



What are some common misunderstandings about TDD?

Asked 16 years, 3 months ago Modified 13 years, 8 months ago

Viewed 1k times


11


Reading over the responses to this question [Disadvantages of Test Driven Development?](#) I got the impression there is alot of misunderstanding on what TDD is and how it should be conducted. It may prove useful to address these issues here.



tdd



Share

Improve this question

Follow

edited May 23, 2017 at 12:08



Community Bot

1 ● 1

asked Sep 16, 2008 at 13:25



wioota


472 ● 4 ● 13

6 Answers

Sorted by:

Highest score (default)





I feel the accepted answer was one of the weakest ([Disadvantages of Test Driven Development?](#)), and the

13

most up-modded answer smells of someone who might be writing over specified tests.



Big time investment: for the simple case you lose about 20% of the actual implementation, but for complicated cases you lose much more.

TDD is an investment. I've found that once I was fully into TDD, the time I lost is very very little, and what time I did lose was more than made up when it came to maintenance time.

For complex cases your test cases are harder to calculate, I'd suggest in cases like that to try and use automatic reference code that will run in parallel in the debug version / test run, instead of the unit test of simplest cases.

If your test are becoming very complex, it might be time to review your design. TDD should lead you down the path smaller, less complex units of code working together

Sometimes you the design is not clear at the start and evolves as you go along - this will force you to redo your test which will generate a big time lose. I would suggest postponing unit tests in this case until you have some grasp of the design in mind.

This is the worst point of them all! TDD should really be "Test Driven **Design**". TDD is about design, not testing. To fully realise the value of benefits of TDD, you have to **drive** your design from your tests. So you should be *redoing* your production code to make your tests pass, not the other way round as this point suggests

Now the currently most upvoted: [Disadvantages of Test Driven Development?](#)

When you get to the point where you have a large number of tests, changing the system might require re-writing some or all of your tests, depending on which ones got invalidated by the changes. This could turn a relatively quick modification into a very time-consuming one.

Like the accepted answers first point, this seems like over specification in the tests and a general lack of understanding of the TDD process. When making changes, start from your test. Change the test for what the new code should do, and make the change. If that change breaks other tests, then your tests are doing what their supposed to do, failing. Unit Tests, for me, are designed to fail, hence why the RED stage is first, and should never be missed.

Share Improve this answer

edited May 23, 2017 at 12:33

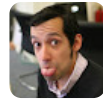
Follow



Community Bot

1 • 1

answered Sep 16, 2008 at 13:42



Chris Canal

4,865 ● 9 ● 37 ● 45



5



IMHO The biggest misconception about TDD is that: **time spent writing and refactoring tests would be time lost**. The thinking goes like "yeah, a test suite is nice, but the feature would be complete much faster if we just coded it".



When done properly, time spend writing and maintaining tests is saved multiple times over the life of the project in time *not* spent debugging and fixing regressions. Since the testing cost is up-front and the payoff is over time, it is easy to overlook.

Other big misconceptions include ignoring the impact of TDD on the design process, and not realizing that "painful tests" is a serious code smell that needs fixing quickly.

Share Improve this answer

answered Sep 16, 2008 at 13:48

Follow



ddaa

54.3k ● 8 ● 52 ● 59

I agree with your point on time spent vs. time lost. I've found that I write code faster when I can use a testing framework and TDD. It keeps me focused on the task, and tells me exactly where I make mistakes. I spend a lot less time head-scratching. – [Bill the Lizard](#) Sep 16, 2008 at 13:52



1



I see a lot of people misunderstanding what tests actually are usefull to TDD. People write big acceptance tests instead of small unit tests and then spend far too much time maintaining their tests and then conclude that TDD doesn't work. I think the BDD people have a point in avoiding the use of the word test entirely.

The other extreme is that people stop doing acceptance testing and think that because they do unit testing their code is tested. This is again a misunderstanding of the function of a unit test. You still need acceptance tests of some sort.

[Share](#) [Improve this answer](#)

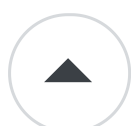
answered Sep 16, 2008 at 13:37

[Follow](#)



[Mendelt](#)

37.5k ● 6 ● 75 ● 97



0



The misconception that I often see is that TDD ensures good results.

Often times tests are written off of flawed requirements, and therefore, the developers produce a product that does not do what the user is expecting. Key to TDD is, in my opinion, working with the users to define requirements while helping manage their expectations.

[Share](#) [Improve this answer](#)

answered Sep 16, 2008 at 13:29

[Follow](#)



[Ian P](#)

13k ● 6 ● 50 ● 70

-
- 2 If you're working from flawed requirements, no development process can help you. This isn't a misconception about TDD.
– [Bill the Lizard](#) Sep 16, 2008 at 13:43
-



These are the issues that in my opinion are quite controversial and hence prone to misunderstanding:

0



- In my experience the biggest advantage is producing far better code at the cost of a lot of time spent writing tests. So it's really worthwhile for projects that require high quality, but on some other, less quality centric sites, the extra time will not be worth the effort.
- People seem to think that only a major subset of the features must be tested, but that is actually wrong IMHO. You need to test everything in order for your test to be valid after refactoring.
- The big drawback of TDD is the false sense of security given by incomplete tests: I've seen sites go down because people assumed that Unit Testing was enough to trigger a deployment.
- There is no need of mocking frameworks to do TDD. It's just a tool for testing some cases in an easier way. The best unit tests though are fired high in the stack and should be agnostic on the layers in the code. Testing one layer at a time is meaningless in this context.



0



Just chucking another answer in the pot.

One of the most common misunderstandings is that your **code is fixed**, ie. I have this code, now how on earth will I test it? If it's hard to write a test, we should ask the question: how can I change this code to make it easier to test?

Why..?

Well The sort of code that's easy to test is:

1. Modular - each method does one thing.
2. Parameterised - each method accepts everything it needs and outputs everything it should.
3. Well Specified - each method does exactly what it should, no more, no less.

If we write code like this, testing is a doddle. The interesting thing is that code that is easy to test is, coincidentally, **better code**.

Better as in easier to **read**, easier to **test**, easier to **understand**, easier to **debug**. This is why TDD is often described as a design exercise.

Follow



superluminary

49k ● 26 ● 153 ● 150
