Agile Test Automation

By James Bach

Summary

This paper describes an agile approach to test automation for medium to large software projects.

Consider creating a small team of test toolsmiths who will apply agile development principles to the problem of test automation. What this means is, on a daily basis, they will pair with testers, witness the test strategy unfold, and find ways to apply a variety of technologies (many of them inexpensive or open source) to specific things that specific testers need. They will do this in measurable, tangible mini-projects that can be completed in no more than a work week, preferably less. The toolsmiths will also work with developers to assure that the products have reasonable testability features. The toolsmiths will apply technology to a range of testing needs, not just test execution.

Principles of Test Automation

In considering how to approach a systematic and productive test automation effort, it may help to keep the following principles in mind. They are culled from the experience of many test automation engineers and managers, over the last 15 years. Some of them I have explained in detail in my article *Test Automation Snake Oil*, and in my book *Lessons Learned in Software Testing.*

1) Test automation cannot duplicate human testers.

Human testers, even ones who have no special skill or training, are capable of doing and noticing things that no conceivable test automation can do or notice. It's true that, even with its limitations, automation can have substantial value. But, it's usually more productive to think of automation as extending the reach of human testing, rather than replacing it. Effective automation efforts therefore begin with effective thinking about testing. Absent good test strategy, automation typically becomes just a lot of repetitive activity that rarely uncovers a bug.

2) Test automation is more than test execution.

Most people who hear about test automation immediately envision some variation of "testing while we sleep." In other words, they want the computer to run tests. This is indeed one useful kind of automation. But, there's so much more to it. For instance, any one of the items in the list below can be automated to some extent, even if the other items remain manual processes.

- *Test generation (data and script generators).* Tools might create specialized data such as randomized email messages, or populate databases, or generate combinations of parameters that we'd like to cover with our tests.
- *System configuration.* Tools might preserve or reproduce system parameters, set systems to a particular state, or create or restore "ghosted" disk drives.
- *Simulators.* Tools might simulate sub-systems or environmental conditions that are not available (or not yet available) for testing, or are too expensive to provide live on demand.
- *Test execution (harnesses and test scripts).* Tools might operate the software itself, either simulating a user working through the GUI, or bypassing the GUI and using an alternative testable interface.
- *Probes.* Tools might make visible what would otherwise be invisible to humans. They might statically analyze a product, parse a log file, or monitor system parameters.
- *Oracles.* An oracle is any mechanism by which we detect failure or success. Tools might automatically detect certain kinds of error conditions in a product.
- *Activity recording & coverage analysis.* Tools might watch testing as it happens and retrospectively report what was and was not tested. They might record actions for later replay in other tests.
- *Test management.* Tools might communicate test results, organize test ideas, or present metrics.

3) Test automation is vulnerable to instant obsolescence.

Software projects revolve around production code. Test code is not production code. So the priorities of a typical software project allow production code to change even when that breaks test code. This is normal, and generally speaking it's a reasonable, economically justified behavior.

Assuming that the product will change, you have basically two options: invest in test code to be robust in the face of such change, or make test code so inexpensive and easy to fix that you don't mind if product changes break it.

4) Test tools are many and varied.

Most people, especially managers, think of test tools as those tools on the market that are sold as "test tools". They tend to be quite expensive. But, in fact, almost anything can be a test tool, and many utilities sold for other purposes are especially useful for testing. Some tools are free, some are provided in repositories for developers. The resources I use include:

- MSDN Universal Library
- Any Microsoft development tool (they always include useful utilities)
- Microsoft compatibility toolkit and other free tools (www.microsoft.com)
- Web-based web testing resources (HTML checkers, accessibility analyzers)
- Windows Resource Kits (available from Microsoft)
- Scripting languages (e.g. Perl, Ruby, TCL) and associated libraries
- Shareware repositories (www.download.com)
- O/S monitoring tools (www.sysinternals.com)
- Open source testware (www.opensourcetesting.org)

- Spyware for monitoring exploratory tests (www.spectorsoft.com)
- The cubicle next door (often people a few paces away have already created the necessary tools)

5) **Test automation depends on product testability.**

Some kinds of test automation become cost effective only if the product under test is designed to facilitate testing—or at least not designed to be exceptionally difficult to test.

The two big pillars of testability are controllability and observability. GUI's are not very testable compared to command-line, batch, or programmatic interfaces, because the latter provide greater controllability. Similarly, observability is increased when log files, probes, or queriable databases are available to augment information visible through the GUI.

6) **Test automation can distract you from good testing.**

Sometimes the technological focus of test automation can lead to a situation where automators become cut off from the mission of software testing, and produce a lot of tools and scripts that might look good, but have little value in terms of a coherent test strategy that makes sense for the business.

Agile Test Automation

Over the last ten years, the "agile development" movement has emerged from obscurity and has begun to have a significant impact on the culture of software development in the U.S., particularly among commercial software companies. I have been involved in this movement since the late eighties. Most of my writing comes from an agile perspective (see my 1994 article *Process Evolution in a Mad World*, http://www.satisfice.com/articles/process-evolution.pdf). One form of agile development is "Extreme Programming" which is a collection of 12 practices promoted by Ward Cunningham and Kent Beck.

One expression of the agile mindset is contained if the so-called Agile Manifesto (http://www.agilemanifesto.org/principles.html). Among the elements of agile development is a practitioner-centered methodology, very close contact with customers, numerous small milestones, and a welcoming attitude about change. Agile development gives management and customers more control over the development process while preserving the creativity and productivity of technical work.

Agile test automation is the application of agile development principles to the test automation problem.

Principles of Agile Test Automation

I propose these as operating principles for getting strong, ongoing results from test automation:

- Test automation means tool support for all aspects of a test project, not just test execution.
- Test automation is directed by testers.
- Test automation progresses when supported by dedicated programmers (toolsmiths).
- Test toolsmiths advocate for testability features and produce tools that exploit those features.
- Test toolsmiths gather and apply a wide variety of tools to support testing.

- Test automation is organized to fulfill short term goals.
- Long term test automation tasks require a compelling business case.

In the absence of dedicated toolsmiths, the rest of the principles still apply, you just apply them with a part-time toolsmith.

By “short term”, I mean 40 work hours or less. Long term is anything longer than that. A longer task might be achieved in short term increments, as long as each such increment is a deliverable, useful unit of work.

By “directed by testers” I mean that the progress of test automation should be measured in terms of how it solves problems for testers and test managers. Test toolsmiths pair with a tester “client” who has a problem and requests help. The immediate goal of the toolsmiths is to render that help in terms acceptable to the tester.

Agile Automation Tasks

Test toolsmiths do the following kinds of things:

- Respond rapidly to requests for assistance from testers.
- Seek out test productivity problems.
- Investigate possible solutions in concert with testers.
- Apply technology to improve the test process.
- Advocate for specific testability features in products.
- Research available tools and learn how to use them.
- Gather tools that developers or testers produce.
- Review upcoming product plans to assess automation possibilities.

Composition of the Agile Automation Team

At its simplest, the team could consist of a single toolsmith, co-located with and supporting an arbitrarily large group of testers. But if there are too many testers, there will be a large backlog of requests, and much missed opportunity for test productivity improvement.

On the other hand, if there are a lot of automation people and just one tester, that’s not necessarily a problem as long as the toolsmiths are also ready willing and able to function as testers (i.e. take primary responsibility a particular product, sub-system, or feature set and do whatever is necessary to test it). If no automation tasks are important enough, and testing is needed, then have toolsmiths do testing. Now, diverting toolsmiths to testing is a bad idea if you have a big automation project that you expect them to do in a certain timeframe. But if the automation tasks are narrowly focused and rapidly carried out, that is not a problem.

Avoid toolsmiths who insist on being programmers only, who remain untrained or unskilled at testing. The natural question arises: why not make everyone in the test team a toolsmith? There are two reasons for that: good toolsmiths are a rare and expensive commodity; and the test team benefits from having a variety of skills and backgrounds, rather than just programming backgrounds.

Agile toolsmiths are not really a team apart from the testers, they are a team *within*. They are testers with certain skills that make them better suited to help other testers test faster and better rather than themselves taking primary responsibility for testing a product.

How Agile Automation Progresses

Test automation justifies itself in terms of how it…

- makes possible new kinds of useful testing,
- makes more productive, faster, or reliable the kinds of useful testing already being done,
- or otherwise contributes to a more successful and cost effective test project.

Test automation is unjustified to the extent that it makes testing possible or faster that isn't needed to begin with, isn't worth the investment, or can't be delivered when it's needed.

Agile automation progresses according to this cycle: understand how the testing is done, identify some technology that would significantly improve it in the opinion of the tester, deliver that solution in less than a week, repeat. This cycle is a bit simplistic, because in reality, a delivered solution often needs tweaking and improvement after the fact.

This process could be managed with five lists, plus a few background tasks. These lists should be very visible, perhaps on a web site. The lists are as follows:

- *Request list.* This is a list of new requests from customers (testers).
- *Assignments list.* This is a list of what each toolsmith is currently assigned to do.
- *Delivered list.* This is a list of solutions that are currently in use by the test team. Each item on the list should include a brief description and statement about the positive impact that solution has on testing productivity.
- *Maintenance request list.* This is a list of solutions that need improvement. It should be divided into two parts: critical maintenance and enhancements.
- *Obstacles list.* This is a list of productivity problems in testing that remain unsolved because they require expensive new tools, substantial testability improvements, or more work than can be done in a short term.

Background tasks:

- Pair with testers to understand how the product is tested.
- Review upcoming product specs and technologies, to understand the technical testing issues.
- Work with testers and test managers to find reasonable ways to assess and report productivity.

How Testers Work with Toolsmiths

Toolsmiths maintain visibility in the test team. Any tester in the team, even if he doesn't personally work with a toolsmith, will see them working with other testers on the team.

Testers are the customers of the toolsmiths. Any solution not delivered in the opinion of a tester is not considered delivered at all.

A tester may request help at any time from the test automation lead. In practice, the automation lead should arrange to visit testers regularly and check in on what the toolsmiths can do for the testers. Alternatively, the test manager may direct specific testers to work with the automation team to solve some problem.

Requests may include things like:

- How do I test this new thing?
- How can I see what's happening inside the product?
- How do I know if this test passed or failed?
- Is there a way I can automatically perform this operation?
- Is there a way to make this bug easier to reproduce?
- Help me investigate this bug.
- Here's a test I want to execute. Can you help me do 1000 variations of it?
- How well have I covered this product?
- I want to stress test this product. Are there any tools that do that?

Having made a request, a tester should expect the automation team to respond promptly. The tester is expected to share the work with the toolsmith. They solve the problem as a pair.

How Developers Work with Toolsmiths

Developers are also the customers of the test automation team. When a developer creates a special tool for his own purposes that might also be useful for testers, it is in that developer's interest to give it to a toolsmith for deployment as a test tool. The toolsmith then takes on the maintenance and technical support task for that tool.

Developers will probably see more of the toolsmiths than of the other testers. The toolsmiths have the technical ability to learning about the inner workings of the products and to discuss testability features in a productive way. They can more efficiently and consistently pass that along to the testers than if every tester sought to engage every developer in a similarly deep conversation.

Toolsmiths are in a better position to conceive of and exploit testability features. This is important, because in practice the onus falls on the test organization to think through testability issues.

How Management Works with Toolsmiths

Management has a strong interest in getting the best bang for the buck. Test automation work is one step removed from testing, which is already itself a step or two removed from revenue streams, so automation must justify itself in terms of how it improves testing, or otherwise improves the business, such that it is worth the price paid.

Management works with toolsmiths mainly by asking for and promptly getting progress reports that answer certain standard questions like:

- What are we getting from test automation?
- What does it cost us?

- Considering the cost, are we getting enough from test automation?
- What bad things would happen if we suspended further investment in test automation?

The task lists mentioned above are designed in part to provide a basis for answering these questions. Another thing testing could do is produce a balanced dashboard of metrics that concisely convey a meaningful impression of the productivity of testing.

Here are some potential metrics and other specific observations that may help shed light on the productivity of testing, and therefore on the value of the automation effort:

- *Fixable bug find rate.* (excludes cannot reproduce, tester error, not a bug, and other noise categories) Many factors affect this number. It is an exceedingly dangerous metric to use on its own. However, sometimes a big improvement in testing can be corroborated by a significant jump in this metric.
- *Find rate spikes.* This is a short term jump in the bug find rate due to a new kind of testing that is deployed. A key factor to look at, here, is whether the bugs found look like they would not have been found without the new test approach.
- *Fix rate*. Increasing the percentage of bugs found that get fixed may tell us something about the quality of the bugs, the timeliness of reporting them, and the quality of the relationship between development and testing. All of these may be positively impacted by test automation.
- *Testing made possible.* Test automation makes some testing possible that humans could not otherwise do. This “metric” is really a list that the toolsmiths maintain for periodic management review. The question management must ask is “Is this possible testing actually happening?” and “Is this possible testing producing results, such as increased confidence or increased bug find rate?”
- *Nightmares averted.* This is a list of serious problems that were avoided in whole or part because of test automation. This can include test activities designed to avert a repeat of some earlier disaster.
- *Test cycle time.* How much time does it take, assuming no significant bugs are discovered, to gather sufficient evidence to justify the release of the product? Cutting down on this time improves the agility of the entire development process.
- *Development/test cycle time.* In practice, testing is tied to development. We want to see a decrease in the overall time it takes to develop a new product relative to the staff we put on it.
- *Management confidence*. This is not so much a metric as a factor. The mission of testing is to provide management with the information it needs to make sound business decisions. Does management see an improvement in the quality of information provided by testing? Sometimes a before and after comparison helps make the point.
- *Technical support costs.* Ultimately, a key factor in assessing the test effort is to check how well our sense of quality prior to release compares with the empirical experience of quality afterwards. This is why technical support is critical. Set up a system to regularly communicate to the development project what kinds of problems are being seen in technical support. Look at the number of calls relative to number of customers. Look at how call time might be cut down by giving tech support better diagnostic tools such as test toolsmiths might produce.
- *Customer retention.* If the quality is better, presumably, customers will be happier.

- *Reputation.* This might seem intangible, but it isn't. Reputation is expressed in the stories told about the test group among other people in the company. Reputation is driven by stories, so you need to have visible, positive, and exciting wins as frequently as you can. Also, the more the automation team and the test team say "yes" to requests for help, the better their reputation will be.
- *Delivered solutions.* Look for a steady increase in the number of delivered solutions by the automation team.

Risks of Agile Automation

- Automation may not significantly improve test productivity unless the testers know how to test.
- Productivity metrics such as number of test cases created or executed per day can be terribly misleading, and could lead to making a large investment in running useless tests.
- The testers may not want to work with the toolsmiths. Members of the agile automation team must be effective consultants: approachable, cooperative, and resourceful, or this system will quickly fail.
- The toolsmiths may propose and deliver testing solutions that require too much ongoing maintenance relative to the value provided.
- The toolsmiths may lack the expertise to conceive and deliver effective solutions.
- The toolsmiths may be so successful that they run out of important problems to solve, and thus turn to unimportant problems.

Agile Automation in Action

- **A Tester's Request.** One team completed a radical improvement in the productivity of installation testing— a *300%+ improvement* in the throughput of packages through his team. It started when their tester complained about the amount of time it took to re-image Macintoshes for testing. The team leader brought in a consulting programmer who then worked with the tester to redesign the process in about two weeks. The consultant looked at several different tools that might help and looked at the Windows side as well as the Macintosh. The team eventually employed a combination of tools, some purchased and some off the shelf to solve the problem. The team worked together to bring the solution to light, but the outside toolsmith was the catalyst that brought it together. It is significant that the team considered using Rational Robot (on the Windows side) to automate the process, but decided that it was the wrong tool for the job. I interviewed the tester about how this innovation helped him in his daily work, and he was extremely enthusiastic in his praise for the new system. This process was *narrowly focused* and driven by the *needs of the tester* (in fact, the needs of the business) and involved a *pairing of programmers and testers* to make it work.

- **Installation Checker.** I installed a product on my system. It almost immediately asked for permission to update itself (as expected) and downloaded the latest version from the Web. Unfortunately, it subsequently crashed every time I tried to start it. After several reboots, I finally went to the Web again and reinstalled it. The problem went away. That raised a question: Is there a tool that would allow a tester to tell if the product was properly installed on a system? In this case the answer appears to be no. Well, it wouldn't be difficult to write such a tool. And doing so would make installation testing for that product a little faster and a lot more reliable. Also, whenever testers experienced strange intermittent failures with any aspect of it, they could run the installation checker to assure that the product had not been corrupted. The difficulty of investigating intermittent bugs is something a good toolsmith can go a long way to solve. *Agile test automation can improve the probability that important bugs get reported in a timely manner.*

- **DB Dumper.** While consulting on one project I heard that the product used a Microsoft Access database back end. It occurred to me that I could write a Perl script to take snapshots of the database, and maybe that could be used as a test probe. By borrowing code from the Perl Cookbook, I had the script written and running in a minute or two. I found that some of the data in the database is packed in binary format that my Perl ODBC library can't read, but most of it I can get to easily. I demonstrated the script to my client and showed how it could be used to create a series of diagnostic snapshots during testing, or how it might be adapted into a test oracle that automatically detects certain kinds of corruption in the database.

  Now, this is a bona fide nifty testing idea, but it's just a toolsmith's fantasy until and unless it is used to test the product. Still, this demonstrates three aspects of agile automation: *a good toolsmith has a broad knowledge of tools that might help* (like scriptable access to a database via ODBC), *agile automation begins with very simple solutions* (like a one minute script), and *sometimes we need developer support* (like a means to decode the binary representation of some records in the database).

- **Client Test Coverage.** On one project that involved an application that sends email, I was looking at some of the manual test scripts. The test scripts referred to a matrix that described kinds of messages to send and receive. I noticed that a DB dumper script like the one mentioned above could be modified to provide a report on which parts of the test matrix had been satisfied. That's called a coverage analysis tool. Coverage analysis tools help free testers from testing paperwork by automatically reporting on what has been tested. That way, the testers and test manager don't have to worry that a feature was accidentally not tested.

  As the testers explored this possibility with me, another idea emerged: maybe we could automate the sending of a wide variety of messages so that a tester could, on demand, elect to receive a thousand different ones. That led to an idea for establishing a message sending server machine in the test lab, with a repository of test messages. We estimated this could be set up and running within a couple of days.

  This is an example of how ideas emerge spontaneously when toolsmiths work with testers. It's also an example of thinking about test automation in a broad way (coverage analysis; a facility for automatic message sending) instead of only in terms of simple "test cases".

- **Ten Minutes with One Tester.** I sat down with a tester to see how he tested his product. He had barely begun when I noticed he was installing it without the use of tools to check that the right files had been installed. So, I showed him InCtrl5, a freeware tool that takes snapshots of a system before and after certain events, and reports of changes to files and registry settings. We used it.

  Then he showed me a bug that he wished to investigate. Apparently a window was being rendered behind another window when it should have been in front. His theory was that the window in front was receiving some kind of setfocus message that brought it to front at the wrong time. He wondered if I knew of any tools that would help test that hypothesis. I immediately thought of Spy++, a tool for watching Windows messages. It ships with Visual

C++. (A toolsmith must invest a certain percentage of time gaining general knowledge about all kinds of tools and where to get them, and here it paid off.) We scrambled around a little looking for Visual C++. Eventually, the tester located a developer and convinced him to show us Spy++. Not only did we get the tool, but we also had an interesting conversation with the developer about how the tool might or might not help us do our investigation.

This shows that agile automation is sometimes about researching solutions, and how that process can be an activity that draws in developers to help the testing cause.

Transitioning to Agile Test Automation

1) **Establish and announce the role of the test automation team to developers and testers.**
2) **Establish a reporting system and a dispatching system for the automation team.**
3) **Make an inventory of known testware within the company.**
4) **Make a quick survey of testware available outside the company.**
5) **Have toolsmiths sit with testers, understand how they test, and identify test automation opportunities**
6) **Create the experience for the testers of making a request and seeing it rapidly granted.**
7) **Review the progress of the automation effort no less frequently than every two weeks.**
8) **Continue to improve the training of the testers, so that they are better able to conceive of sophisticated test strategies.**