The REAL Costs/Benefits of Test Automation

Are you frustrated with vendors of test automation tools that do not tell you the whole story about what it takes to automate testing? Are you tired of trying to implement test automation without breaking the bank and without overloading yourself with work? I experienced first hand why people find test automation difficult, and I developed useful ways to cut testing costs. We must focus on simple tools that produce results. Testing is like systems development. If you want quality results, start with quality requirements. You should not start with test automation; you start with an organized approach to QA testing that will facilitate test automation. This paper explains how you can succeed when you address the REAL Costs/Benefits of Test Automation.

Testing, or test automation, is not rocket science. Some people make it more complicated than necessary. All you need to succeed is a good testing process that embodies a vision of what works and what does not work. As you select applications ripe for test automation you will find some that are better tested with manual approaches. Your overall approach must be consistent. You must avoid the cost of duplication. Most people are not even sure what they mean by test automation. What is test automation? Is test automation simply capture/replay processing? Is test automation programming in some script-like language?

Capture/replay works to fill time slots on a glut of TV channels, since all you change are the commercials to be aired. For software testing, think of the corollary of an old saying: “The more you want things to stay the same, the more things have to be changed”. If you want to replay tests, isn’t that because you want to review the effects of change in the application? You will soon discover that updating playback scripts is a maintenance nightmare, since each case is a unique recording.

Contrary to what some experts claim, test automation should not be program development. Linda Hayes called that “writing programs to test programs” and she rightly classified that idea as absurd. Some test automation product vendors want us to believe in a programming paradigm. Many experts lament that testers are too busy with manual testing to write test automation scripts. Some testers may not be qualified to write such test scripts. Is anyone ready to propose we ask development to double their workload? I didn’t think so.

To develop any software application, you have to start with fundamentals. Script writing is labor-intensive. It is difficult to maintain scripts. Generally, our initial testing is best done manually, so that we can stabilize application interfaces before we attempt test automation. If we use different OS/Browser combinations, we face the challenge of automating so many interfaces that it will seem a lot less work to just test combinations manually. Of course, the fact that so many tools are only concerned with GUI testing should be a concern to us. Don’t we test batch systems? Don’t we need to test individual layers in an N-tier server structure?

People may demonstrate “relative payback alternatives” of manual testing vs. automated test execution in terms of how many test cycles for automation to pay off. What does that mean? Why not focus on the challenge of employing automated tools for any kind of testing? Why not engineer solutions that eliminate a supposed inherent duplication of first establishing manual test scripts and then repeating the effort to produce automated test scripts? Well, IBM has published study results that claim 75% of all testing is still done manually. That means no solution is complete unless it addresses this larger part of the testing needs. I will explain how I created a solution that dramatically reduces the effort of test script creation, and especially of test script maintenance, that provides scripts for manual testing as well as for test automation.

Think about those analyses of breakeven points of manual vs. automated testing that cite a number of automated test sessions after which automated testing becomes cheaper. They seldom account for the need to maintain both manual scripts (as confirmed by IBM) and automation scripts. Many break-even analyses do not tell you how to account for the costs of manual script production plus for the *incremental costs* of added automation. What I will explain is how we can bring the cost curve down for manual testing by streamlining the scripting, and how some manual testing can be replaced by test automation (but not in terms of replacing all manual testing as some breakeven points imply). The biggest obstacle in the way of progress is to set unrealistic expectations of what people may gain from a specific decision. Overselling the benefits of test automation does nobody any good.

I focus on creating test scripts and eliminating unnecessary duplication. Test automation should be viewed like any other business process automation. I want a solution that meets the needs of my business. If that solution must simplify manual script writing, then that is the focus of automation. I use a practical solution for creating and maintaining test scripts by separating the fixed and variable aspects of scripts. A small set of scripts may be combined with a table of alternate data values so you do not have to replicate scripts for individual combinations. You may add a second table to list test database access keys for individual test cases. The reason is that, over time, a test database tends to change. The indirect approach lets you access the right test database records, while you avoid test script changes due to test database changes.

You can use my solution manually, but then testers must consult 3 sources of data in order to execute those scripts + data values + profile definitions. I created an automated solution that merges these 3 sources dynamically just prior to a test execution, so that testers see a "virtual" instance that contains the current data values. It is possible to cut script creation costs by 50% using this strategy, and maintenance by at least 75%. Database changes just prior to the start of testing are no problem: you can update the profiles and generate a new set of scripts that same afternoon, rather than to search through a stack of test scripts to find all the instances that will have to be changed.

I implemented the process used to support this "manual test automation" solution as a VBA macro within Excel™. I added data generation capabilities with that logic, to identify data by attributes, to identify pairs/triplets/quads for optimized combinations, to set mapping tables for equivalent values, and to define Governing Business Rules that determine what are testable condition combinations. I believe in fundamental testing, to relate test cases and scripts to business requirements and functional specification and to incorporate risk-based testing priority setting. These capabilities support complex script requirements with almost no extra effort on the part of the QA analyst.

By contrast, compute-bound data generation efforts are generally avoided in most manual script preparation initiatives. Shortcuts can seriously compromise the credibility of what is actually tested. With my tools I can account for what I test and why, which is an important automation benefit. I can provide reassurance that all that testing is based on fundamental requirements for the application, not based on what developers thought they had to build, which can be a major source of functionality bugs. Because manual scripting tends to be done in a hurry, QA analysts may look for the easiest source of requirements, so that tests are influenced by what the developers think they should be implementing.

Functional testing is not the only game in town. We need unit testing and integration testing that may not be done properly due to the effort involved in producing needed test cases. With a test framework (such as modeled after JUnit), I can use my tools to produce test data using the same test cases that the application must be able to handle in black-box testing. I output data into "*.CSV" files that can be input into a test framework to present alternative tests and to validate the results as well, so developers can stop bugs from infecting the code before it even reaches QA.

I demonstrated test execution automation with Certify™ (by Worksoft) because the internal working tables from my tools are directly compatible with recordsets in Certify™. A large part of the test automation engineering was done with cut-and-paste operations. I wrote a simple, reusable driver script for Certify™ that mimics my table-driven script generation process. That driver script invokes simple task oriented action scripts, such as specific screen dialogs, and Certify™ inserts the relevant data for each instance that must be tested. The Certify™ keyword-driven architecture allows me to produce automated script segments in hours. My data-driven architecture produces full execution automation within days of debugging the manual scripts. I use one common input, my source specifications, that I can update in order to regenerate scripts and recordsets in minutes. This all but eliminates script maintenance concerns and removes the logistics of managing manual test scripts in parallel with automated test scripts. I know that not everybody uses Certify™. Some people already bought into more elaborate products (perhaps based on writing test dialogs the hard way in VB Script). Initial feedback was that my approach was fine for a select user community, but not for a majority of users of test automation software. Some skepticism is healthy, too much is paralyzing! I like to think of solutions that are easily understood by most testing analysts. I want quick payback from test automation.

There is no benefit from products that end up as “shelfware” because they consume more resources than what they return in benefits. Certify™ is good, but really no single product is sufficient. Look at the complex IT solutions with different products integrated into a complex production environment. Ask if it is realistic to want a one-tool-solves-all solution. Look beyond individual tools and consider multiple tools. You need more software and additional learning, which adds complexity to integrate those tools to meet specific testing challenges.

To support other test automation products I created a second tool. I use the same data and I created a generic keyword structure to generate executable scripts for any procedural testing tool. This requires custom OPDEF’s (operation definitions) that use keywords to select script code segments. My tool can insert appropriate data values to make each script instance functionally operational. It takes a little more effort than if you use Certify™. First you need to write the same simple task oriented action scripts as explained for Certify™. You also need to write reusable OPDEF’s that convert keyword based actions into script code that can then be executed by the targeted test automation product. This way you can continue to use your existing test automation software if you are able to feed it directly from the same inputs we use to generate manual scripts. When changes are presented, you simply pull out the input workbook, you make the changes, and you regenerate your test collateral right down to your favorite test automation scripts, usually within the same day. If you need to work with multiple test automation tools, you can provide multiple OPDEF libraries and compile the appropriate scripts with the right set of definitions for each tool.

In summary, my solution requires only one level of maintenance effort that we minimize by using a 3-pronged approach of scripts + data + profiles. Any extra effort to produce the initial input data to ensure that your testing is sufficiently thorough adds extra cost to the script preparation effort. With my approach, the cost of initial scripting will often be reduced, and maintaining that script base is clearly much more efficient than if done manually. Consider:

- Scripts change because physical interfaces change. We can design scripts so that we use the fewest lines of unique code per GUI screen segment (or transaction file, etc.). Such changes are simple to implement with a tool like Certify™ that is able to “re-learn” a screen. We can eliminate duplication of low-level dialogs to dramatically lower the testing costs. The risk of script changes is so common that projects wait as long as possible before they start scripting. You cannot escape that risk in regression testing. Unless you avoid duplication of low level scripts, maintenance efforts will be significant. With my reusable OPDEF architecture you cannot minimize scripting any further unless you use a keyword-driven architecture that we use in Certify™.

- Data change when new functions support new (or additional) conditions or options. If data are entrenched in scripts it is difficult to find what you must change or add in the scripts to reflect new conditions. By keeping the data separate, it is easy to focus on data instances that are affected by application changes. We can update those data and regenerate scripts relatively quickly (typically in an hour or two after we receive the changes). This includes all the testing collateral necessary for the execution of test cases that reflect the new version of the application, which is a significant advantage.
- Profiles change because databases are not static. We make test scripts adaptable by using "profile" references in the data and by providing separate "definition" data to map each profile to actual database keys. There are many aspects to this concept. As data become stale, they are edited or replaced, and you reflect that in the profile definitions. Since you can describe data attributes for a profile you keep a clear focus of what you are testing, while account references become confused when the underlying data are subject to change. The extra step of using a profile is a critical step that safeguards your investment in the test collateral by formalizing the design documentation.

These concepts are lacking in most test automation tools. This explains why test projects fail due to multiple duplication opportunities. You need manual testing and test automation, and perhaps different test automation tools to test different aspects in different environments, and transactions files, and updates to database files. Unlike other approaches, I recognized the key issue as creation and maintenance of test scripts, transactions, and test data.

Maintenance of test collateral is onerous. It can consume over 75% of the total costs of small upgrade projects. You think not? Ah, what probably happens is that your testers use only 10% or so of the scripts for regression testing, and they only update that 10% of scripts. As a result that script base becomes corrupt after a few upgrade cycles. I recommend that you keep 100% of all scripts updated, even if you use 10% of those scripts in a given regression testing session. Test automation lets me regression test 100% of all scripts each time. A full regression provides a better level of quality assurance than what you can achieve with a purely manual approach, and at a fraction of the cost.

Keep your efforts to code automation scripts to a dull roar. Avoid "writing programs to test programs" as Linda Hayes cautioned. This is especially true if you need multiple tools to deal with different types of applications under test, because that would take an army of programmers with different skills to keep up with the scripting needs. Keep in mind the limitations of test automation when you deal with web-based applications that must be validated for compatibility with a myriad of customer configurations. You still need manual scripts. My solution uses Excel workbooks, IntelliData.xls and IntelliScript.xls, that automate the procedures described above. I have documentation and data-driven / keyword-driven training materials that I use in my business. Like many "simple" solutions, this did not evolve overnight: it is the result of many trial and error versions of code to solve specific problems. What I see in many testing tools is a constant effort to update the tools due to technological changes. What I see in most testing organizations is a desire to keep the application-specific test scripts as static as possible. Based on that premise my tools bridge the gap between testing tools and business needs and they satisfy the primary goals and objectives for test automation: to cut costs in manual testing as well as in test execution automation. I will be happy to provide you with additional information about my tools and my testing methodology.

Frits Bos (Frits@pm4hire.com) is a veteran of over 30 years in IT, and a witness to the tendency to reinvent the wheel. He provides contract services in Project Management, Business Analysis, QA, and BCP. He also develops training seminars and creates development and testing tools, and is an active contributor at the www.stickyminds.com site for QA testing professionals.