**Automated Testing: Getting More From Less**

**Trenton Snoble**

**Department of Computer Science**

**University of Wisconsin Platteville**

**E-mail:** [**snoblet@uwplatt.edu**](mailto:snoblet@uwplatt.edu)

**Abstract**

Most business are trying to make money. An easy way to save on costs is to take a process that takes several hours, and cut it down to mere minutes. In a nutshell, that is what automation testing does. It takes hours and hours of real people manually going through test cases to verify results, and creates a thorough, repeatable process that can be run without human guidance. There are many tools out on the market today that provide a user friendly environment for creating automation scripts. Telerik’s *Test Studio*, for testing web apps, is one of the most popular and widely used tools for testing web applications.

Discussion will also include what benefits automation testing has to a business, the similarities between creating a manual script compared to an automated script, why creating a good initial script makes upkeep simple, and how regularly running automated test suites can prevent a million dollar mistake.

**Background – What is Automation Testing?**

Normally, to test something like a web application, a tester would have to go through and manually execute a list of test cases to ensure they produce the desired result. This may not look bad at first glance, but as time goes on companies move more and more towards digital applications. These applications grow in size and functionality every day, therefore requiring at least daily testing as updates and patches are released. This can consume hundreds upon thousands of man-hours, costing the company and its customers money. These applications must be tested though, as the cost of not doing so is even higher.

Now picture a company that has a huge website with lots of different functionality that is being constantly updated. The amount of time that goes into the website being thoroughly tested is huge. Some test scripts might be dynamic, meaning they change with each new update as sections of the website change. However others that deal with more simple actions, like logging in, hardly ever change. These kinds of scripts are repeatable, and make good candidates to be run by an automation tool. Doing this will be saving the company time, and therefore, money. That type of routine, repeatable testing, is called smoke testing. The name comes from the way plumbers test a network of pipes by pumping smoke into them and checking for leaks. It is a quick and easy way to catch a potential mistake.

An array of automated smoke tests that run with each deployment of an application can catch mistakes like pages not loading properly or text boxes not taking user input: things that completely destroy an app’s functionality. More in depth testing is called feature-level or regression testing, and is much harder to automate, though not impossible. [6]

**Terminology**

Along with any field of work comes a long list of acronyms, definitions, and all other sorts of technical jargon. These are some of the things that any reader will need to know if they wish to understand this write up. A script, in the context of this paper, is a series of instructions that can carry out a specific order [5]. That makes a manual script something carried out by a human, manually clicking through pages and typing in text. An automation script uses computing to do the exact same thing a human would be doing to execute a manual script. For the purpose of this paper, an automation engineer will be someone who creates an automation script either from scratch or based off a manual script. A tester will be someone who carries out a manual script. An automation tool is a software, usually written by a third party, which gives an automation engineer a GUI (graphical user interface) to create automated scripts.

**Value to the Business**

Businesses that have even a small portion of their testing automated see lots of bonuses. First and most obvious of all, automated tests do not have to be run by a human. This means that companies do not have to pay anyone to run the tests every day, and their employees can focus on other things, saving the company money. Freeing up personnel time also makes projects get done faster, as there is now more time to work on them. Also, because a human is not required to run the tests, more tests can be run for virtually no extra cost. This allows the company to test more aspects of their apps than they were before for about the same cost. More value for the same cost is definitely a plus for any business. Automated testing, when set up the correct way, can report what made the test fail, showing not only a screen shot of where the failure happened, but what exact step it failed on. This makes the debugging process very easy and much quicker for whom ever is monitoring results.

Shown below is a screenshot from Telerik’s *Test Studio* of the results file each script automatically generates if any step of a script fails. The calendar on the left of the screen would show the history of testing, so any patterns or constant failures can be seen easily. On the right is the step that the test failed on. More in-depth discussion of this popular testing software will occur later.

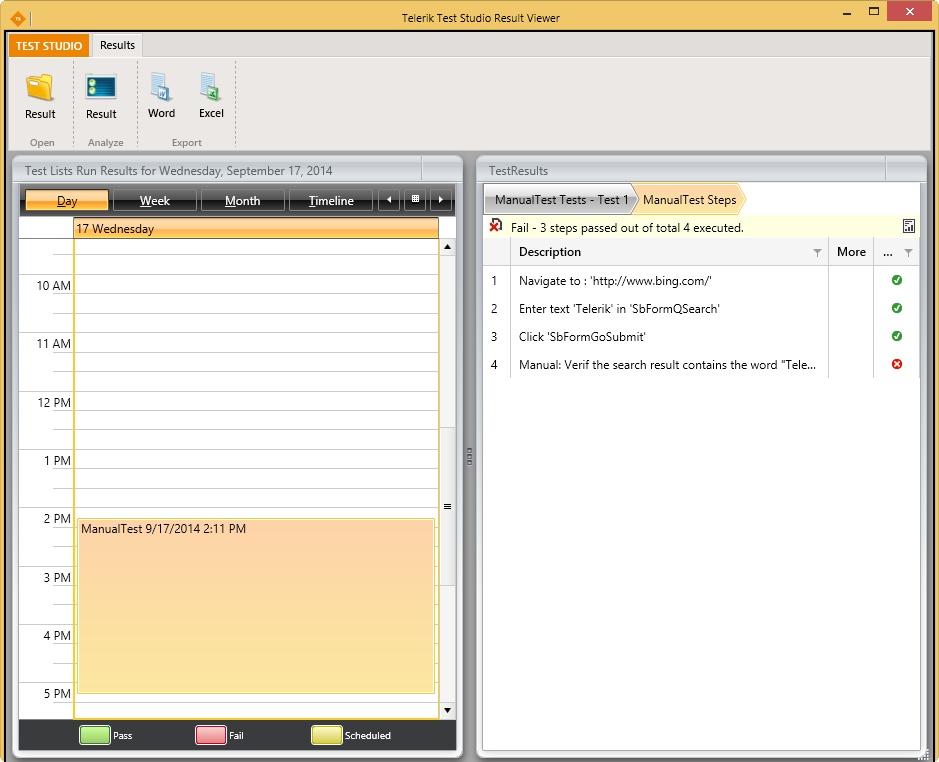


Figure 1: Screenshot of Test Studio's results file [1]

**Why Not Automate Everything?**

With all the benefits automation testing brings to the table, why not automate all tests and save tons of money? Not every test script makes a good candidate to be converted to an automated script. In order to be considered a good candidate for automation, a script should be unlikely to change drastically. Writing an automation script takes time, much more time than writing a manual script. This means if the test criteria is constantly changing, the time it takes to update the script. If the script changes enough, it is easier to just let it stay a manual test because changing the automated one every time does not provide enough return on investment.

To the untrained eye, it might look like every part of the testing should be automated. This is simply not a realistic expectation. Even the most testing focused companies are not trying to achieve 100 percent coverage of for their automated tests. It is neither cost efficient nor realistic to strive for that wide of coverage, at least in today’s world.

**Automating a Manual Test**

To become a successful automation engineer, a strong understanding of how to execute a manual test script is needed. A good automation script should be imitating exactly what a manual tester would do, just do it automatically to save time. Keeping this in mind, it is fair to say that the differences between creating a manual test and an automated test are actually quite small.

In fact, if a company is looking to start automating portions of its testing process, the best way to start would be taking a manual script and recreating the steps one by one. With an effective tool and someone that knows how to use it, this will not be a problem. Most automation software involves little to no knowledge of actual coding to use, though it does not hurt to have someone with at least a rudimentary understanding of coding.

Some manual scripts will be easier to automate than others. A simple test to make sure pages of a website are loading and all the input fields and buttons are working will be easy to create. Some tests may require additional validations. For example, an online store might want to test their checkout process to make sure totals are being calculated correctly. This will require the automation engineer to make variables to store the correct values, so a basic knowledge of coding will be needed.

**Creating Object-Oriented Automated Scripts**

These days, any programmer worth their salt will tell you the only way to write code is in an object-oriented fashion. Anything else is confusing, hard to follow, and quite frankly bad programming. Similarly, the only real way to write automation scripts is in an object-oriented manner.

When writing code, programmers make sub routines and call upon them as needed. This practice helps to avoid copying sections of code multiple times and makes the code more readable to anyone outside the project. Most automation software allows for the creation of subtests that can be called just like a subroutine. For example, if an automation engineer is writing scripts to test a company’s website, he might want to create a login subtest. Now every time he needs to test functionality of anything that requires logging into the website, all he has to do is call the login subtest.

This saves countless hours of redoing the same set of steps over and over again, not to mention making the script look nice and clean. Making good scripts is a key element to having a good testing process. Without good, object-oriented scripts, automation testing is hardly even worth the effort.

It is also worthwhile to note that not all testing software allows for the creation of subroutines. Some of the more outdated ones lack this feature. While these cheaper options might be suitable for a smaller company, any company that wants to automate a good portion of their testing should steer clear.

**Right Tool for the Job**

A good tradesman is nothing without his or her tools and the right tool can make all the difference in the world when used properly. Telerik’s *Test Studio* is a popular and effective tool for making and executing automated testing scripts. Using this tool to create basic smoke tests is very easy. That does not mean that a person without experience will be able to pick the tool up and start pumping out automated scripts the same day. Learning to use *Test Studio* effectively takes some training and hands on experience.

Here is how *Test Studio* works. It will link to an open web browser and record steps that a user is doing. The image on the next page shows what a script looks like in *Test Studio*.

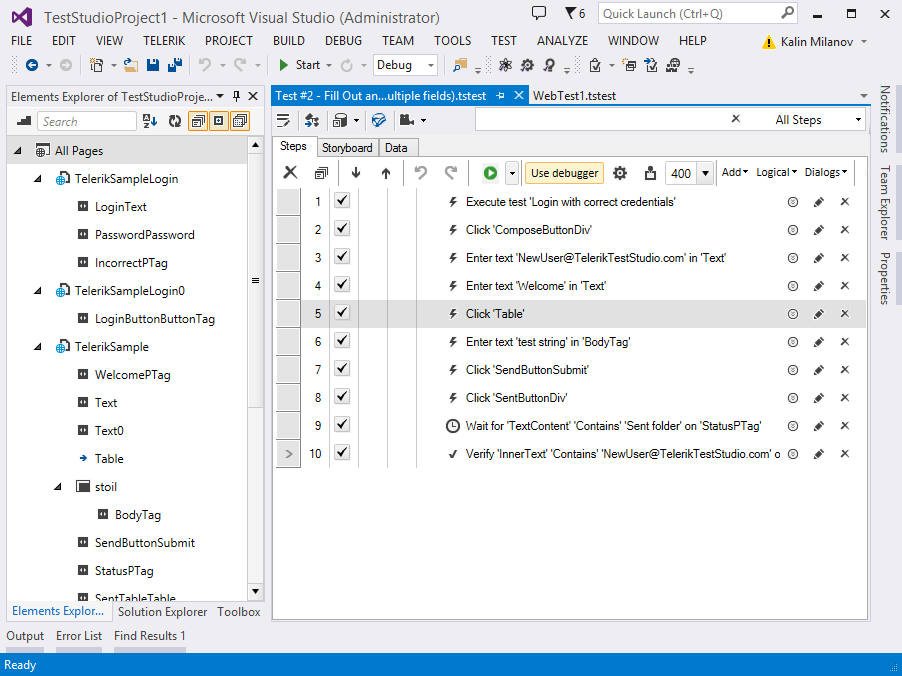


Figure 2: A script in Telerik's Test Studio [2]

There are several ways to make a script using *Test Studio*. A user can execute steps that are part of a manual script on a webpage and the actions will be recognized by the *Test Studio* recording device. Steps are being shown on the right portion of the screen. These steps can include things like entering text into a field, clicking a button, and waiting for a certain part of the page to load and become visible to the user. For more veteran users, on the far left side is a list of elements that the tool has found on an open webpage. Anything from text boxes to labels to buttons show up in this element library. A user can select any element and pick a type of action for *Test Studio* to do to that element. For example, the first element in the pictured element library is LoginText. That element is a login text box. The user would select LoginText and then pick an enter text option. The user can either pick the text to be entered or choose to data drive the text off an Excel sheet or database.

*Test Studio* acts like a sort of IDE (Integrated Development Environment) for making tests. If a user want to execute a script one step at a time, they can using step over and step into controls. Breakpoints can also be set by the user to make a script halt before executing a particular step.

There is also plenty of documentation for *Test Studio* exists along with lots of tutoring videos, so acquiring a basic knowledge, at least enough to set up some simple tests, is very simple and straight forward.

**Upkeep and Maintenance**

If an automation script fails, there are several candidates for the cause. The first failure cause is the easiest to deal with from a testing point of view. The application being tested is flawed and a developer should be notified. Even though the script failed, it did so because the application is not behaving the way it should be. The second reason can be harder to fix. As an applications life goes on, things about it will change. This means that scripts will have to be updated to keep up with these changes. For example, consider a web application prompts a user for log in credentials and then goes to a home page. The developers now feel that the first page a user sees after logging in should be a welcome screen that requires a user to click to continue to the home page. While a manual test might not care, this type of change will break the functionality of an automated script, causing failures.

A human tester could simply see the change and navigate through the new page, understanding that the app is still functioning as expected. An automated script, on the other hand, is not “smart” enough to understand this. In this specific example, the fix would be to simply add as many steps as necessary to get the old first page. This would allow the script to handle the new first page without having to change much. However, many things can change within an application to break an automated script, find logic being one of the main offenders.

Find logic is what the tool is using to locate an element on the page. Most common automation tools use the CSS or HTML tags to locate what element an action is trying to be performed on. If a script is set up to look for a specific tag within the code of a webpage and that tag changes, then the test will not be able to locate the element. The tool might also locate an element that is not what the tester originally meant. Below is an example screen shot of what Telerik’s *Test Studio*’s find logic can look like.

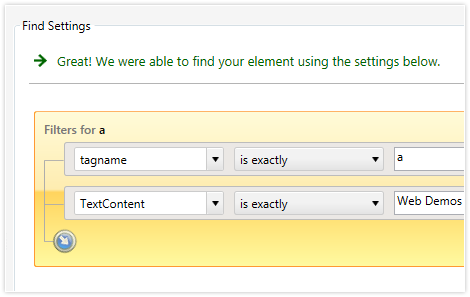


Figure 3: Find logic editor in Test Studio [3]

The element that this find logic is trying to locate has a tagname of exactly “a” and the text says “Web Demos.” This element is most likely the title or heading of a webpage. It is worth knowing that more than one element can be found with the same logic. If this happens, *Test Studio* will select the occurrence that satisfies the find logic. The automation engineer can change this by adding an index stipulation to force the tool to look for the second (or third or fourth etc.) occurrence.

**Benefits of Routine Automation Testing**

The greatest advantage automation testing has over manual testing is the ability to run tests without needing any personnel to spend time running them. If an application is being constantly patched and updated, it will need testing at least once daily, if not more. Having automated tests run every day is basically free for the company, once the automatic script is created. Not having to pay employees to test the application everyday frees up those funds for use in other areas of the business [7].

Another benefit of having automated scripts to run is that an application can be tested at any time. Before automation, there would have to be testers available to make sure each and every update to an application changed only the functionality it was supposed to change. With automation, that is not needed. All it takes is a one person and a few key strokes and there could be hundreds of tests kicking off in a matter of minutes.

Having scripts run daily to test an ever changing application can catch many small mistakes. An estimated 60 to 100 billion dollars is lost every year by companies due to poor testing [4]. These losses come from losing sales due to buggy and unreliable software as well as costs to repair damage done by not catching a bug early in an applications development lifecycle. So it’s more like a multibillion dollar mistake, not a million dollar one.

**Conclusion**

Automation testing has its drawbacks, like being harder to maintain. However, in the end, the numerous benefits outweigh the drawbacks. Benefits such as saving the company time and money, being able to run more thorough tests more often, and being the tools being fairly easy to use.

A company doing any type of application testing should have an automation team working committed to streamlining the testing process and eliminating the need to have a human tester run the same tests day in and day out.

**References**

**Images**

[1] <http://docs.telerik.com/teststudio/features/testing-types/manual-testing/fast-forward>

[2] <http://www.telerik.com/sfimages/default-source/blogs/432-png>

[3] <http://d585tldpucybw.cloudfront.net/sfimages/default-source/productsimages/teststudio/findexpression-ui.png?sfvrsn=2>

**Content**

[4] Hayes, L. G. (2004). *The automated testing handbook*. Richardson, TX: Software Testing Institute.

[5] Peppler, K. A., Tekinbaş, K. S., Gresalfi, M., & Santo, R. (2014). *Short Circuits: Crafting e-Puppets with DIY electronics*.

[6] Nguyen, H. Q. (2000). *Testing applications on the Web: Test planning for Internet-based systems*. New York: Wiley.

[7] Humble, Jez, Farleey, David (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Crawfordsville, Indiana: Addison-Wesley.