SUBJECT7
QUALITY IS A CULTURE

Why Adaptive Testing is beating the West Coast School

By Matthew Heusser Matt@xndev.com

While the world of marketing is full of "best practices" and universal truisms, engineers tend to think in terms of *trade offs*. That is, this approach creates these problems and that approach creates those problems, and I value this over that, so I have made my choice. The Agile Manifesto may be one of the best examples of speaking in this language of tradeoffs, and it changed the way the world does software.

Today, I'd like to briefly introduce another one, the New Jersey School versus the MIT school of software design, as described in Richard Gabriel's Worse is Better, and go on to explain how that relates to the debate of our time - the Adaptive Tools Vs. The West Coast approach to software testing.

Worse is Better

A real debate about software design should include both sides, represented accurately. In Worse is Better, Richard Gabriel presented the MIT approach as "better," and his approach, the "New Jersey Approach," as "worse." The MIT approach is to build feature-complete software that can handle its own errors, while the New Jersey approach is to make good enough software that gets into the hands of the users quickly. Of course that is a vast oversimplification, please, go read the original essay. Today I will cut to Gabriel's conclusion:

> *The lesson to be learned from this is that it is often undesirable to go for the right thing first. It is better to get half of the right thing available so that it spreads like a virus. Once people are hooked on it, take the time to improve it to 90% of the right thing.*

Imagine if we could talk about testing this way. Not in terms of the "right" approach, but which one will be easier to adopt and, once adopted, easier to grow and improve.

The results of that might shock you.

West Coast Style Vs. Adaptive Tools

Getting testing closer to development has been a consistent trend I have seen, dating back to when the Manifesto was created in 2001. This makes sense. Development moved from desktop to web to mobile, and along with that came the ability to "push" new code to production at any time. With physical disks in boxes, it might make sense to have a "golden master" that was tested extensively. Today, new changes can push in minutes, and with the right infrastructure, be rolled back in minutes as well. That means a three-month test/release cycle just isn't

feasible. Today, I see testers embedded in the teams, taking the new features built by the programmers in the morning and testing in the afternoon. In some cases, the developer and tester work side-by-side on a project.

The extreme end of this is to eliminate the tester entirely, and simply have the programmer do the testing. In this world, testing is always "caught up" with programming, as the two happen within the same feedback loop. Programmers typically write test code inside the same version control system, so everything is versioned together. Features are done when they have running tested features, and testing includes automation. This school is emphasized in Silicon Valley, where development managers are typically programmers, and the organization typically exists in order to ship software. At Facebook, Google, Twitter, and Microsoft, the software is the product. In this world, testers look a lot like a cost center. Any executive that can eliminate a large department is likely to see quite the bonus. When Yahoo eliminated testing and QA five years ago, people took notice.

Then there are the companies that do not produce software for a living, that have customers that pay real money for physical objects, where the store is only one component of the business. Insurance companies, banks, brick and mortar retailers, manufacturers who need to ship inventory and put it on a shelf are a lot more cautious. Having Facebook, a product you do not pay for, down for a bit is considerably different than sending a truck with the wrong product to the wrong location. These companies see a difference between the builder job of programmer, and the breaker/evaluator job of testing. ***They typically see the tester as customer advocate and expert breaker***. This person is unlikely to be a programmer. If test artifacts exist, they would like them to be understandable by a wider group, not a more narrow one. This group sees code written that produces more code as a jumble of confusion, lacking any visualization of coverage or accountability. The automation tools that visualize the flow, that can be read by a "mere mortal" are appealing. These tools allow for a more classic tester/subject matter expert role to continue to exist, while speeding testing up. This group of people values their ability to do independent assessment and actually want to do testing, as opposed to programmers, who, well, don't.

The folks pursuing end-to-end, adaptive tools might be doing the "wrong thing" according to certain standards of computer science. But their outcomes are better. For them, as Gabriel said so well, "Worse is better."

There are other ways to think about testing. If all of your software consists of small, independent elements that can be tested and released separately, then you might not need automation at all.

The West Coast approach seems to be working … on the West Coast … for certain sets of technology companies. Will it work for you?

Tomorrow

This is just a tiny little essay, making a tiny little point, that the West Coast way, while it is the "right thing" on paper, is fraught with errors. It forces the people building the software to actually test it, while encouraging those outside to keep their noses out using complexity and code. Meanwhile, the tools we have been critiquing for decades, tools that model the testing and present it in a human-readable way, have been slowly getting better. These tools are "the wrong

thing" from a software engineering perspective - but it might just turn out that, in some contexts, worse is better.

You're free to disagree with me about their applicability in your organization. Perhaps you work for one of those "West Coast" companies that seem to take the approach and do just fine. All I am trying to do in this little essay is shift the conversation for "right" and "wrong" to tradeoffs, values, and environments. If you can explain why a style is a fit for you (or not), then you are doing better than most.

That said, I am going to give you some homework, if you would like it.

Describe the style of testing that is right for your organization, then defend it.

The dangerous position here may not be in Adaptive tooling or West coast. Either one can work in certain contexts. Instead, the danger is in not knowing where you stand, or why. Being able to explain the choices your organization makes, and why, is a good first step. That "why" should include the outcome you are seeking, and a willingness to change if the process isn't working. That introspection and self-awareness, and willingness to change might just be the table stakes to true success in software testing.