SUBJECT7
QUALITY IS A CULTURE

Platform or Point Solution?

For Subject7 by Matt Heusser

A few years ago, it was popular for companies to create "suites" that were really just a bunch of small tools, purchased, and bundled together. Often the technology did not interoperate. Many of those companies are no longer in business, or else they have been purchased by a MegaCorp. None of that is what I mean by platform.

By platform I mean a single tool that the other tools integrate with. Programmers have something like this in Microsoft Visual Studio, or, to a lesser degree, something like Eclipse. These "studios" have their main purpose as code editors, they also do debugging, compiling, visual front-end design work, and even include the installer and dependency mappers. They tend to have a microkernel architecture that allows other people to write "plugins" to extend the product with new windows and new technologies. Unit testing tools like nUnit may be the best examples of plugins for Visual Studio. These platforms aim to be "the one ring" for software, as the one ring is in Tolkein's mythology: *One Ring to rule them all, One Ring to find them, One Ring to bring them all, and in the darkness bind them.*

Critics of the one ring say they are like standards. You have ten different standards and want one to control them all, so you write "the one standard." Congratulations, now you have eleven. That's certainly been true for testing, where my desktop has been so littered with tools for so long that no list I could produce would be complete. I have forgotten too many of them. At one point, I tried to put them on a USB drive. With the advent of Mobile, I got a new series of things like the network link conditioner, which intentionally slows down the bandwidth and corrupts packets to a cell phone to simulate poor-coverage conditions. I've long given up on the USB stick, though a website with a list of tools might make sense. Having a single platform is appealing, but does it really make any sense given the depth and breadth of our testing work?

Point Solutions

It's easy enough to understand the *appeal* of platforms. Point solutions force people into their own little, specialized role, a role that no one else understands. They often involve esoteric open source systems that take time and energy to master. This protects the specialist, but it completely denies the power of collaboration and slows the pace of overall development. And when the specialist is on vacation, sick, or, heaven help us, quits, there is a huge amount of relearning that needs to happen. In some cases, especially test automation, the team simply throws away everything that went before and starts over.

When things evolve naturally, you get a bunch of separate tools. The security people do their thing, the database people do theirs, there is a lot of confusion when the roles need to talk. Things work … sort of … until eventually they don't. Then there is the third approach, the "Swiss Army Chainsaw" that UNIX and Linux fans know so well.

Interoperability

These are a large number of little tools, strung together to create some great outcome. Many a UNIX hacker has combined a half-dozen commands into something to direct to a file, then used awk, sed, perl, or ruby to process that file and put it into a database. More than a few have used those sorts of commands to capture traffic and create low-level performance or even functional tests.

In my experience, when people build these sort-of "hacks in the best sense," it is possible that no one else on the team understands what they do - and it's *likely* that no one understands how. Plus, those sorts of things are useful to combine tools that process information the same way to produce a result. Load, Accessibility, API, and Database testing are very different modes of testing. It may be possible to re-use functional tests as load tests, and they may have a similar reporting engine, but under the hood, these are all very different animals.

That brings me back to the MicroKernel architecture, where we have a single "runner" that can run any combination of plugins. The plugins can have similar reporting and could push information to each other, yet be separate. That is the vision Subject7 has for the elements of testing that can be automated.

But will it blend?

As a consultant, one of my deepest frustrations is when I go into an organization and the role of testing is limited to just one of those plugins. Security, accessibility, performance, and load are all the domain of some other "expert" or contractor or just ignored completely. To borrow a phrase from Thomas More's Utopia: first we create a low-value-add role and then we disrespect and punish the people in it.

The Subject7 approach is to take just those components that can be automated and put them together under one platform. In Subject7, a "suite" consists of Test Scenarios. Each scenario is a set of commands. Commands can come in any type, from GUI/Functional tests, to database, API, or SSH to name a few. Consider this logical set of actions:

> *1] Setup a new user with no orders that belongs to a shared account with no orders*
> *2] Login as user #1*
> *3] See no orders*
> *4] Create another new user in that account and add an order*
> *5] See an order appear on user #1's screen*

Step #1 and #4 are not actually commands to the web browser. They are done on the back-end, adding rows to a database or calling a web service. What we are testing here is that the screen auto-refreshes. By creating database or API commands and adding them to a functional test, we've got peanut butter in our chocolate - and speeding up testing by hours. Consider the alternative: driving all the instructions through the user interface in one browser, or worse, trying to coordinate two different browser objects. The first isn't really a test of the real-time-update feature at all, while the second is going to lead to a lot of debugging and heartache.

There is power in the ability to have the tools work together. Another example might be security, load, or accessibility testing the page you touch by re-using functional tests created in Subject7 or tracking requirements to features to get some idea of coverage or traceability using a tool like Jira. All of this is possible in Subject7.

Moving Forward

More than a few tries have been made at creating a platform for software testing. Most of these are frankly naive, over-promising, and under-delivering. Today, I've laid out some of the problems with point solutions, and discussed how Subject7 has suggested a "one ring" -- while being very specific and pointed with claims on what it is capable of doing.

If you've got a better idea, I'm all ears. As James Bach once wrote, we are a young field. We should make the most of it. That means experimenting and trying. The folks at Subject7 have a MicroKernel architecture that enables different types of tests to interweave and different roles to communicate in a shared language.

What have you got?