SUBJECT7
QUALITY IS A CULTURE

New Tech has made Test Re-Use a Reality

By Matthew Heuser for Subject7

You hear it at the conference, in the session with the performance consultant. It all sounds so helpful. She points out that you already have functional test automation. Those run through realistic scenarios, end to end. So put them on a grid, maybe in the cloud, then use them as the basis of your load and performance test platform.

It sounds great. It is easy to understand, even intuitive. It skips the painful process of figuring out what the traffic should be and, if you use Selenium Grid, how to drive it. Everyone recommends it; "reuse functional tests as load tests" has two hundred and twenty thousand search results on Google. Using the exact-phrase match returns three hundred and twenty results.

Except it never quite seems to work out.

The original sirens in Homer's Odyssey had beautiful voices and called to sailors to land on their island. The sirens were actually monsters leading ships to crash on rocky shores. In my experience, reusing GUI test tools for performance seems to have a similar effect. The promise is real, even admirable; the problem is getting tools that were born for different purposes to work together. Here are a few ideas about why things haven't worked out, and an outlook on how new tech can help keep your performance testing from crashing on the rocks.

Actually getting started

After the conference, you get back home and share the great idea with the team. Next you need to find someone to do the work. That person has to do one of two things. The first is to run a single test and capture the packets that run across the internet, and replay multiple versions of the packets with a load tool. The second is to try to keep the tests in the format you have and run a large number of instances of them. Essentially this is building your own tool to create load and capture the results. If you use Selenium, you are in some luck, as Selenium Grid at least exists. You will likely need to rent a cloud, paying by the CPU hour - and run a complete technical infrastructure, including browsers, not just a load generator. Either way, if you record the traffic or re-use with Selenium, you will still run into the first hurdle.

A test that works one time, repeated a hundred times simultaneously, is not the same as a hundred users. At the very least, you'll want the users to use different email addresses. You'll probably want to randomize the data. The setup and teardown will then be a little more complex. The tests will likely need to adapt to work with more generalized data – mainly to click on buttons and enter values that are reasonable, and not worrying as much about correctness of result. It is, after all, checking speed, not features. Creating the test environment, the performance test data, the load generator, the reporting, and customizing the test scripts will take a bit of work -- especially taking recorded traffic and creating variables. Eventually, this moves from someone's great idea to a project requiring coding expertise. That means either taking a programmer off production work (slowing down the team) or hiring a contractor. This brings us to our second challenge.

Enter the performance tester

The common strategy here is to bring in a performance tester from the outside. In my experience, there are two kinds of performance testers: the cheap ones, and those that can do the job. It turns out that the modeling, load generation, cloud coordination, raw TCP/IP networking, and reporting skills to put together a real performance program are not trivial. When my company has put bids on a real performance tester, the cost is not cheap. Every time we have lost a proposal, the company either subsequently let the winning (cheap or "fake" performance test contractor) go, or the tester quit, or occasionally, they get test results … that do not match the profile of the production system.

When I've put real performance testers on projects, they are likely to throw away whatever has come before and use their system of choice. Because they are expensive, the company has an incentive to get the work done and transition to an employee for maintenance. Here we have our third problem. It is unlikely the employee really understands what the performance test tool does. Instead, the tests just run as part of continuous integration. Over time, those tests become out of date. Web page addresses change. New required fields are added that make the old tests fail. If the functional tests are updated, the performance tests become out of date. Eventually someone notices, and throws the performance tests away, or starts the process again.

The idea here was noble, but in the end, it makes roughly as much sense as any other performance test approach. It also costs as much. Or more. The problem isn't the idea. The idea is good. It is the execution. Let's consider a different possibility.

What if the right tool existed...

Imagine that your tests are created in an adaptive, no-code testing tool. The software represents the tests in a human, visual, readable format. The tool lets you pick and choose whether you want to run the test flows for functional or performance testing. Better yet, the tool was designed to do both. You can create all the right variables to use for username, password, or changing data, then run the tests for one set of inputs. Or, take those same tests and substitute them for a larger set of variables under load.

When you change a functional test, you could now down-compile it to a performance test, and re-use it on the next test run. For that matter, you can set a flag and invoke security or accessibility tests, which could run on any web page the functional test touches.

This would, of course, run the risk of vendor lock-in. So does any approach. It does not quite offer the support and power of hiring one of the expert test consultants in the world, the type that fly around on jets in first class. The world will certainly need those types of experts. However, I am convinced this kind of re-use of functional tests for other kinds of testing may finally have arrived.

It is hard for me to adequately stress the importance of keeping the functional tests versioned with the performance tests. That one small tweak may be the difference between success and failure.

The first time I heard that pitch at a conference to re-use functional test assets was probably fifteen years ago. It's a great pitch, an important idea - and a challenging one. After fifteen years, Subject7 is the first tool I can endorse for their ability to reuse test assets.

It's about time.