Do We Really Understand Test Driven Development?

Although this technique is twenty-five years old, there are still a lot of misconceptions being circulated.

David Rodenas, PhD

07/20/2022

Share this on:

How TDD Is Seen?

TDD is a great technique, full of benefits. But this is not evident to everyone. **For some people, TDD is a practice that any professional should practice; for others, it is a waste of time**. It seems that there is a chasm between both profiles.

I believe that I got lucky; I have been on both sides. I started as a test hater. Well, not a hater, but I was unable to understand the value, so I saw it as useless. Later I had the opportunity to put it into practice with great mentors, and since then, my appreciation has not stopped growing.

I have met people that hate TDD or do not see the value. I have met people that have started TDD with great mentors, and they do not understand how not everyone does TDD.

Furthermore, I have met people that like to do TDD, but they practice it wrong, and although they lose a great part of the benefits, they still do it. And of course, I have met people "forced" to do TDD by its organization, and as Robert C. Martin warned once, they find imaginative ways to comply without delivering its value.

The Two Practices of TDD

The first time I started to understand TDD, and when I finally saw its value, others told me: you are not doing TDD. You are doing BDD instead. In those days, I did not know what BDD was, but for me, I was doing TDD as I had been taught. So, as you can understand, that remark puzzled me. I kept working the same way until I met another colleague, very self-confident, with a different TDD approach. Because we were on the same project, and both were sharing the same code, we needed to achieve an understanding. But, **instead of imposing a way to work from one to another, we chose to run an experiment**. During a few months, two types of tests and TDD were coexisting in the project. Inevitably, specs changed, the code needed refactors, and we discovered something interesting. My TDD tests resisted refactors, but his tests needed to be thrown away and rewritten almost every time.

What was the difference between both practices? The tests. While I was testing behaviors, he was testing the code. It seems strange; in both cases, you are executing the code, but the difference was the intention of the test. As you can suppose, by then, I did not say test behaviors, but I said that the test should be the documentation. While my tests were a guide about how to use the application, he tested functions and code. The difference was small, but the impact was huge.

Years later, I realized that most programmers think TDD should test the code, not the behavior. And that is a big mistake.

QA Unit Tests

When people test the code, they start asking the wrong questions: Do I have to test the public methods or also the private methods? Do I have to test all possible values of the arguments for every function? When you deeply understand the TDD, **the answer seems obvious: the code does not exist before the test**, so there are no public or private methods to test, and there are no arguments. But this is not obvious because we have been taught differently.

Looking for the root cause of why most programmers test the code but not the behavior, I came up with one clear answer: they are borrowing QA concepts. It turns out that QA was born inside the Waterfall methodology. In any Waterfall organization, once the developers have delivered the code, the testing team comes and tests all the code. They copy the model of an assembly line, in which thousands of copies of a product are tested by stressing the weak points. The Waterfall testing assumes that we already have the code and looks for cracks in the code. Thus, the QA testing methodology defined three levels of testing according to the type of code they are testing. This is how the testing pyramid emerged and how we all learned about Unit Tests, Integration Tests, and End-to-End tests.

In general, when new programmers learn to do TDD, they are taught to do Unit Tests as defined by the QA practice. QA Unit Tests are focused on pieces of code, not on behaviors. These tests mock everything, so they are useless for code design changes; and are very coupled to the code, so almost any change in the code implies a change in the test. As a result, the code becomes rigid, tests become an impediment, and programmers feel TDD is a waste of time.

TDD Unit Test

The other option is testing the behavior. **Instead of writing the test thinking in the code, we must think about the business rules**. It does not matter if we test all the public methods or all of the possible function argument values; what matters are all the relevant use case examples.

That changes how we manage the code. The tests become decoupled from the code, and refactors are possible without changing tests. That allows the design to evolve, and if there is any problem, you always know which business rule you broke.

Unfortunately, we also call these tests Unit Tests, but they are very different, with different objectives. They got this name because Kent Beck, the inventor of TDD, was the creator of the first xUnit testing tools.

Common Criticism

The first criticism people give me when I explain this is that we need to pinpoint bugs. The idea is simple, QA Unit Tests focus on isolated units, so they pinpoint exactly where the problem is, but TDD Unit Tests avoid mocks, so a failure can be at any part of the code.

Well, I have discovered that it is a fallacy. It turns out that if you have to change the test when you change the code, they are short-lived, and they very rarely have the opportunity to pinpoint the origin of a problem.

In contrast, the TDD Unit Tests are created inside the context of the TDD practice, which has a special meaning. When people practice TDD, they are supposed to do small steps. If at any moment, any test breaks, although the test does not pinpoint the origin, they know perfectly what broke it, the last edit. **If they cannot discover the origin quickly, they can undo all the changes and try again step by step**.

So, Do We Understand TDD?

I would say that, in general, no.

On one hand, because of the influence of Waterfall and classic test definitions, almost every programmer I know tends to use the wrong type of tests.

But on the other hand, although I have been practicing TDD for years, **I still find new benefits every few months**. For example, when I started, the big benefit was knowing I had not broken anything. Later, I discovered that I could change any design and improve iteratively any application, which was my main benefit. Later, because I discovered that tests were based on usage examples, I could go to business and testing, decide which usage examples were important, and then create these same test examples.

Currently, I have a list of almost a dozen benefits. Of course, I would not use TDD in all developments, but I have seen that misunderstandings have impeded its potential.

Is Testing Useless?

As you can see, I believe that we have too much research pending on software engineering. We need to create a stronger science.

I believe that it was the words of Edsger W. Dijkstra "Testing shows the presence, not the absence of bugs," that distanced testing from the academic world. Why bother testing if you can never test all the cases; thus, you always will have bugs.

It turns out that there is a twist to Dijkstra words.

What does TDD do? It makes tests fail first. It allows the test to work as expected – fail and detect bugs. Furthermore, and more importantly, to verify that it tests what is supposed to test.

About the Writer

Dr. David Rodenas is Ph.D. in Computer Architecture and MS in Computer Engineering. He is currently a freelance consultant, helping to create better software with better techniques, deeply focused on Agile, Lean, and Testing. He is also a teacher in the university; he teaches Software Engineering in the last courses of the Computer Engineering degree, and he also is an advisor in many final degree projects. Dr. Rodenas is a member of the Official Association of Professionals Computer Engineers of Catalonia, he was vice-dean from 2016 to 2020, and he has participated in several acts and talks in the Catalan community.

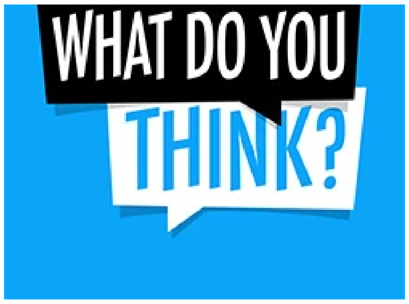
Do you have another perspective on the topic shared in this article? Let us know by filling out our guest blog form and share a few points you'd like to address. Complete the form here.

Recommended by IEEE Computer Society

Do We Really Understand Test Driven Development?

IEEE COMPUTER SOCIETY MEMBERSHIP CONFERENCES PUBLICATIONS

EDUCATION & CAREER VOLUNTEER ABOUT JOIN US



On the Weaponization of Open Source

Video Library

COMPUTING RESOURCES

Jobs Board

Courses & Certifications

Webinars

Podcasts

Tech News

Membership

COMMUNITY RESOURCES

Conference Organizers

Authors

Chapters

Communities

BUSINESS SOLUTIONS

Corporate Partnerships

Conference Sponsorships & Exhibits

Advertising

Recruiting

Digital Library Institution Subscriptions

POLICIES

Privacy

Accessibility Statement

IEEE Nondiscrimination Policy

XML Sitemap