



Automation By Design: Organizational Bloopers

by Jamie Mitchell

This column was originally published in the Journal of Software Testing Professionals, Volume 1, Issue 2.

Last issue, we discussed reasons why test automation projects often fail. So how do we ensure success? The bad news is that there is no magic formulae for success; it is conceivable that your organization may do everything right and still fail. Software development (and test automation is most definitely software development) just works that way. The good news is that you can enhance your chances of success by attention to detail, good processes, and hard work.

Let's look at a common scenario for a nameless company and then try to come up with methodology and process improvements to increase the chances of success.

In our mythical organization, testing always seems to take far too long. The marketing people promised the delivery of this release six months ago, so every day the product languishes in test is a day that our competitors are stealing our market. This release cycle, the development manager has decided to change the dynamics of the process and purchase an automation tool to speed up the testing. The major tool vendors are each brought in and each is given time to put on a glitzy demo of their product's capabilities. One tool is selected and is handed off to an experienced tester. Several of the testers are sent to a three-day class on how to use the tool; all of the testers start to record scripts. For the first time it starts to look like some regression testing may actually get done in this cycle. Since the tool has some really cool features that make it easy to test entire dialogs and menu systems, it becomes a high priority to make sure all dialogs and menus are tested. During this first cycle, the tester / automators are often pulled from their automation development to do manual testing when a new build comes out.

Based on my experience, this organization is already deep in trouble. The worst thing is that they do not know it. In this all too common scenario, eventual failure is predictable. Every step of the way, mistakes were made. Let's analyze what has gone wrong.

The need for automation: Why should we automate testing? There are likely very good reasons to automate certain portions of our testing; however, the simple fact that it appears to take too long is not, in and of itself, a valid reason. Automation is almost never a quick fix for any test process shortfall, especially time to market. If the marketing people did a poor job of anticipating the needs of the marketplace, it is certainly not the fault of the testers at the back end of the process.



The requirements for developing test automation need to be thoroughly evaluated on their own merits. Do not assume that the tool will reduce the time it takes for the product to go through test; even in the best circumstances, it may not. Indeed, for the first cycle, unless extra testers are brought in to do the required manual testing, the cycle will probably be even longer – and less actual testing will get done. Every minute spent automating is a time not spent testing.

For an excellent treatment of automation requirements, I highly recommend a paper written by Cem Kaner called, “**Avoiding Shelfware: A Managers’ View of Automated GUI Testing.**” This paper is chock full of good advice for organizations that should be asked before embarking on a new automation project, including information on requirements, common mistakes, and different automation architectures. The full paper can be downloaded from www.kaner.com.

Test automation must be viewed as an investment. Your organization puts money and person hours into an automation project and expects value in return. This value may be in reduced time to market, in better quality (i.e. fewer bugs shipped), or perhaps in other, less tangible ways. There are times when automation is definitely indicated; for example, when legal requirements must be met, when the effect of software failure is life threatening, etc. Other times, automation should be seen as a choice with both positive and negative effects and a decision on its use made accordingly.

In my opinion, there is always one over-riding reason to choose to use test automation. That is to free up the manual testers to do that which they are so good at: manual testing. Unfortunately, manual testing is full of boring, mind deadening, repetitive tasks which must be done. Creating data, verifying that the environment is ready for test, basic flow tests, etc. -- these are tasks which automation can do well. By offloading this drudgery to a computer tool, which does not mind repetitive tasks, we can free up our manual testers to do the exploratory testing where most bugs are going to be found.

Choosing a tool: It is my experience that any of the major automation tools can be programmed to do just about anything to any application under test (AUT.) The operative word here is programmed, however. Each tool that I have used has both strengths and weaknesses. Programming around a weakness of the tool costs extra time and effort (translated: a direct dollar cost.)

Therefore, selecting the correct tool for your applications and environments means spending more than just a few minutes listening to a canned presentation from a slick sales person. Insist on a proof of concept (POC) using your application and environment(s); and be sure to get your people involved. In the tool, work your way through a representative portion of your application, concentrating especially on custom controls and clever programming techniques that your developers use. Determine which portions of your applications will have to be automated using programmed work-arounds which are required when the tool does not understand the widgets. Do the same POC



with several tools. Since all tools can be programmed to work with any controls, choose the tool that will cost you the least in work-around time. Remember that the initial cost of the tool is just the beginning; try to evaluate the entire cost of the tool considering the time and effort it will cost your organization over its life.

And just a bit of heresy. One of the worst reasons to purchase a tool is that it (theoretically) integrates with other tools. Far too often I have seen the purchase decision for a tool made based on the integration into a suite of other very cool tools. Invariably, however, those other tools do not get used because they don't fit in with your processes and actual development techniques. Buy an automation tool only because it works best with your applications, environments, and processes.

Who uses the tools? Notice in my hypothetical scenario, we have given the newly purchased testing tool to an experienced tester. But why?

A test automation tool, stripped of all the glitz and added on features, is nothing more than a programming language and compiler (or interpreter.) No magic, no smoke and mirrors; just a programming environment. The recording process is done by an associated application that taps into the operating system (OS) innards and writes code based on actions the application performs. The code generated is, of course, in the tool language.

Now, generally, a senior tester is a tester by choice. For whatever reason, the person did not want to be a code developer or business analyst; they wanted to test. How does it make sense in any way to give that person a programming tool and expect them to be happy and productive using it? I will make the same assertion I made in the last column: there are developers and testers, code makers and code breakers. It is a mistake to view test automation as anything different than software development. The actual automation process (i.e. creating, debugging, and maintaining scripts) is development in every meaning of the term. Why force it on a someone who, by choice, is not a developer.

For test automation to be successful, the automation should be developed using much the same software development engineering practices that are used on any other software development project in your organization. That calls for a set of developer skills which the tester does not likely possess. Instead of trying to take professional testers and turn them into amateur programmers, it is much more cost effective to take experienced (or even relatively new) developers and turn them into automators. Of course, this costs some money; see the definition of investment, above. Unfortunately, the tool vendor sales people tend to over-emphasize the ease of use of these tools, so you will likely need to educate management as to the realities of automation.

Even if your project has a “renaissance tester” who can handle the development portion of the automation, it would be a huge mistake to set all of the testers to creating scripts. Remember that “test” is more than banging keys at crunch time. Planning, risk analysis,



prioritization in the face of limited resources and similar activities are extremely important for the success of the misnamed “testing phase.” In fact, the testing phase begins with the earliest gathering of requirements and ends long after the software ships.

Regression Testing. Generally speaking, the most effective type of testing which can be done with automated tools is regression testing. This makes sense if you realize that playing back a script causes the application to be exercised in exactly the same steps as it was before when the script was created (the very definition of regression.) There is a kind of knee-jerk reaction amongst the testing community when it comes to regression testing – more is better is the unspoken rule, perhaps because we so rarely get the time to do it. Is this view valid, however? Consider that every minute spent on regression testing a feature which did not change is a minute taken from new or changed feature testing. All things being equal, which is more likely to harbor serious errors: something that used to work and was tested previously, or something that has never been tested?

This is definitely not to say that regression testing is not necessary; different test professionals often disagree as to how much is really needed. Risk analysis and test prioritization should determine the correct amount of regression testing needed. My point here is to not uncritically accept the myth that more is better when it comes to regression testing. In many cases, regression testing is mainly done to provide a “warm fuzzy” before shipping, important but not necessarily earth shattering. Schedule some time and read the Cem Kaner paper mentioned earlier to ascertain exactly what your requirements are before expending too much on the regression testing to the detriment of your other testing.

Dialog and Menu testing. Some of the most ballyhooed features of these automation tools are their ability to do aggregate testing easily. That is, with the touch of one button, you can easily capture an entire menu system or dialog and all of its properties. At test time, the same dialog or menu is checked, with each of the [possibly] thousands of properties being checked with their earlier values. But is this a good idea? Consider the kinds of bugs you can find by making this type of check. Spelling errors, changed colors, sizes, locations: all of the properties of each control are thoroughly checked.

Even if you find changes here, are they bugs? Possibly. I would argue, however, that these types of tests are a waste of valuable automation time for three specific reasons.

1. These tests are essentially unit tests; it should be the responsibility of the developer who changes any code to completely check the actual interface elements. Using the automation tool for unit testing is problematic.
2. Many of the failures which will show up will be of the trivial type. The tool will flag an error which will turn out to be a small change in the position of a control on the dialog – a common tweak which changes no functionality but causes the check to fail. Each false negative requires valuable time to be taken in analysis.



Innovative solutions to maximize your testing investment.

While all of the tools allow you to mask certain properties, that also requires valuable time.

3. The important things to be tested, business rules, calculation accuracy, connectivity problems, etc. will never be found using these kinds of tests. Prioritization theory states that you test those things which, if they fail, will cost the most (in dollars or other ways.)

In general, a serious problem with the automation tools is that they make it very easy to do very bad, wasteful testing. Any test to be run (manually or automated) should be chosen because it needs to be executed based on an identified risk. Don't create automated tests because they are easy to create. They will eventually cost much more than you believe.

Part-time automators: One of the worst disservices you can do to your testers is to try to make them part time automators. It is very difficult to keep a sense of continuity with a task when you are pulled off it repeatedly at crunch time. My general rule of thumb is: if you would not pull a software developer off their task to help out in another area, don't do it to an automator.

Don't despair: test automation can succeed in your organization. Next month, we will start to get down to specifics; how do you structure your organization to be successful using automation.