Beginning TDD - Challenges? Solutions? Recommendations? [closed]

Asked 16 years, 4 months ago Modified 13 years, 3 months ago

Viewed 21k times

43

votes

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, visit the help center for guidance.

Closed 13 years ago.

Locked. This question and its answers are locked because the question is off-topic but has historical significance. It is not currently accepting new answers or interactions.

OK, I know there have already been questions about getting started with TDD.. However, I guess I kind of know the general concensus is to *just do it* , However, I seem to have the following problems getting my head into the game:

- When working with collections, do will still test for obvious add/remove/inserts successful, even when based on Generics etc where we kind of "know" its going to work?
- Some tests seem to take forever to implement.. Such as when working with string output, is there a "better" way to go about this sort of thing? (e.g. test the object model before parsing, break parsing down into small ops and test there) In my mind you should always test the "end result" but that can vary wildly and be tedious to set up.
- I don't have a testing framework to use (work wont pay for one) so I can "practice" more. Are there any good ones that are free for commercial use? (at the moment I am using good 'ol *Debug.Assert* :)
- Probably the biggest.. Sometimes I don't know what to expect *NOT* to happen.. I mean, you get your green light but I am always concerned that I may be missing a test.. Do you dig deeper to try and break the code, or leave it be and wait for it all fall over later (which will cost more)..

So basically what I am looking for here is not a " *just do it* " but more " *I did this, had problems with this, solved them by this* ".. The **personal** experience :)

unit-testing language-agnostic tdd

Share edited May 23, 2017 at 12:34

Community Bot
1 ● 1

asked Aug 24, 2008 at 10:43
Rob Cooper
28.9k ● 26 ● 105 ● 142

Comments disabled on deleted / locked posts / reviews

11 Answers

Sorted by: Highest score (default)

50 votes

First, it is alright and normal to feel frustrated when you first start trying to use TDD in your coding style. Just don't get discouraged and quit, you will need to give it some time. It is a major paradigm shift in how we think about solving a problem in code. I like to think of it like when we switched from procedural to object oriented programming.

Secondly, I feel that test driven development is first and foremost a design activity that is used to flesh out the design of a component by creating a test that first describes the API it is going to expose and how you are going to consume it's functionality. The test will help shape and mold the System Under Test until you have been able to encapsulate enough functionality to satisfy whatever tasks you happen to be working on.

Taking the above paragraph in mind, let's look at your questions:

1. If I am using a collection in my system under test, then I will setup an expectation to make sure that the code was called to insert the item and then assert the

count of the collection. I don't necessarily test the Add method on my internal list. I just make sure it was called when the method that adds the item is called. I do this by adding a mocking framework into the mix, with my testing framework.

2. Testing strings as output can be tedious. You cannot account for every outcome. You can only test what you expect based on the functionality of the system under test. You should always break your tests down to the smallest element that it is testing. Which means you will have a lot of tests, but tests that are small and fast and only test what they should, nothing else.
3. There are a lot of open source testing frameworks to choose from. I am not going to argue which is best. Just find one you like and start using it.
   - MbUnit
   - nUnit
   - xUnit
4. All you can do is setup your tests to account for what you want to happen. If a scenario comes up that introduces a bug in your functionality, at least you have a test around the functionality to add that scenario into the test and then change your functionality until the test passes. One way to find where we may have missed a test is to use code coverage.

I introduced you to the mocking term in the answer for question one. When you introduce mocking into your arsenal for TDD, it dramatically makes testing easier to abstract away the parts that are not part of the system under test. Here are some resources on the mocking frameworks out there are:

- Moq: Open Source
- RhinoMocks: Open Source
- TypeMock: Commercial Product
- NSubstitute: Open Source

One way to help in using TDD, besides reading about the process, is to watch people do it. I recommend in watching the screen casts by JP Boodhoo on DNRTV. Check these out:

- Jean Paul Boodhoo on Test Driven Development Part 1
- Jean Paul Boodhoo on Test Driven Development Part 2
- Jean Paul Boodhoo on Demystifying Design Patterns Part 1
- Jean Paul Boodhoo on Demystifying Design Patterns Part 2
- Jean Paul Boodhoo on Demystifying Design Patterns Part 3
- Jean Paul Boodhoo on Demystifying Design Patterns Part 4

- Jean Paul Boodhoo on Demystifying Design Patterns Part 5

OK, these will help you see how the terms I introduced are used. It will also introduce another tool called Resharper and how it can facilitate the TDD process. I couldn't recommend this tool enough when doing TDD. Seems like you are learning the process and you are just finding some of the problems that have already been solved with using other tools.

I think I would be doing an injustice to the community, if I didn't update this by adding Kent Beck's new series on Test Driven Development on Pragmatic Programmer.

Share

edited Aug 27, 2011 at 20:33

answered Aug 24, 2008 at 17:19

Dale Ragan
18.3k ● 3 ● 55 ● 71

1 Dale, bloody awesome answer mate! Very clear and concide, thanks for taking the time to write a really good answer. Clear winner! ;) – Rob Cooper Sep 22, 2008 at 8:00

Great answer - I couldn't get the video's from JPB's site to work (http download or silverlight player). Is there other place these might be hosted? – Rob Allen Nov 25, 2008 at 20:31

The videos are from dnrTV. You may need the camtasia video codec to view them after downloading. You can find it here: perseus.franklins.net/tscc.exe – Dale Ragan Dec 11, 2008 at 0:11

You may want to check out this BDD testing framework: nspec.org – Amir Apr 11, 2011 at 19:55

6 votes

From my own experience:

1. Only test your own code, not the underlying framework's code. So if you're using a generic list then there's no need to test Add, Remove etc.
2. There is no 2. Look over there! Monkeys!!!
3. NUnit is the way to go.
4. You definitely can't test every outcome. I test for what I expect to happen, and then test a few edge cases where I expect to get exceptions or invalid responses. If a bug comes up down the track because of something you forgot to test, the first thing you should do (before trying to fix the bug) is write a test to prove that the bug exists.

Share

answered Aug 24, 2008 at 10:49

Matt Hamilton
204k ● 61 ● 392 ● 321

1 #3 isn't very language-agnostic, and isn't very informative to people not using C#. – Andrew Grimm Jun 9, 2010 at 2:51

@Andrew NUnit is definitely language agnostic. You can code in C#, VB.NET, F#, Boo, Delphi, Cobol ... anything on the CLR. Perhaps you meant "platform agnostic"? Regardless, it's not limited to C#. – Matt Hamilton Jun 9, 2010 at 2:59

Does ruby (apart from IronRuby) use the CLR? – Andrew Grimm Jun 9, 2010 at 3:10

No, just IronRuby (which is explicitly a CLR version of Ruby). – Matt Hamilton Jun 9, 2010 at 3:16

2
votes

My take on this is following:

- +1 for not testing framework code, but you may still need to test classes derived from framework classes.
- If some class/method is cumbersome to test it may be strong indication that something is wrong with desing. I try to follow "1 class - 1 responsibility, 1 method - 1 action" principle. That way you will be able to test complex methods much easier by doing that in smaller portions.
- +1 for xUnit. For Java you may also consider TestNG.
- TDD is not single event it is a process. So do not try to envision everything from the beginning, but make sure that every bug found in code is actually covered by test once discovered.

Share

answered Aug 24, 2008 at 11:17

Dima Malenko
2,835 ● 2 ● 27 ● 24

2
votes

I think the most important thing with (and actually one of the great outcomes of, in a somewhat recursive manner) TDD is successful management of dependencies. You have to make sure that modules are tested in isolation with no elaborate setup needed. For example, if you're testing a component that eventually sends an email, make the email sender a dependency so that you can mock it in your tests. This leads to a second point - mocks are your friends. Get familiarized with mocking frameworks and the style of tests they promote (behavioral, as opposed to the classic state based), and the design choices they encourage (The "Tell, don't ask" principle).

Share

answered Aug 24, 2008 at 12:46
On Freund
4,436 ● 2 ● 24 ● 31

2
votes

I found that the principles illustrated in the Three Index Cards to Easily Remember the Essence of TDD is a good guide.

Anyway, to answer your questions

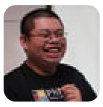
1. You don't have to test something you "know" is going to work, unless you wrote it. You didn't write generics, Microsoft did ;)
2. If you need to do so much for your test, maybe your object/method is doing too much as well.

3. Download TestDriven.NET to immediately start unit testing on your Visual Studio, (except if it's an Express edition)
4. Just test the *correct thing that will happen*. You don't *need* to test everything that can go wrong: you have to wait for your tests to fail for that.

Seriously, just do it, dude. :)

Share

answered Aug 24, 2008 at 13:14


Jon Limjap
95.3k ● 15 ● 103 ● 153

1 #1 supposed to be reassuring?? (I know - select isn't broken) – Andrew Grimm Jun 9, 2010 at 2:53

0 votes

I am no expert at TDD, by any means, but here is my view:

- If it is completely trivial (getters/setters etc) do not test it, unless you don't have confidence in the code for some reason.
- If it is a quite simple, but non-trivial method, test it. The test is probably easy to write anyway.
- When it comes to what to expect not to happen, I would say that if a certain potential problem is the responsibility of the class you are testing, you need to test that it handles it correctly. If it is not the current class' responsibility, don't test it.

The xUnit testing frameworks are often free to use, so if you are a .Net guy, check out NUnit, and if Java is your thing check out JUnit.

Share

answered Aug 24, 2008 at 10:55

Erik Öjebo

10.8k ● 4 ● 55 ● 76

0

votes

The above advice is good, and if you want a list of free frameworks you have to look no farther than the xUnit Frameworks List on Wikipedia. Hope this helps :)

Share

answered Aug 24, 2008 at 11:18

Eric Scrivner

1,849 ● 1 ● 19 ● 23

0

votes

In my opinion (your mileage may vary):

1- If you didn't write it don't test it. If you wrote it and you don't have a test for it it doesn't exist.

3- As everyone's said, xUnit's free and great.

2 & 4- Deciding exactly what to test is one of those things you can debate about with yourself forever. I try to draw this line using the principles of design by contract. Check out 'Object Oriented Software Construction" or "The Pragmatic Programmer" for details on it.

Share

answered Aug 24, 2008 at 12:49

Chris B-C
1,434 ● 14 ● 11

0
votes

Keep tests short, "atomic". Test the smallest assumption in each test. Make each TestMethod independent, for integration tests I even create a new database for each method. If you need to build some data for each test use an "Init" method. Use mocks to isolate the class your testing from it's dependencies.

I always think "what's the minimum amount of code I need to write to prove this works for all cases ?"

Share

answered Aug 24, 2008 at 18:33
Chris Porter
2,449 ● 2 ● 24 ● 26

0
votes

Over the last year I have become more and more convinced of the benefits of TDD. The things that I have learned along the way: 1) dependency injection is your friend. I'm not talking about inversion of control containers and frameworks to assemble plugin architectures, just passing dependencies into the constructor of the object under test. This pays back huge dividends in the testability of your code. 2) I set out with the passion / zealotry of the convert and grabbed a mocking framework and set about using mocks for everything I could. This led to brittle tests that required lots of painful set up and would fall over as soon as I started any refactoring. Use the correct kind of test double. Fakes where you just need to honour an interface, stubs to feed data back to the object under test, mock only where you care about interaction. 3) Test should be small. Aim for one assertion or interaction being tested in each test. I try to do this and mostly I'm there. This is about robustness of test code and also about the amount of complexity in a test when you need to revisit it later.

The biggest problem I have had with TDD has been working with a specification from a standards body and a third party implementation of that standard that was the de-facto standard. I coded lots of really nice unit tests to the letter of the specification only to find that the implementation on the other side of the fence saw the standard as more of an advisory document. They played quite loose with it. The only way to fix this was to test with the implementation as well as the unit tests and refactor the tests and code as necessary. The real problem was the belief on my part that as long as I had code and unit tests all was good. Not so. You need to be building actual outputs and performing functional testing at the same time as you are unit testing. Small pieces of benefit all the way through the process - into users or stakeholders hands.

Share

answered Sep 17, 2008 at 11:07

Hamish Smith
8,181 ● 1 ● 38 ● 49

0
votes

Just as an addition to this, I thought I would say I have put a blog post up on my thoughts on getting started with testing (following this discussion and my own research), since it may be useful to people viewing this thread.

"TDD – Getting Started with Test-Driven Development" - I have got some great feedback so far and would really appreciate any more that you guys have to offer.

I hope this helps! :)

Share

answered Sep 22, 2008 at 7:53

Rob Cooper

28.9k ● 26 ● 105 ● 142