TDD Outcomes

The Good & The Bad

KENT BECK
NOV 15, 2023

People keep saying, “TDD isn’t an X technique, it’s a Y technique,” where X & Y are chosen from {testing, design, development, collaboration}. I don’t understand this reductive impulse.

Here are some things you should be able to observe when using TDD:

- Fewer defects. Some mistakes you catch sooner, like in seconds instead of days.
- Fewer regressions. The test suite left behind after you test-drive code highlight unintended behavior changes.
- Better design, especially API design. The test you write gives you rapid feedback about the quality of your interface. (Note that you still have to make the design choices & the design will only be as good as you make it.)
- Improved collaboration. TDD gives you the opportunity to make your assumptions explicit. “What were they thinking? Oh, here’s a list.”
- Faster development. The efficiency stems not from “going faster” but rather wasting less time on surprises & over-engineering.
- Less anxiety. TDD is intended to help you transmuted fear into confidence. Worried about something? Write a test. Worried about implementing the whole thing? Make a list of tests, then satisfy them one at a time.

Note that TDD doesn’t do anything *for* you. Your design will only be as good as your design decisions. Your code will only be as reliable as your implementation decisions. Your communication will only be as good as the quality of your thoughts & your ability to express those thoughts.

But...

Here are some negative outcomes folks have reported after applying TDD. I'm not arguing that people *don't* experience these outcomes, I'm arguing that they aren't necessary.

- Slow tests. When I've had slow tests, I've been able to change the design to maintain the same confidence with more-granular tests.
- Structure-coupled tests. I saw this happen at Facebook, where a bunch of tests got written that embedded intimate knowledge of the design. At that point it was nigh-impossible to change the design. Too many tests would have to change, leaving you choosing between a) leaving the design "wrong", b) ripping out the tests, or c) changing the design through herculean effort. As with slow tests, a different design is likely testable without embedding design assumptions in the tests.
- Slower development. This is an interesting outcome. "TDD slows me down." Okay, but you "finished" the code, "froze" the code, then you had a bunch of bugs that interfered with ongoing development. If you added all the bug-related time spent, would it come to more or less?
- Difficulty with the sequence. "My brain just doesn't work that way." When I taught my eldest to program I used TDD. Then for the first 10 years of hir career ze didn't use it. (Now ze does.) I think reversing the order of test & implementation is a learnable skill, but I'm also open to the possibility that it isn't for everyone.
- Useless tests. When code coverage metrics become a substitute for actually caring about the reliability of code, then folks are incentivized to write tests without assertions, just to get over the bar. Tests that don't discriminate between well-behaved & badly-behaved code are costs without benefits.

The theme here is that many of TDD's negative outcomes are really "programming with poor design" negative outcomes. TDD can't solve that. People doing better design can (which is why software design's the topic of my recent book).

Conclusion

- Testing technique. You get fewer defects out compared to some other programming workflows.
- Design technique. You get fast feedback about the quality of your interface design.
- Programming technique. TDD results in running code.
- Collaboration technique. TDD gives you the opportunity to make some of your programming assumptions explicit.
- Emotional management technique. TDD can quell anxiety.

Why does it have to be just one of these? Or not to be one of these? Or exclude other techniques?

51 Likes · 2 Restacks

6 Comments

Write a comment...

Jay Bazuzi 3 hrs ago ❤ Liked by Kent Beck

TDD offers all these benefits and more. How we practice TDD affects which of the outcomes we get. Which outcomes we value and believe are offered affects how we choose to practice TDD.

One person, thinking that TDD is about delivering tests along with implementation because that protects against future regression, will be happy writing tests after. They will also see value in measuring code coverage and driving it up.

Another person, thinking that TDD is about design, will roll their eyes at code coverage: "my CC is usually high without trying, occasionally low and that's OK, and driving it higher won't help."

Another, who values TDD for the tiny pulse of satisfaction that comes with every new passing test, will focus on the tiniest red/green increments (possibly missing refactoring).

I want all of the benefits and none of the drawbacks. So that shapes how I strive to practice TDD.

LIKE (2) REPLY SHARE

Nilesh Thali 2 hrs ago ❤ Liked by Kent Beck

Thanks for broadening my perspective. I am one of the guilty ones who always points out that TDD is more a design paradigm than a testing paradigm.

On the teaching others TDD and not having it adopted, I had a team where most enthusiastically adopted except for one person. Then months later, that person said, "you know, i started using TDD, and i really love working that way now". I think they needed some time to themselves, which i agree. i frequently feel the pressure to pick something up quickly, instead of taking my time to understand and assimilate it.

Finally, TDD hasn't been "easy". it takes doing it and helping others do it (where have i heard that before?), failing, learning and improving. That's the "problem" with agile. I wanted someone to hand me the definitive best way to do everything, instead of this "empowered to learn by doing and improving" shizzle.

LIKE (1) REPLY SHARE

4 more comments...