If a keyword-driven automation framework is used, limiting the wording is preferred. For example, a manual test case may have a step phrased like this:

- *Click on the City field and input the following state: 'Florida'*

The automation test case could have the same functional step, phrased as the following:

- *Action – Event – Object or Repository Item*
- **Mouse – Click – CityEditBox**
- **KeySequence (Input) - Florida**

#### 2.1.3.2 Test Data

As mentioned in the section above, data-driving the test is a best practice. Just as hard-coding data within development code is to be avoided unless absolutely mandatory, hard-coding data in tests should follow the same practice. Of course there are times that having static data elements may be beneficial to the test, but overall, this would be better to have an external data source for maintenance. This way, the responsibility is shared between team members. The test automation analyst is responsible for maintaining the harness and the functional test analyst is responsible for maintaining the data.

#### 2.1.3.3 Test Results

The reason why this is not listed as a step required for edits and maintenance is that you can rely on the test automation solution to provide all of the test results automatically for you from the execution run. Not only will this be a more reliable source of information, but also will provide much more. It's a best practice to separate your results from your planning information. You will usually have red-flagged or a failed test result if a test step or validation does not match, green-flagged if everything proceeded as expected.

## *2.2 Execution:*

### 2.2.1 Learning how objects are recognized in test automation

This is a key area of understanding for a functional test analyst, how objects and controls are identified by the test automation associate or the test automation solution. There may be a new term learned called the '*Accessibility*' layer. What this is referring to are those attributes and properties that are made

accessible from the development team. Some examples of this are Object Name, Object Index, Inner Text, Caption, Label, etc. The test automation solution latches on to one of these properties to be able to accurately identify this test step with repeated success.

Again, this is such a key attribute to be able to know how the test automation analyst is working or how the test automation solution is working to replay your tests with success each and every time.

#### 2.2.2 Working with the object repository and organizing this

You may have an application that has a number of forms, windows or screens. Then on each or some of these forms, there may be 30+ controls or objects contained within here. When the automation solution is interacting with each of these, the object repository itself could get very large. The object repository is the collection of objects that the test automation solution has either clicked, pressed, inputted data or done any other event on that specific object.

With respect to the section above, there are attributes that the test automation solution has been in contact with and that information is stored in the repository. This information is centralized so you would not have to record or write another test to be able to work with that object. What you may see though is a series of controls that may or may not have good object names or other properties for automation. For example, if you have a series of radio buttons for each state in the United States, the name of the object might be RadioButton1 through RadioButton50.

What's recommended is that following the initial recording or coding for that object, make sure to edit the name of the object to make it more meaningful if it is not already. Most solutions have an object spy tool that can aid with this. You don't want to have to go back to the object repository to figure out object names for a test you did weeks or months ago.

#### 2.2.3 Building test automation actions from the ground up

When starting off with an automation effort just like anything else, planning is key. It would be good to have *'an idea'* of how you wanted this structured, but it should not be mandatory. For example, you may create the following structure for your automated test cases:

Login TC – Administration TC – Work Order TC – Checkout TC

What if there are steps you want to add to the Administration TC that impact the Work Order TC and ultimately the Checkout TC? Does that mean you have to re-record or re-code your effort? You shouldn't.

What is recommended is to concentrate on a quality recording, end-to-end. Make sure the steps flow clean with the proper objects clicked on. If done through code, the test automation analyst should build this in such a way that there is flexibility to add new test logic so the functional test analyst can modify this as needed without starting fresh. If you wouldn't recreate a brand new test suite or case for some inserts of logic for a manual test case, then you shouldn't have to with automated test case.

## *2.3 Analysis:*

#### 2.3.1 Adding error-handling in your testing flow

Test automation error-handling is different than manual testing in many regards. For example, some automation conditions that may have to be dealt with are:

- What if an unexpected window appears in the middle of the automated test execution run?
- What if objects are moved, added, removed or changed within my screen or application?
- What if objects are delayed in appearing in the automated test run?
- What if additional steps need to be added to your test automation recording?

These questions should be on the forefront of automation preparation and error-handling for the functional test analyst. Knowing how the automation will and should react to these situations will ensure these tests are robust for future test execution runs.

### 2.3.2 Knowing how to step into test cases and debug

Test automation test cases are similar to a manual test case in documenting the step-by-step procedures. Most of the test cases have code generated in the background. One key advantage of automation is the ability to step into a test case to see where an error has occurred or what application behavior is present. The following is a three-step process for enabling this detection:

#### 2.3.2.1 Set your breakpoint

Evaluate your test case and put a stopping point on the specific step you would like. What this does is run the test up until the point and does not proceed further until instructed to do so.

#### 2.3.2.2 Choose the 'Step' for the breakpoint

You want to select this after careful analysis because this is dependent on the state you want to see your application in. Knowing this step is very important in driving the subsequent step where the application displays an unexpected behavior from what you thought.

#### 2.3.2.3 Execute the test case to the breakpoint

Once the breakpoint is hit, you have a few different options for further analysis:

- *Step Over*: stepping over the breakpoint to execute the next set of commands
- *Step Into*: stepping into the current function to see if that runs as expected
- *Step Out*: stepping out of the function entirely

### 2.3.3 Reading report results and interpreting this

Test Automation reports can be very informative and produce much more information than a manual test case would have noted. The fact that this information is produced automatically makes it great to review. Such report information is now available for your review within the automated test case execution run:

- Execution Time
- Machine Name
- Operating System
- Screen Dimensions
- Language
- Duration
- Total Errors
- Total Warnings
- Global Parameter Values
- Ability to jump right to the failed step
- Total Number of Iterations

# 3. Case Study

**Titled:** *Automation through the Back Door (By Support Manual Testing)* [1]

**Background:** To improve the rate of test automation in the organization, modifications were made to the test automation framework to support manual testing.

**Technical Solution:** Develop a framework that is based off of keyword-driven testing called command-driven testing.

**What is Command-Driven Testing?**

- Uses keywords that are simple commands (SELECT, BUTTON)
- Interpreter scripts are the same for all products
- Using the advantages that come from Data-Driven testing, navigation was placed into a DRIVER-File; data in a DATA-File
- These two files built together would form a command script
- The script-runner reads sequentially the commands in the DRIVER-File. DATA-Codes are substituted with data from the DATA-File

**With Command-Driven testing:**

- Testers don't necessarily need to learn tool scripting
- The separation in navigation and data sections makes the command scripts flexible and reusable
- DRIVER-Files can be easily ported to different applications
- DRIVER-Files need not be changed on migrating to another tool
- The test tool is needed only to prepare the templates and to run the tests

**Prerequisites:**

- You cannot start if you don't know the application and the test cases that are to be automated
- You need a working engine that can interpret all the commands you are going to need for your test cases
- You must have registered all the GUI elements that will be used in test execution in the proprietary mapping of the deployed capture/replay tools in order to normalize the names of the GUI controls

**Process Steps:**

1. Record the test case with the capture functionality of the test automation solution
2. Translate generated script to command-scripts
3. Developing planned TC's from template and build test suites

**Case Study Key Points:**

- Due to limited testing resources for test automation, significant effort still had to be spent on regression testing. However, the command-driven framework was adopted for ***manual testing*** as well as automated testing
- This approach helped limit the number of times a tester would simultaneously work on the same template, which was a current weakness
- Defect reporting became much easier. It avoided the testing team having to repeat the same steps to recreate the defect
- Continuous reviews were done to assess what features were available and what was needed to support manual testing

---

[1] *Experiences of Test Automation – Graham & Fewster*

- Implementation included a feature that supports the execution of partially automated tests, that could aid with tedious test preparation tasks
- Manual tests focused on customer-specific conditions now instead

# 4. Conclusion

Test Automation has different requirements than those for a traditional manual testing role. Understanding object recognition and repositories, error-handling techniques, automation frameworks such as data-driven testing, as well as debugging and reporting are key elements in test automation that are not so prevalent with manual testing.

Knowing some of these aspects going into the automation effort, allows a manual tester to jump right in and aid in the record / playback. The industry trend has been driving more towards scriptless automation so manual testers can make the transition easier. Knowing the application under test (AUT), is the largest hurdle and most manual testers know this already.

All manual testers should get involved in some aspect of automation. Not only does it increase your knowledge of testing tools in the market, but also it increases your market worth as a tester!