Should unit tests be written before the code is written?

Asked 16 years, 1 month ago Modified 7 years, 1 month ago

Viewed 11k times

30

I know that one of the defining principles of Test driven development is that you write your Unit tests first and then write code to pass those unit tests, but is it necessary to do it this way?

I've found that I often don't know what I am testing until I've written it, mainly because the past couple of projects I've worked on have more evolved from a proof of concept rather than been designed.

I've tried to write my unit tests before and it can be useful, but it doesn't seem natural to me.

unit-testing tdd

Share

Improve this question

Follow

edited Oct 29, 2008 at 15:38

Sunny Milenov

22.3k ● 6 ● 82 ● 107

asked Oct 29, 2008 at 14:49

Omar Kooheji

55.6k ● 71 ● 186 ● 243

1 Please, make this community wiki, as it's more than poll than real question. – Sunny Milenov Oct 29, 2008 at 14:51

1 This ought to be a real question, rather than yet another poll. I'd prefer Omar reworded the question slightly to make it a proper question, rather than a poll-type question. – David Arno Oct 29, 2008 at 14:53

I'm happy to reword I'm not sure what to re-word it to though. – Omar Kooheji Oct 29, 2008 at 14:56

How about something along the lines of "When is the best time to write unit tests, before you start coding, or after you're done?" – Elie Oct 29, 2008 at 14:57

I'd be interested in knowing the methodology everyone uses to write their Unit tests. Do they try to gain a complete understanding of the problem before writing tests? Or do they write tests based on the current understanding, and change those tests later on if the the understanding changes? – George Stocker Oct 29, 2008 at 14:57

20 Answers

Sorted by: Highest score (default)

43

Some good comments here, but I think that one thing is getting ignored.

writing tests first drives your design. This is an important step. If you write the tests "at the same time" or "soon after" you might be missing some design benefits of doing TDD in micro steps.

It feels really cheesy at first, but it's amazing to watch things unfold before your eyes into a design that you

didn't think of originally. I've seen it happen.

TDD is hard, and it's not for everybody. But if you already embrace unit testing, then try it out for a month and see what it does to your design and productivity.

You spend less time in the debugger and more time thinking about outside-in design. Those are two gigantic pluses in my book.

Share Improve this answer

Follow

answered Oct 29, 2008 at 15:08

Ben Scheirman

40.9k ● 21 ● 103 ● 139

How do you create Unit Test first? The IDE will pop a lot of warning and error and the intellisence won't work! I do not see how to do it. – Pokus Oct 29, 2008 at 22:25

Let the IDE generate the method stub. All the test will fail at first. – flukus Oct 29, 2008 at 23:22

@Pokus - the quick answer is... I turn on auto-statement completion off so that intellisense doesn't get in my way. CTRL+space will always bring it up if I need it. I'd highly recommend ReSharper (jetbrains.com/resharper). It aids my TDD workflow heavily. – Ben Scheirman Oct 30, 2008 at 13:13

I think the idea is that you write an interfae first and implements it with stubs that (in C#) throw not implemented exceptions. All your tests will fail and writing your code will make the tests pass one by one. – Omar Kooheji Oct 30, 2008 at 15:16

@Ben - you can clear the list of characters that will commit from the autocomplete list; then only TAB will do it. Best of

both worlds. – Jay Bazuzi Dec 29, 2008 at 17:01

11

There have been studies that show that unit tests written after the code has been written are better tests. The caveat though is that people don't tend to write them after the event. So TDD is a good compromise as at least the tests get written.

So if you write tests after you have written code, good for you, I'd suggest you stick at it.

I tend to find that I do a mixture. The more I understand the requirements, the more tests I can write up front. When the requirements - or my understanding of the problem - are weak, I tend to write tests afterwards.

Share Improve this answer Follow

edited Oct 29, 2008 at 15:02

answered Oct 29, 2008 at 14:56

David Arno

43.2k ● 16 ● 88 ● 132

1 can you please provide links to the studies? Thanks – roundcrisis Oct 29, 2008 at 14:57

2 The link in this answer points to a blog post questioning the study results, not to actual studies with different results. I'd discount the validity of the first line of this answer as a result. – Scott Lawrence Oct 29, 2008 at 16:42

2 The link points to a detailed analysis of the conclusions the study's team drew and the reasons why their conclusions were wrong. The study actually clearly showed that test after is better than test before (just so long as it is carried out) – David Arno Oct 29, 2008 at 19:24

7

TDD is not about the tests, but how the tests drive your code. So basically you are writing tests to let an architecture evolve naturally (and don't forget to refactor !!! otherwise you won't get much benefit out of it). That you have an arsenal of regression tests and executable documentation afterwards is a nice sideeffect, but not the main reason behind TDD.

So my vote is: Test first

PS: And no, that doesn't mean that you don't have to plan your architecture before, but that you might rethink it if the tests tell you to do so !!!!

Share Improve this answer Follow

answered Oct 29, 2008 at 15:32

René

6

I've lead development teams for the past 6-7 years. What I can tell for sure is that as a developer and the developers I have worked with, it makes a phenomenal difference in the quality of the code if we know where our code fits into the big picture.

Test Driven Development (TDD) helps us answer "What?" before we answer "How?" and it makes a big difference.

I understand why there may be apprehensions about not following it in PoC type of development/architect work. And you are right it may not make a complete sense to follow this process. At the same time, I would like to emphasize that TDD is a process that falls in the Development Phase (I know it sounds obsolete, but you get the point :) when the low level specification are clear.

Share Improve this answer

Follow

answered Oct 29, 2008 at 15:48

a-sak

1,340 ● 6 ● 22 ● 33

3

I think writing the test first helps define what the code should actually do. Too many times people don't have a good definition of what the code is supposed to do or how it should work. They simply start writing and make it up as they go along. Creating the test first makes you focus on what the code will do.

Share Improve this answer

Follow

edited Nov 15, 2017 at 21:59

Andrew Brēza

8,267 ● 3 ● 36 ● 41

answered Oct 29, 2008 at 15:10

Jim C

4,981 ● 23 ● 25

Not always, but I find that it really does help when I do.

2

Share Improve this answer

Follow

answered Oct 29, 2008 at 14:50

Paul Croarkin

14.7k ● 16 ● 81 ● 119

2

I tend to write them as I write my code. At most I will write the tests for if the class/module exists before I write it.

I don't plan far enough ahead in that much detail to write a test earlier than the code it is going to test.

I don't know if this is a flaw in my thinking or method's or just TIMTOWTDI.

Share Improve this answer Follow

answered Oct 29, 2008 at 14:55

community wiki
J.J.

TIMTOWTD? what on earth is that? – Omar Kooheji Oct 29, 2008 at 14:58

1 There is more than one way to do it. Code + Tests about the same time can still be test-driven. Rough out the code, write good tests, finish the code. – S.Lott Oct 29, 2008 at 15:18

2

I start with how I would like to call my "unit" and make it compile. like:

```
picker = Pick.new
item=picker.pick('a')
assert item
```

then I create

```
class Pick
 def pick(something)
 return nil
 end
end
```

then I keep on using the Pick in my "test" case so I could see how I would like it to be called and how I would treat different kinds of behavior. Whenever I realize I could have trouble on some boundaries or some kind of error/exception I try to get it to fire and get an new test case.

So, in short. Yes. The ratio doing test before is a lot higher than not doing it.

Share Improve this answer

Follow

answered Oct 29, 2008 at 14:57

Jonke

6,533 ● 2 ● 27 ● 40

2

Writing the tests first defines how your code will look like - i.e. it tends to make your code more modular and testable, so you do not create a "bloat" methods with very complex and overlapping functionality. This also helps to isolate all core functionality in separate methods for easier testing.

Share Improve this answer

Follow

answered Oct 29, 2008 at 15:22

Sunny Milenov

22.3k ● 6 ● 82 ● 107

Directives are suggestion on how you could do things to improve the overall quality or productivity or even both of the end product. They are in no ways laws to be obeyed less you get smitten in a flash by the god of proper coding practice.

Here's my compromise on the take and I found it quite useful and productive.

Usually the hardest part to get right are the requirements and right behind it the usability of your class, API, package... Then is the actual implementation.

1. Write your interfaces (they will change, but will go a long way in knowing **WHAT** has to be done)
2. Write a simple program to use the interfaces (them stupid main). This goes a long way in determining the **HOW** it is going to be used (go back to 1 as often as needed)
3. Write tests on the interface (The bit I integrated from TDD, again go back to 1 as often as needed)
4. write the actual code behind the interfaces
5. write tests on the classes and the actual implementation, use a coverage tool to make sure you do not forget weid execution paths

So, yes I write tests before coding but never before I figured out what needs to be done with a certain level of details. These are usually high level tests and only treat the whole as a black box. Usually will remain as

integration tests and will not change much once the interfaces have stabilized.

Then I write a bunch of tests (unit tests) on the implementation behind it, these will be much more detailed and will change often as the implementation evolves, as it get's optimized and expanded.

Is this strictly speaking TDD ? Extreme ? Agile...? whatever... ? I don't know, and frankly I don't care. It works for **me**. I adjust it as needs go and as my understanding of software development practice evolve.

my 2 cent

Share Improve this answer

Follow

answered Oct 29, 2008 at 15:38

Newtopian

7,702 ● 4 ● 51 ● 73

2

I've been programming for 20 years, and I've virtually never written a line of code that I didn't run some kind of unit test on--Honestly I know people do it all the time, but how someone can ship a line of code that hasn't had some kind of test run on it is beyond me.

Often if there is no test framework in place I just write a main() into each class I write. It adds a little cruft to your app, but someone can always delete it (or comment it out) if they want I guess. I really wish there was just a test() method in your class that would automatically

compile out for release builds--I love my test method being in the same file as my code...

So I've done both Test Driven Development and Tested development. I can tell you that TDD can really help when you are a starting programmer. It helps you learn to view your code "From outside" which is one of the most important lessons a programmer can learn.

TDD also helps you get going when you are stuck. You can just write some very small piece that you know your code has to do, then run it and fix it--it gets addictive.

On the other hand, when you are adding to existing code and know pretty much exactly what you want, it's a toss-up. Your "Other code" often tests your new code in place. You still need to be sure you test each path, but you get a good coverage just by running the tests from the front-end (except for dynamic languages--for those you really should have unit tests for everything no matter what).

By the way, when I was on a fairly large Ruby/Rails project we had a very high % of test coverage. We refactored a major, central model class into two classes. It would have taken us two days, but with all the tests we had to refactor it ended up closer to two weeks. Tests are NOT completely free.

answered Oct 29, 2008 at 16:19

Bill K

62.8k ● 18 ● 112 ● 158

I'm not sure, but from your description I sense that there might be a misunderstanding on what test-first actually means. It does *not* mean that you write *all* your tests first. It does mean that you have a very tight cycle of

1. write a single, minimal test
2. make the test pass by writing the minimal production code necessary
3. write the next test that will fail
4. make all the existing tests pass by changing the existing production code in the simplest possible way
5. refactor the code (both test and production!) so that it doesn't contain duplication and is expressive
6. continue with 3. until you can't think of another sensible test

One cycle (3-5) typically just takes a couple of minutes. Using this technique, you actually *evolve* the design while you write your tests and production code in parallel. There is not much up front design involved at all.

On the question of it being "necessary" - no, it obviously isn't. There have been uncountable projects successfull without doing TDD. But there is some strong evidence out there that using TDD typically leads to significantly higher quality, often without negative impact on productivity. And it's fun, too!

Oh, and regarding it not feeling "natural", it's just a matter of what you are used to. I know people who are quite addicted to getting a green bar (the typical xUnit sign for "all tests passing") every couple of minutes.

Share Improve this answer

Follow

answered Nov 1, 2008 at 20:59

Ilja Preuß

2,421 ● 17 ● 15

2

There are so many answers now and they are all different. This perfectly resembles the reality out there. Everyone is doing it differently. I think there is a huge misunderstanding about unit testing. It seems to me as if people heard about TDD and they said it's good. Then they started to write unit tests without really understanding what TDD really is. They just got the part "oh yeah we have to write tests" and they agree with it. They also heard about this "you should write your tests first" but they do not take this serious.

I think it's because they do not understand the benefits of test-first which in turn you can only understand once you've done it this way for some time. And they always seem to find 1.000.000 excuses why they don't like writing the tests first. Because it's too difficult when figuring out how everything will fit together etc. etc. In my opinion, it's all excuses for them to hide away from their inability to once discipline themselve, try the test-first approach and start to see the benefits.

The most ridicoulous thing if they start to argue "I'm not conviced about this test-first thing but I've never done it this way" ... great ...

I wonder where unit testing originally comes from. Because if the concept really originates from TDD then it's just ridicoulous how people get it wrong.

Share Improve this answer Follow

answered Dec 1, 2010 at 21:06

user519499

1

Personally, I believe unit tests lose a lot of their effectiveness if not done before writing the code.

The age old problem with testing is that no matter how hard we think about it, we will never come up with every possibly scenario to write a test to cover.

Obviously unit testing itself doesn't prevent this completely, as it restrictive testing, looking at only one unit of code not covering the interactions between this code and everything else, but it provides a good basis for writing clean code in the first place that should at least restrict the chances for issues of interaction between modules. I've always worked to the principle of keeping code as simple as it possibly can be - infact I believe this is one of the key principles of TDD.

So starting off with a test that basically says you can create a class of this type and build it up, in theory, writing

a test for every line of code or at least covering every route through a particular piece of code. Designing as you go! Obviously based on a rough-up-front design produced initially, to give you a framework to work to.

As you say it is very unnatural to start with and can seem like a waste of time, but I've seen myself first hand that it pays off in the long run when defects stats come through and show the modules that were fully written using TDD have far lower defects over time than others.

Share Improve this answer

Follow

answered Oct 29, 2008 at 15:54

Adam Cobb

894 ● 4 ● 14 ● 33

1

Before, during and after. Before is part of the spec, the contract, the definition of the work During is when special cases, bad data, exceptions are uncovered while implementing. After is maintenance, evolution, change, new requirements.

Share Improve this answer

Follow

answered Oct 29, 2008 at 22:23

dongilmore

624 ● 3 ● 7

0

I don't write the actual unit tests first, but I do make a test matrix before I start coding listing all the possible scenarios that will have to be tested. I also make a list of cases that will have to be tested when a change is made to any part of the program as part of regression testing

that will cover most of the basic scenarios in the application in addition to fully testing the bit of code that changed.

Share Improve this answer

Follow

answered Oct 29, 2008 at 14:56

Elie

13.8k ● 25 ● 78 ● 128

0

Remember with Extreme programming your tests effectly are you documenation. So if you don't know what you're testing, then you don't know what you want your application is going to do?

You can start off with "Stories" which might be something like

"Users can Get list of Questions"

Then as you start writing code to solve the unit tests. To solve the above you'll need at least a User and question class. So then you can start thinking about the fields:

"User Class Has Name DOB Address TelNo Locked Fields"

etc. Hope it helps.

Crafty

Share Improve this answer

Follow

answered Oct 29, 2008 at 15:00

CraftyFella

7,570 ● 7 ● 49 ● 62

0

Yes, if you are using true TDD principles. Otherwise, as long as you're writing the unit-tests, you're doing better than most.

In my experience, it is usually easier to write the tests before the code, because by doing it that way you give yourself a simple debugging tool to use as you write the code.

Share Improve this answer

Follow

answered Oct 29, 2008 at 15:00

chills42

14.5k ● 4 ● 44 ● 78

0

I write them at the same time. I create the skeleton code for the new class and the test class, and then I write a test for some functionality (which then helps me to see how I want the new object to be called), and implement it in the code.

Usually, I don't end up with elegant code the first time around, it's normally quite hacky. But once all the tests are working, you can refactor away until you end up with something pretty neat, tidy and proveable to be rock solid.

Share Improve this answer

answered Oct 29, 2008 at 15:01

Follow

JamShady
2,924 ● 1 ● 22 ● 22

0

It helps when you are writing something that you are used writing to write first all the thing you would regularly check for and then write those features. More times then not those features are the most important for the piece of software you are writing. Now , on the other side there are not silver bullets and thing should never be followed to the letter. Developer judgment plays a big role in the decision of using test driven development versus test latter development.

Share Improve this answer

Follow

answered Oct 29, 2008 at 22:12

Nikola Stjelja
3,687 ● 9 ● 39 ● 47