TDD is dead. Long live testing.

By David Heinemeier Hansson on April 23, 2014

Test-first fundamentalism is like abstinence-only sex ed: An unrealistic, ineffective morality campaign for self-loathing and shaming.

It didn't start out like that. When I first discovered TDD, it was like a courteous invitation to a better world of writing software. A mind hack to get you going with the practice of testing where no testing had happened before. It opened my eyes to the tranquility of a well-tested code base, and the bliss of confidence it grants those making changes to software.

The test-first part was a wonderful set of training wheels that taught me how to think about testing at a deeper level, but also some I quickly left behind.

Over the years, the test-first rhetoric got louder and angrier, though. More mean-spirited. And at times I got sucked into that fundamentalist vortex, feeling bad about not following the true gospel. Then I'd try test-first for a few weeks, only to drop it again when it started hurting my designs.

It was yoyo cycle of pride, when I was able to adhere to the literal letter of the teachings, and a crash of despair, when I wasn't. It felt like falling off the wagon. Something to keep quiet about. Certainly not something to admit in public. In public, I at best just alluded to not doing test-first all the time, and at worst continued to support the practice as "the right way". I regret that now.

Maybe it was necessary to use test-first as the counterintuitive ram for breaking down the industry's sorry lack of automated, regression testing. Maybe it was a parable that just wasn't intended to be a literal description of the day-to-day workings of software writing. But whatever it started out as, it was soon since corrupted. Used as a hammer to beat down the nonbelievers, declare them unprofessional and unfit for writing software. A litmus test.

Enough. No more. My name is David, and I do not write software test-first. I refuse to apologize for that any more, much less hide it. I'm grateful for what

TDD did to open my eyes to automated regression testing, but I've long since moved on from the design dogma.

I suggest you take a hard look at what that approach is doing to the integrity of your system design as well. If you're willing to honestly consider the possibility that it's not an unqualified good, it'll be like taking the red pill. You may not like what you see after that.

So where do we go from here?

Step one is admitting there's a problem. I think we've taken that now. Step two is to rebalance the testing spectrum from unit to system. The current fanatical TDD experience leads to a primary focus on the unit tests, because those are the tests capable of driving the code design (the original justification for test-first).

I don't think that's healthy. Test-first units leads to an overly complex web of intermediary objects and indirection in order to avoid doing anything that's "slow". Like hitting the database. Or file IO. Or going through the browser to test the whole system. It's given birth to some truly horrendous monstrosities of architecture. A dense jungle of service objects, command patterns, and worse.

I rarely unit test in the traditional sense of the word, where all dependencies are mocked out, and thousands of tests can close in seconds. It just hasn't been a useful way of dealing with the testing of Rails applications. I test active record models directly, letting them hit the database, and through the use of fixtures. Then layered on top is currently a set of controller tests, but I'd much rather replace those with even higher level system tests through Capybara or similar.

I think that's the direction we're heading. Less emphasis on unit tests, because we're no longer doing test-first as a design practice, and more emphasis on, yes, slow, system tests. (Which btw do not need to be so slow any more, thanks to advances in parallelization and cloud runner infrastructure).

Rails can help with this transition. Today we do nothing to encourage full system tests. There's no default answer in the stack. That's a mistake we're going to fix. But you don't have to wait until that's happening. Give Capybara a spin today, and you'll have a good idea of where we're heading

tomorrow.

But first of all take a deep breath. We're herding some sacred cows to the slaughter right now. That's painful and bloody. TDD has been so successful that it's interwoven in a lot of programmer identities. TDD is not just what they do, it's who they are. We have some serious deprogramming ahead of us as a community to get out from under that, and it's going to take some time.

The worst thing we can do is just rush into another testing religion. I can just imagine the golden calf of "system tests only!" right now. Please don't go there.

Yes, test-first is dead to me. But rather than dance on its grave, I'd rather honor its contributions than linger on the travesties. It marked an important phase in our history, yet it's time to move on.

Long live testing.

Continue down the rabbit hole with Why Most Unit Testing is Waste by James Coplien and my RailsConf keynote on Writing Software: Part 1 (starts at 11:00), Part 2.