The Democratization of Test Tools

by Lisa Crispin

Lisa Crispin says the future of test automation is working together.

Democratization is the transition from authoritarian or semi-authoritarian systems to democratic political systems, where democratic systems are taken to be those approximating to universal suffrage, regular elections, a civil society, the rule of law, and an independent judiciary. (Wikipedia)

Some may not find it as dramatic as, say, the fall of the Berlin Wall, but from where I sit the world of test automation has undergone a revolution. I'm on a team in which testers, programmers, and product directors collaborate to automate all of the regression tests for each new feature we deliver. Tests and production code are delivered together. From my vantage point, I work in the Republic of Testing, taking advantage of test tools developed by teams just like us.

Back in the '90s, when I automated tests, I was limited to tools that worked through the graphical user interface (GUI). Programmers were busy with their own work and weren't available to give us hooks for testing different layers of code. I was lucky in that I usually worked for managers who bought the tester-recommended tools and gave us time and training to master the proprietary scripting languages. Only on one occasion did someone tell me, "We spent $75,000 on this test tool, so you'd better use it." If we automated 10 or 20 percent of our testing, we were grateful.

The drumbeat of the revolution first sounded for me in 2000, when I joined my first agile team. It was amazing to see how interested the programmers were in testing. My teammates, all programmers, learned how to automate unit tests—before even writing the code to make them pass—using an open source tool called JUnit.

Imagine—someone on a team like ours had needed a robust, programmer-friendly, easy-to-use unit test tool; written it; and shared it with the world! Our team could just decide, by consensus, to use it! The test results even turned red or green to show which tests passed! How could it *not* cost $75,000?

But automating acceptance tests was another story. I hadn't heard of any open source tools for automating tests beyond the unit level, except for JUnitPerf. Then again, it didn't occur to me to look for them! I stuck to my vendor tool and listened to my programmer teammates grumble about having to learn its scripting language. More revolution was fomenting.

I'm certain other development teams were grumbling about test tools they were forced to use, or perhaps about having to do manual regression testing. An Orange Revolution of automation gained momentum. My team heard about new open source tools, such as HTTPUnit, and tried them out. With the whole team working on test automation, my team felt good about automating as much as 50 to 60 percent of our acceptance tests, using a combination of open source unit test tools and vendor tools. Sometimes a tool got voted down during a team meeting and another was brought in. Short iterations gave us room to experiment.

Participating in mailing lists and attending user group meetings and conferences, I met many other agile development teams that were feeling a need for more than just unit test automation. And test automation was no longer a chore just for people in a tester role; entire software development teams were hugely invested in successfully automating tests that would let them safely and speedily change code without sacrificing quality. Testing through the GUI is difficult and expensive, and I could imagine many programmers saying, "Gee, it would be pretty easy to test the code if we bypass the GUI." They started writing test frameworks and harnesses. Other tools, such as automated build tools and source code control systems, were written to support the testing and provide quick feedback.

Some teams happily used their homegrown tools to automate their tests. Some frameworks and tools grew into something bigger, and developer communities formed across the world to contribute to and support them. Collaborative software development tools such as Framework for Integrated Tests and FitNesse grew out of agile projects. These tools met the needs of people in different roles and allowed them to communicate better and work together. Instead of having a tool imposed upon them, business experts, testers, and programmers had a framework to specify

and automate tests together. Here was a way to both communicate expectations from the customer side and verify that those expectations were met.

At some point, though, you still have to test the GUI. A team developing a Web-based application needed a tool that would give them feedback from the user perspective quickly enough to keep up with the pace of development. Efforts with commercial tools failed, so they came up with a homegrown tool—the open source tool Canoo WebTest—that let them specify rather than program tests. Other teams noticed that some programming and scripting languages lent themselves to the needs of test automators as well as exploratory testers. Web Application Testing in Ruby (WATIR) is just one example of what happens when people with both programming and testing skills get together and solve automation problems.

These are just a few examples. The scenario has repeated itself many times in the past few years.

If the programmers join in, automating tests is a collaborative effort. Not only do we get automated regression tests, but we also have an important reason to sit down and talk every day about the features we need to deliver. We have to talk to the business stakeholders too, and now we have a process and framework to capture their input and give the whole team continual feedback.

Test automation isn't something imposed on teams anymore. The days of "We've spent all this money on this tool to fix our automation problem, so use it, by golly!" are (or should be) history. Technical members of the team may decide to code their own testing harness, select one or more open source or commercial tools, or customize those tools with homegrown solutions. They collaborate with business stakeholders to encapsulate requirements into executable tests. Tools aren't selected by an autocratic manager but by team consensus. This might involve vigorous debate over the relative merits of various approaches, but the outcome is much more productive.

I'm a tester on a team where automating 100 percent of the regression tests for each new feature produced has become routine. Test automation tasks, using different tools for different purposes, are part of every iteration and aren't just the job of the team's testers. Tests run via continual builds throughout the day, so we learn right away if a code change breaks an existing feature. Pretty civilized, if you ask me.

The future of test automation is everyone working together to find good solutions. Everyone on the team has an equal stake in quality. I'm not saying test automation is easy. It's not, but it's a lot easier when an entire team—programming, testing, and analysis—takes it on. Every issue of this magazine has articles that help lead all of us into this bright testing future. Rip those barricades down, and start solving test automation problems as a team! **{end}**

Lisa Crispin has been a tester on agile teams developing Web-based applications since 2000. She co-authored Testing Extreme Programming *(Addison-Wesley, 2002) with Tip House and is a regular contributor to* Better Software. *Read more about Lisa's work at http://lisa.crispin.home.att.net.*