SUBJECT7
QUALITY IS A CULTURE

What does it Mean to Scale Test Automation?

By Matthew Heusser for Subject7 *Matt@xndev.com*

In computer science we have two kinds of scaling. In horizontal scaling, where we add more machines to a cluster to serve many clients at the same time. With vertical scaling we "beef up" the machines we have, adding more powerful CPU, more RAM, or faster hard drives. In Software Engineering, scaling can also mean coordinating a large project, across teams of teams and larger groups. All that raises the question - what does "scaling testing" even mean?

Today I'll address scaling test automation. To do that, I'll start at the word itself - very quickly - then move on to the important bit, what that means to you, the reader, right now.

A Moment on the Word Scaling

If you look up the word scaling in the dictionary, you'll find a reference to, well … a fish. Let's throw away that definition for now. In computer science, when we use the term "scaling," the image that comes to mind might be "scaling a cliff," that is, getting to the highest point. The logo of the Scaled Agile, Inc is a mountain, while Leading Agile uses a mountain as the entire metaphor for their transformation services, with basecamps, a compass, field notes, and so on.

The *root word* here is scale. That is the thing you use to find out how much something weighs. Or, perhaps the scope of things. "We need to consider this issue on a global scale," the President might say, because to do "the right thing" in a border city could lead to complications. If you've used scales, you know they have limits. When you add too much weight, the variables get too large, things break. This is another definition of scaling - when paint cracks off in small pieces.

To get test automation to scale, then, means to do a lot more of it. A lot more…*without things breaking down*, which is exactly what happens on a large project, or program of projects, if you're not paying attention.

Scaling Test Automation

Ironically, people often turn to automation because they believe software testing doesn't scale. The logic here is pretty simple: Every new sprint, the programmers add more features, but the testing team does not add more time to test. In the waterfall era, this was less of an issue, as bigger projects just meant a longer testing effort. When Scrum made two week sprints the standard, teams turned to test automation to reduce the testing time. Yet if you run tests on a laptop and the time to execute those tests grows by 15 minutes every two weeks, after two years, that could be a *thirteen-hour delay.* That amounts to 1.5 business days. I call this

automation delay, and that is a major source of failure of test tooling projects. It fits one definition of scaling, as the team is trying to cover more features or more software. Let's look at some other ones.

- ***More coverage (Deeper testing)***. It's easy enough to create a sort of "smoke test" that runs the bare minimum of scenarios. This shows it is possible to use the software as intended, and it is not broken in some horrendous way for the majority of users. Smoke tests cover something between build verification and the core scenarios. Of course, the software could still be horrendously broken for some minority of users, or modestly broken for all users. So management attempts to scale by adding deeper testing.

- ***Find problems faster.*** When I started at Socialtext, we ran our Selenium-based test suite on all supported browsers toward the end of each sprint. More than twelve years later, those tests all run *before* a human starts exploratory testing the feature. Other companies I work with run the tests immediately after the code is committed and before it is merged into the master branch. This isolates defects when they are introduced, to the exact change that created them.

- ***Cross team / More applications.*** This takes the approach of a successful automation effort for one team, or project, and applies it to many teams and projects. There may be re-use of components or code libraries involved, especially for a large website, like an eCommerce site, with multiple teams.

Notice that the first two goals conflict with each other. *The more testing you have, the longer it takes to run.* Yet one of the goals of test automation, in general, is earlier feedback. This creates automation delay. In more metaphorical terms, these are the cracks in the wall when you have too many layers of paint; the paint *scales off*.

Solving the Scaling Problem

The first step is to figure out exactly what problem your company is trying to solve. Do you want to have multiple teams, find problems faster, or have deeper coverage? How deep is deep enough?

In my experience, these are tough questions. Companies like to defer them. Even changing the conversation from testing "done or not" or "scale of one to ten" can be incredibly frustrating. As a consultant, one common reply I hear is "I want it all! That's why we hired you; if I wanted just one thing, my team could do that. I want to have my cake and eat it too."

Yes, I have literally heard executives use those words. Which is fine, as far as it goes.

For today, step one is defining the particular challenge you want to overcome. What problems do you want to solve first? One way to do figure this out is with a two-step survey. First, ask people the challenges to scaling -- these are the ways the wall is cracking. Then make a list,

give them a hundred points, and ask how many they would invest in each. That frames the problem of where to focus investment in solving a problem where it is not possible to do everything at once. If there are ten items, and the solution is ten points each, then expect slow, incremental improvements, and make a plan to address that. It is possible that the word "scale" is a red herring, a term thrown around as in "that won't scale" simply to stop the business.

Once you know what it means, it's time to consider solving it. The most common approaches to scaling test automation are crafted through the use of adaptive tools that can be easily adopted by a larger organizational audience. On top of that, companies layer continuous integration, self-service and self-loaded environments, and test coverage rubrics. After that, they may add running multiple tests in parallel. I'll cover a case study of these on my next blog entry. For now, let's isolate the problem you are trying to solve. Next, we'll build on it.