



Top 10 Test Automation Script Mistakes

A WHITE PAPER PROVIDED TO ASPE BY JERRY E. DURANT, CERTELLUS LLC

Top 10 Test Automation Script Mistakes

By Jerry E. Durant, Managing Director, Certellus LLC

Unlike the Top 10 Hit Parade in the music world, I'm not sure that anyone has either surveyed or has been willing to offer up their own personal list of dirty automation script problems. The excuses that I have heard range from "legal won't allow public disclosure", to very legitimate feelings that companies are alone in making these types of errors. After all, the vendor has assured them that the tool is so easy to use and implement that "even a caveman could do it".

With even slight shifts in operating paradigms one is apt to make naïve and unintentional mistakes. This is a natural part of the learning process, and while unavoidable, the impact can be minimized to a large extent. How? Acknowledge the risk, establish a remediation plan, and deploy measures to contain the potential damage. Sound too easy, maybe so, but often it's unrealistic expectations that challenge us to overlook pragmatic solutions and lay open exposures unnecessarily.

The list of the Top 10 Test Automation Script Mistakes is not a survey, but is based solely on engagements, inquiries and stories shared with me over the last several years. They are not reflected in order of importance or frequency; just being on the list is enough. What is provided is a description of the mistake, how it got propagated, and how one might minimize the impact should it occur in your scripting involvement.

Script Mistake #1: Deep Dive - Shallow End

The transition from manual tests to automated test scripts involves a natural series of steps. Even though we have a bunch of manual scripts we need to examine whether they are suitable for automation, are they in a condition that allows for implementation, and whether their value contribution is sufficient. Companies either implement everything or throw everything away and start over (the Deep Dive). The result is a complete automated script implementation, which may be practical for limited situations (re-engineered solutions) but often creates unnecessary chaos in most testing situations.

Remedy: Evaluate existing testware and determine what should be automated, how existing scripts should be cleaned up before implementation, and establishing a value to all scripts. Consider evaluating the scripts in groups rather than individually if you have a large manual script repository.

Script Mistake #2: Automating Without Sketching

I once watched a watercolor artist pencil sketch a scene and wondered why he sketched before painting. I discovered later that it provided both perspective and the ability to concentrate on specific details of the painting. Testing an application is a lot like a painting,

sure you can just start testing but what we discover is that the result is fragmented, overlapping, and sometimes confusing test scripts. Taking the time to understand the application design, the specific nuances, and having an overall test design in mind will help to expedite and produce cost effective scripts. While some tool solutions have test design modeling tools, tests need to employ techniques that fit their particular style. Some find that the use of Unified Modeling Language (UML), Cause-and-Effect Graphing, Diagramming, Narrations and even Picture Sketches help to unlock the vision of what they are wishing to accomplish.

Remedy: Develop (or evaluate) your high level and specific test designs. Consider the use of various techniques, which may uncover new testing means. Supplement with unsketched tests, but never use this as the principal means for test design.

Script Mistake #3: Too Big/Too Complex

With Return-on-Investment (ROI) comes the drive to implement. We not only want to show that our expectations have been met, but also that we are making a professional commitment to product implementation. There is nothing wrong with these desires, right? Well not exactly. Many times lofty expectations and unfounded estimations force us to do things that may be unwise. In this particular case it is a mistake of Too Big/Too Much. Complexity rears its head when we start to discover all of the things that the tool can do. Some of these things are valuable and some are on the very edge of what is necessary to get the test job done.

Remedy: Use a phased in approach and grow the test script repository at a reasonable pace. Concentrate on core test requirements and address those areas where the majority of our effort is placed. Use the rule of thumb that if a specialist is required than it might be a candidate for test automation; however insure that the script does not become overly complex. A wise man once said, “you can have a complex problem but it doesn’t always require a complex solution.”

Script Mistake #4: Extended Maintenance vs. Re-design

When is it time to throw away that article of clothing that has been patched, altered, re-altered and re-patched? We all know how comfortable it is, but there comes a time when we must look in the mirror and face the reality that it is starting to look pretty shoddy. Not to mention the seamstress is starting to refuse to do any further repairs, regardless of price. Test scripts, an integral part of testware, evolve from introduction into maintenance that closely parallels the evolvment of the application. Depending on how flexible we have designed (Script Mistake #2) our tests will play a large part as to how durable they will be. All good things don’t last forever, and test scripts are no exception.

Remedy: Repeatedly assess your test scripts. This should be done on a scheduled periodic basis as well as during each reuse cycle. It is strongly advised that criteria for test script goodness (also useful for developing tests) be established in order to provide advance unbiased assessment potential.

Script Mistake #5: Should Have Never Been Scripted

Tests provide more than results, they also provide insight into a broader range of application behavior. Have you noticed changes in performance, access, or capabilities as a bi-product of running a test? Would these attributes be lost if automated scripts were used in these areas? Virtually every test provides some form of secondary insight capability. However, the key here is overall value contribution. It might be worth automating but not at the expense of secondary attributes that may provide valuable advanced insight. Pick your choices wisely.

Remedy: Based on experience, you know what these secondary touch point indicators are and what tests they are a part of. Consider whether these tests need to remain manual or whether a separate touch-point test may be appropriate (to retain value).

Script Mistake #6: Failing to Recognize Effectiveness in Accomplishing Goals

What goals? Yes, as hard as it might be, most test scripts seldom do more than establish an expected result and do not connect with an overarching objective. Numerous studies have shown that the one document specified in the IEEE829 Test Documentation Standard, the Test Design Specification, is seldom developed. Despite numerous companies who have marched forward with the development of a test process framework, time pressures continue to challenge appropriate consideration of test fundamentals. The Test Design Specification was introduced as a way to segment large test suites into families of test scripts for purposes of efficient regression testing and expedient summarization. However, the challenge that is introduced is to form neat, tidy groups. We know that test scripts often have multiple objectives and as a result they could be in one or more places. Here is the good news; you can have a script that is developed once but referenced and used many times.

Remedy: There are many things you can do here. If you haven't developed an overarching test design framework, do one. Second of all, don't worry about trying to fill the groups. Go about constructing your test scripts in a sound fashion and measure what they accomplish when executed. At that point you are ready to implement your work and provide cross-referencing. You will be amazed at what you see in terms of efficiency opportunities and gap filling.

Script Mistake #7: Scripted with Meaning/Purpose

Even scripts have a need to have meaning (and you thought as testers you were the only ones with this lament). Some of this has to do with not only how we approach testing but also separating having a desire for early insight, as in the case of smoke-tests, and taking the easy route to getting started. My advice, don't change totally. Rather go into a short test stretching exercise, just to get limbered up, and then put some formality around it. I can assure you that you will look at the testing that you are doing in a much different light, and are apt to be much more focused on what you test and how much. It shouldn't be a big surprise that when one is given a tool that has the capability of testing a large range of conditions that we always want to take advantage of this. The task that isn't considered is that test results have to be scrutinized. Even if all of the tests pass, did the tests actually perform correctly (yes a pass isn't conclusively a pass until we assess the test's ability to accomplish a goal)? Tests that fail require in-depth root cause analysis and test correction. And for those tests that produce erratic results, more investigation and analysis will be required. In short, be prudent, cautious and conservative.

Remedy: Control mindless wandering, unless that is what your objective is. Yes, there are cases where this is appropriate such as exploratory testing. Encourage not only prudent test case design but also give consideration to appropriate test methods (e.g. pair-wise, boundary, flow...) that help to form a base of testing that is reasonable without being excessive.

Script Mistake #8: Untested Scripts (I Never Make Mistakes... Whoops!)

Are your scripts, and possibly your script designs, subjected to critical and formal review? Have they been tested using a controlled set of conditions to insure proper functionality? If this sounds a lot like what we expect from programmers and designs, it should. We are not immune to testware defects, shortcomings and outright testware failures. A client told me that they never make test mistakes. I guess it was true since they never measured these and simply fixed the problem and moved on. Again, very similar to defects found in unit test... "I'm still working on it so it's not really a defect" (food for thought... consider the difference between debugging and unit testing and maybe this will help you understand why this attitude persists). We need to share this information and encourage that reviews & testing occur before sharing and implementing.

Remedy: Subject all test scripts and test script designs, including high-level test design to critical and formal review. Since this process will disclose exposures a sample set of tests used to test-the-tests will find additional areas for attention.

Script Mistake #9: Undocumented

While some would argue that scripts are self-documenting we can't overlook the confirmation effect that we get when we take a moment to document. The separate, and important, task of documenting forces us to examine and describe what we have done. Sometimes we are forced to answer questions that we don't have answers for. This is the value that documentation provides, along with helping others to take over and utilize our scripts. Besides, we expect our developers to document their work product, why not follow good practice?

Remedy: Document within the test script and with supporting documents that would help someone to use and maintain the scripts. Balance the depth of detail by the skill level of the person expected to come in contact with the script(s). An additional consideration is the audience. There may be project and team managers that require a higher-level set of documentation than those who would come in contact with the test script.

Script Mistake #10: Not Used/Misused

How often are your test scripts used? Is this measured by script or by the general population of scripts? How many of you don't know? I am willing to stick my neck out on this one and suggest that it's highly unlikely that we could even answer this question, as it relates to manual tests. Test scripts are an asset and assets need to be utilized appropriately. It is just as bad to use them for everything, even when there is a lack of relevance, than it is to not use them at all. Why? The reason is simply time and value contribution. You might be lulled into believing that more is better, but I assure you that management is looking at the delivery of service with a mindful eye as to how we can do the job in a practical fashion.

Remedy: Measure the use of your test scripts. Have they been used, how often have they been used, what is the cost of deployment per test script, and as we will discuss later (Script Mistake #12) evaluate the results that are provided. Even if you are testing in a manual fashion, track these metrics. Automated scripts are capable of generating most of this information from log records, and as for manual scripts the manual test log is a suitable vehicle to use.

Ten isn't enough... how about a few more? We are not going for abundance, but there are simply a few other mistakes that one cannot overlook.

Script Mistake #11: Scripts by the Unskilled

A client of mine was quoted as saying "we never trust our children with our most prized possessions". Yet it seems to be common practice that we allow the unskilled to try their hand, not in an education fashion, but to cold implement the tool. While I have seen this in

both manual and now automated test construction, I can assure you that cautious mentored oversight is absolutely essential. On one occasion a client lamented that they had acquired a tool, set their staff about to implement test, and they found no problems. It wasn't until the application was deployed did they realized that they had some major problems. How did this occur? Largely it was because the testers had assumed that the tool would prove goodness without realizing that a benchmark of goodness required the careful attention of the professional tester. I firmly believe that learning and working as a group on initial tool script implementations are excellent vehicles for making the unskilled skilled when combined with test script review/testing.

Remedy: Manage the deployment, utilization, exposure and implementation of tool technologies. Subject all scripts to critical formal review and testing, especially during initial implementation periods.

Script Mistake #12: Judging the Value on Numbers not Results

Are your automated test scripts approaching a usage level where the time to run is close to what is available? A client that had a monthly release cycle that could not be changed because of regulatory requirements. This client had 10,000 test cases that took thirty-seven days to execute (and all were viewed as mandatory tests). I think you can see the problem. We were asked for help, and through a process of close examination we discovered that the same extent of test coverage could be accomplished with 70 tests and only required a four (4) day execution window. Very little redesign was required. How did this happen? Again, I blame time. Rather than re-design/adapt existing test scripts to fit application changes, it was easier to just create another script. Over less than 3 years the script population grew to this critical level.

Remedy: More is not better unless you are measuring coverage. Even then be careful. The more you cover with a single script the more difficult it may become to isolate failure causes. Mindful attention must be given to coverage, time consumption, and also test case effectiveness. A prominent company, whose identity shall remain confidential, called lamenting that the test approaches that we had helped them implement were not working. Closer examination revealed that the one step that wasn't exercised was continual test script effectiveness assessment. That examination revealed that those early scripts were effective in not only uncovering problems, but also forcing process change, thus eliminating future creation. Unfortunately, bugs have a way of cropping up in other areas and this is what they were experiencing.

And because I'm not superstitious...

Script Mistake #13: Uncoupled is not a Disconnect

A lot of my present practices are based upon making some pretty fundamental mistakes. As a developer I made the critical mistake of hard coding constants into my application, thinking about process and resource efficiency. I soon learned that every time conditions changed, my programs were subject to the change as well. This lesson is equally applicable to automated test scripts. Uncoupling test data from a test script allows for application changes to take place without impacting the script. Test Data is far easier to modify and update than changing a tightly bound script.

Remedy: When possible uncouple test data from test scripts. This step reduces overall script maintenance and also affords the use of test data by a number of test scripts.

Conclusion

Learning from mistakes, whether ours or someone else's, is an important part of the maturing process. Keeping these lessons to ourselves is easy, but consider the loss in advancing technologies and our profession if we do. I'm sure that there are other script errors that have not been mentioned and it's only a matter of time before they too will surface and become visible.