SUBJECT7
QUALITY IS A CULTURE

What are some Automation Misconceptions?

By Matt Heusser and Joe Kiriacos

A few years ago I was working with a company that wanted to "move to test automation." The director of HR brought me in as a consultant to help with recruiting, pick the tool, help with the proof of concept, and so on. It was, for the most part, a good assignment.

One thing that surprised me was the confidence everyone had. Automation was the way, the truth, and the future -- although no one had ever done it. They didn't know how. They had heard, however, that all the big companies were doing it. And, to be fair, they correctly saw regression testing as a boat anchor that was delaying the frequent release of software. The company had no solid measurement for success, unclear goals, a belief in something they had never seen.

Sounds like a great prescription for success, doesn't it?

On the other hand, that company was actually off to a better start than many. They had clear priorities and were willing to invest time and effort. With an understanding of how incremental development works, they could design an experiment with the intent to figure out what is next.

The big risks on that project, on so many projects, were the myths and misinformation. These are the things everyone knows about test automation. All too often the belief is that everyone knows these things. So we say "100% test automation," nod our heads, all mean different things, and go into the world with different assumptions. The results may be tragic, but they are also predictable.

Today I'd like to start about myths and misconceptions in test automation, starting with that recurring legend of 100% test automation.

100% TEST AUTOMATION

Eighteen years ago, when Lisa Crispin co-authored Testing Extreme Programming with Tip House, there was a chapter on Manual Tests. In the body of that chapter were the words "No manual tests."

That chapter was mostly for effect, and they went on to explain what they meant. Reading the follow-up chapter very carefully and talking to Lisa at some length, it is very clear she meant no do-the-same-thing-every-time, follow-these-steps, documented, human manual tests. That is, if the test really is exactly the same thing every time, let the computer do it.

The confusion here is between testing as a document and testing as an activity. Consider, for example, you are in a "show and tell" meeting to discuss new features. A visitor, a senior executive asks "What happens if you leave the middle name field blank and press save - is it required?" Someone tests it and you get information, which could make for a new feature. Now imagine someone said "Wait - stop - that is a TEST. All our tests are automated. I need to automate this. I will let you know by email within a day or two what the software does."

Of course not.

Push on that thread a little bit further, and we realize that exploration will happen, especially as new features are developed. Some small amount of those checks will be institutionalized and saved to run every time. Done well, it might be possible to eliminate all the repetitive work done by humans at regression-test time, at least for new development. Existing systems that have testing ratcheted-on will be a work in progress for a long time.

LESSON

- ✓ Be very careful about rhetoric about "100% test automation" or "automate all the things."

  Dig for concrete examples and timelines.
- ✓ Fight for deep agreement over shallow points, or else
- ✓ that shallow agreement may come back to bite you.

Speaking of a shallow understanding of testing, let's talk about the assumption that automation will cover all possible cases.

AUTOMATION SHOULD COVER ALL POSSIBLE CASES

Imagine a simple auto insurance application. The application uses factors to determine your insurance price. There are different age brackets (5 options), different insurance types (comprehensive collision with 3 different deductibles or standard), the value of the vehicle (3 options), the age of the car (5 options), a good driving record discount, a bad driving penalty, and so on. Even the examples above yield 5*3*3*5*3 choices, or 675 combinations. In reality, the number of possibilities will approach infinite, and that is just to get a simple answer to filling in forms and clicking submit.

A typical application, such as Google Documents, will have an infinite expansion of possible workflow. We call this problem the impossibility of complete testing. Without understanding this, executives who hear "we are now doing test automation" are likely to hear "...and it covers all the possibilities."

LESSON

Don't set up unrealistic expectations. Create a guideline for how deep to automate, and educate your executives.

AUTOMATION IS "FREE"

Sean McMilllan, my old colleague, told me that once a test is created, it is free. You can run it every time without costs. Sean was speaking of unit tests, which tend to be less brittle, and run in milliseconds. GUI driving test tools do not have that impact. Each new regression scenario adds time to the test run (even if you are running parallel executions in the cloud).

Every change to the user interface could cause problems with tests where they need to be "fixed" and re-run. This is a maintenance cost that slows you down. The more tests you have, the more the cost. Another colleague, working for a military group, told me off-record that his entire job was essentially fixing and re-running tests using an automated tool. The job would never end, as it was military defense, but it was incredibly unsatisfying to him.

As it turns out, automation is not free. There is an ongoing maintenance cost. You can work to minimize that cost, but also to pick, again, the powerful few test scenarios. Done well, the value of the powerful few will grow over time, as good tests that rarely need maintenance continue to find important bugs.

LESSON

If you have a great deal of defects, mass inspection through automated tools will be expensive. Instead, improve software quality before testing, so you can afford to have the powerful few tests. Then set reasonable expectations for the defects the software will find.

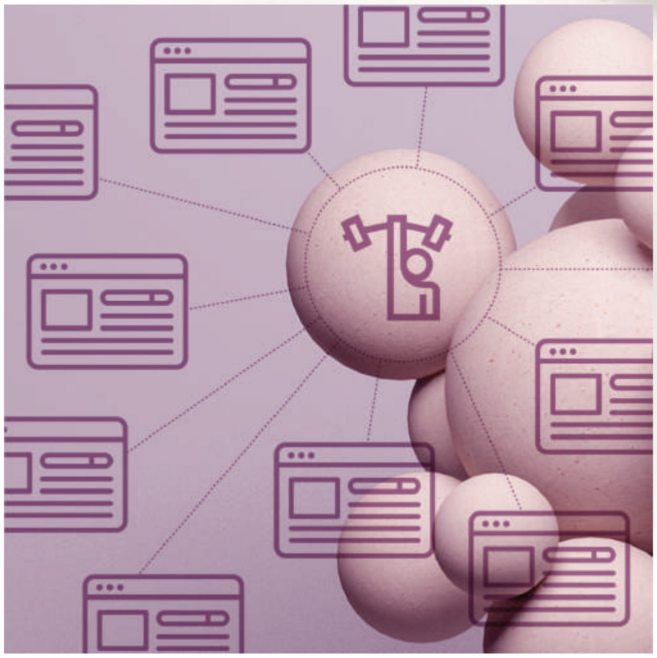
If you really want your costs of tooling to skyrocket (and to fail trying), set a goal that automation should find all defects.

SUBJECT7 QUALITY IS A CULTURE

AUTOMATION SHOULD FIND ALL POSSIBLE DEFECTS

By now, you certainly realize that what a computer does as it assists in testing is different from what a human does, and the aim is to blend the two.

If you realize this you are in the minority. Sadly, I have to admit, I expect many of the people who came to this page were searching for ways to automate 100% of the testing or to find all the defects -- that is why I kept the subtitles as claims, instead of putting "Myth:" in front.



Automation tools are unlikely to find a problem where the tab order is wrong. They are unlikely to find defects that involve the intersection of two features -- and they certainly won't find when two features, combined, create a logical third feature that should not exist. Those "third features" are often security bugs. One might allow a single user to spam everyone on a system, or allow a user to accidentally send emails to the entire company, and so on. Sometimes a sharp tester can identify these as unintended consequences of the requirements, sometimes during feature testing, but it is unlikely a tool will find them during an automation regression-test run, unsupervised by a human.

What the tools can do is provide some confidence that the largest, or more important features generally continue to work along the happy path - including the most common errors and boundaries. That combined, with a balanced breakfast that includes high-quality unit-test components, monitoring, and the ability to identify problems and roll them back quickly, could result in systems that can change quickly while having high resilience.

LESSON

Set expectations for what the testing can accomplish; don't look for testing alone to solve every engineering problem.

Believe it or not, I was talking about this over a decade ago at Google, along with Sean McMillan. And there's video!

WATCH IT ON YouTube

DEFLATING THE HYPE

We've said a lot in this paper, but to put a bow on it, here's a reminder about some of the other misconceptions we've covered in other writings.

OPEN SOURCE IS FREE

When I was in high school, a relative gifted me his old lawn mower when he bought a new one. I was delighted; I had a free mower that I'd use to make a bundle running a part time business. Trust me when I tell you, that mower wasn't free, especially when it came to the targeted outcome of perfectly groomed lawns in my neighborhood. When it worked, the easy part was keeping gas in the mower, pushing it around, and bagging the clippings. But that relied on a key premise which was not my reality, that I had a mower that was easy to use and worked reliably. My experience was everything other than that. Some of those same principals apply to open source tools. If you have unlimited time and patience to build around the rough edges, they can work, but they seldom lead to anything that can be described as quick or efficient. Refresh your reading of our Open Source Isn't Free blog to hear more on this.

TEST AUTOMATION WILL REPLACE MANUAL TESTING

If we had a nickel for every time we heard this, we'd probably have around $50 in nickels. So, let's debunk this one once and for all. Automation should never intend to eliminate manual testing. If you try, you will fail. What automation seeks, however, is to automate the mundane, repetitive tasks of testing, enabling your testers to have more time for activities that deliver higher value and quality, like exploratory testing. Remember what people and machines are good at. Machines, can quickly perform prescribed actions. Humans can apply creativity and know-how to find the breakage points in a product.

DEVELOPERS ARE BETTER FOR TEST-AUTOMATION THAN TESTERS

Pulling the thread on the previous point and disclaiming that in life, there are seldom absolute truths, what we've found in practice is that developers usually have a better mindset for creating things than breaking them. And testers usually aren't as technical as developers, so you would not want them building your code, but they are great at anticipating user-actions and finding the break-points. With the right codeless automation tool, testers can work independently from developers to identify defects and weaknesses, usually with more coverage and at a fraction of the cost of using developer-grade resources for testing.

Brining this article in for a landing… we expect many of the people who came to this were searching for ways to automate 100% of the testing or the strategies to find all the defects -- that is why we kept the subtitles as claims, instead of putting "Myth:" in front. We hope this has challenged your thinking, or at least given you counterpoints when leadership expectations of automated testing carry unrealistic connotations. Taking the time to explain how this works and keeping your team aligned to reality will enable the benefits of automation to shine through without letting the misconceptions erode confidence in your work.

That's a worthy goal and one that we know leads to success…