Investing in Software Testing: Manual or Automated?

Abstract

Automated test tools are powerful aids to improving the return on the testing investment when used wisely. Some tests inherently require an automated approach to be effective, but others must be manual. In addition, automated testing projects that fail are expensive and politically dangerous. How can we recognize whether to automate a test or run it manually, and how much money should we spend on a test?

Introduction

In the last article, I wrote that not only do we have to test the right things, but we also have to test them the right way in order to get a positive return on our testing investment. Static, structural, and behavioral test techniques are each good for finding certain kinds of bugs during certain phases of the project. However, there's another decision we have to make as test professionals for each test suite: Automated or manual?

Some tests are well-suited for automation—indeed, some kinds of tests can't be done manually in any meaningful way. Other tests, conversely, as a practical matter are either most effective when done manual or only doable manually. Test automation projects are just as risky as system development projects, and just as many fail. As with test techniques, selection of the appropriate choice in this regard will have a serious impact on the return on test investment.

When Test Automation Makes Sense

Let's start with the tests that ideally are automated. These include:

- Regression and confirmation. Rerunning a test against a new release to ensure that behavior remains unbroken—or to confirm that a bug fix did indeed fix the underlying problem—is a perfect fit for automated testing. The business case for test automation outlined in *Software Test Automation* is built around this kind of testing.
- Monkey (or random). Tests that fire large amounts or long sequences of data, transactions, or other inputs at a system in a random search for errors are easily and profitably automated, as described in Noel Nyman's article, [continue—get ref for his dumb monkeys article on www.stickyminds.com]
- Load, volume, and capacity. Sometimes, systems must support tremendous loads. On one project, we had to test how the system would respond to 50,000 simultaneous users, which ruled out manual testing! Two Linux systems running custom load generating programs filed the bill.

- Performance and reliability. With the rise of Web-based systems, more and more automated testing is aimed at looking for slow or flaky behavior on Web systems.
- Structural, especially API-based unit, component, integration. Most structural testing involves harnesses of some sort, which brings you most of the way into automation. Again, the article I wrote with Greg Kubaczkowski, “Mission Made Possible,” provides an example.

Other tests that are well-suited for automation exist, such as the static testing of complexity and code standards compliance that I mentioned in the previous article. In general, automated tests have higher upfront costs—tools, test development, environments, and so forth—and lower costs to repeat the test.

When to Focus on Manual Testing

High per-test or maintenance costs are one indicator that a test should be done manually. Another is the need for human judgment to assess the correctness of the result or extensive, ongoing human intervention to keep the test running. For these reasons, the following tests are a good fit for manual testing:

- Installation, setup, operations, and maintenance. In many cases, these tests involves loading CD-ROMs and tapes, changing hardware, and other ongoing hand-holding by the tester.
- Configuration and compatibility. Like operations and maintenance testing, these tests require reconfiguring systems and networks, installing software and hardware, and so forth, all requiring human intervention.
- Error handling and recovery. Again, the need to force errors—by powering off a server, for example—means that people must stay engaged during test execution.
- Localization. Only a human tester with appropriate skills can decide whether a translation makes no sense, is culturally offensive, or is otherwise inappropriate. (Currency, date, and time testing can be automated, but the need to rerun these tests for regression is limited.)
- Usability. As with localization, human judgment is needed to check for problems with the facility, simplicity, and elegance of the user interface and workflows.
- Documentation and help. Like usability and localization, checking documentation requires human judgment.

There’s no return on investment in trying to automate these kinds of tests, typically. One of my clients once spent thousands of dollars and staff hours trying to automate configuration and compatibility tests, only to give up months into the effort.

Wildcards

In some cases, tests can be done manually, automated, or both.

- Functional. Functionality testing can often be automated, and automated functional testing is often part of an effort to create a regression test suite or smoke test. However, it makes sense to get the testing process under control manually before trying to automate functional testing. In addition, you'll want to keep some of the testing manual. As Cem Kaner, James Bach, and Bret Pettichord write in *Lessons Learned in Software Testing*, manual testing is adept at finding different kinds of functional bugs than automated testing.
- Use cases (user scenarios). By stringing together functional tests into workflows, you can create realistic user scenarios, whether manual or automated. The trick here is to avoid automation if many workflows involve human intervention.
- User interface. Basic testing of the user interface can be automated, but beware of frequent or extensive changes to the user interface that can incur high maintenance costs for your automated suite.
- Date and time handling. If the test system can reset the computer's clocks automatically, then you can automate these tests.

Higher per-test costs and needs for human skills, judgment, and interaction push towards manual testing. A need to repeat tests many times or reduce the cycle time for test execution push towards automated testing.

Financial Considerations

For any given test, manual or automated, we can do a quick, first-order cost-benefit analysis. First, calculate the cost to design and implement the test. Next, add the cost to repeat the test multiplied by the number of times the test you need to repeat it.

The benefit of the test can be calculated using the cost of quality technique I discussed in the first article, but you might find it hard to come up with precise bug estimates for any given test. If so, consider the following approach. Estimate the likelihood of bugs in the feature or function you plan to test, in rough fractions like 25%, 50%, 75%, or 100%. Now, estimate the cost of such bugs, on average. The likelihood of the bugs multiplied by the cost of the bugs gives you the benefit. If the benefit exceeds the cost to create and repeat the tests, then you have a business case for the test.

For example, suppose that you plan to spend $25,000 on tools and labor to create a performance test for your e-commerce application. You expect to spend about $1,000 in maintenance and execution costs per test run, and you expect to run the test ten times over the course of a year. The test will cost $35,000. Suppose that, based on your study of your system and what you know about e-commerce sites, you think there is a 25% likelihood of slow performance.

Suppose you expect 100,000 potential customers per year, at an average profit of $10 per customer. If half of your customers will abandon their transaction—and your site—if transactions are handled too slowly, then your estimated loss is $125,000 from poor performance. Spending $35,000 to mitigate the chances of such a loss makes sense.

If you expected only 10,000 potential customers, then you'd need to figure out how to run the test for $12,500 or less, for example by outsourcing a single run. If a test can be run manually as well as with automation—though performance tests typically can't—then you might want to consider the cheaper manual test, performed less frequently. Alternatively, you'd want to see if people were comfortable skipping the test based on your analysis.

Of course, there might be other considerations that would affect your analysis. Perhaps the word-of-mouth issues that would accompany bad performance make spending more money a priority for sales and marketing. However, if there's no real business case for the test, then you should not expect to get a positive return on investment from it, regardless of the other benefits.

Reasons to Be Careful with Automation

Automated testing is a huge investment, one of the biggest that organizations make in testing. Tool licenses can easily hit six or seven figures. Neophytes can't use most of these tools—regardless of what any glossy test tools brochure says—so training, consulting, and expert contractors can cost more than the tools themselves. And then there's maintenance of the test scripts, which generally is more difficult and time consuming than maintaining manual test cases.

Capers Jones, in *Estimating Software Costs*, cites failure rates for large, complex system development efforts as high as 50%—or higher. A test automation project can be just as complex as developing software, and, indeed, Dorothy Graham and Mark Fewster cite similar failure rates for automation projects in *Software Test Automation*. Two anecdotes will serve to illustrate this point.

One of my clients will not allow me to use the phrase "test automation" in front of him. When I do refer to it, I jokingly call it "the A word." His organization spent over half a million dollars on tools and consulting to try to automated their complex software package. While the tool vendor, the vendor's salesperson, and the consultant all cashed their checks, what my client has to show for that investment is a maze of unmaintainable automated test scripts and some nice-looking test tool boxes on a shelf.

These failures don't necessarily have to do with incompetence of the automation itself. Remember my cautionary tale in the second article, when we focused exclusively on (automated) functional and regression testing to the exclusion of (manual) installation and setup testing. This was a well-designed, highly maintainable test system, one of the most elegant test automation solutions I've ever worked on. But it didn't solve the real test problem, delivering an accurate assessment of the users' and customers' experiences of quality.

Sadly, these are not isolated incidents. Many who've tried their hands at test automation have unhappy stories to relate. And gravely, when an investment this big goes awry, the loss of credibility for the associated parties is huge. People get fired for losing this kind of money. You'll want to position yourself for success when embarking on test automation by doing your homework with a book like *Software Test Automation*. Also, keep in mind that much of the work associated with automated testing must be done manually, such as the design of the test actions and data, the calculation of the expected result, and the analysis of the results. Successful test automation efforts don't focus on eliminating the test team, they focus on doing a more effective and efficient job of testing with the resources they already have.

And Context Matters, Too

The last two articles have run the risk of making you think that all the matters is testing prowess, picking the right techniques and tools. But that's not true. Context matters. When do we start testing? Who does the testing, and what makes them a fit? Does the test team understand where they fit into the project, and vice versa? In the next article, we'll look at the importance of context through a topic I call pervasive testing.

Author Biography

Rex Black is President and Principal Consultant of RBCS, Inc. (www.RexBlackConsulting.com), a consultancy that provides testing experts world-wide, serving clients such as Bank One, Cisco, Hitachi, IMG, and Schlumberger in consulting, training, and hands-on implementation. He's written *Managing the Testing Process, Critical Testing Processes Volumes I* and *II*, and numerous articles, along with presenting papers and keynote speeches at international conferences.

Bibliography

R. Black and G. Kubaczkowski, "Mission Made Possible," Software Testing and Quality Engineering magazine, Volume 4, Number 4.

C. Jones, *Estimating Software Costs*. McGraw-Hill, 1995.

C. Kaner, J. Bach, and B. Pettichord, *Lessons Learned in Software Testing.* Wiley, 2001.

D. Graham and M. Fewster, *Software Test Automation*. Addison-Wesley, 1999.