Unit test adoption [closed]

Asked 15 years, 7 months ago Modified 9 years, 3 months ago

Viewed 10k times

69

Closed. This question is opinion-based. It is not currently accepting answers.

Want to improve this question? Update the question so it can be answered with facts and citations by editing this post.

Closed 7 years ago.

Improve this question

We have tried to introduce unit testing to our current project but it doesn't seem to be working. The extra code seems to have become a maintenance headache as when our internal Framework changes we have to go around and fix any unit tests that hang off it.

We have an abstract base class for unit testing our controllers that acts as a template calling into the child classes' abstract method implementations i.e. Framework calls Initialize so our controller classes all have their own Initialize method.

I used to be an advocate of unit testing but it doesn't seem to be working on our current project.

Can anyone help identify the problem and how we can make unit tests work for us rather than against us?

unit-testing tdd automated-tests agile

Share

Improve this question

Follow

edited Feb 5, 2013 at 19:33

cwash

4,245 ● 5 ● 46 ● 53

asked May 28, 2009 at 14:00

Burt

7,758 ● 20 ● 72 ● 130

Interesting question. Keep us informed please! – borjab May 28, 2009 at 14:13

19 Answers

Sorted by: Highest score (default)

109

Tips:

Avoid writing procedural code

Tests can be a bear to maintain if they're written against procedural-style code that relies heavily on global state or lies deep in the body of an ugly method. If you're writing

code in an OO language, use OO constructs effectively to reduce this.

- Avoid global state if at all possible.
- Avoid statics as they tend to ripple through your codebase and eventually cause things to be static that shouldn't be. They also bloat your test context (see below).
- Exploit polymorphism effectively to prevent excessive ifs and flags

Find what changes, encapsulate it and separate it from what stays the same.

There are choke points in code that change a lot more frequently than other pieces. Do this in your codebase and your tests will become more healthy.

- Good encapsulation leads to good, loosely coupled designs.
- Refactor and modularize.
- Keep tests small and focused.

The larger the context surrounding a test, the more difficult it will be to maintain.

Do whatever you can to shrink tests and the surrounding context in which they are executed.

- Use composed method refactoring to test smaller chunks of code.
- Are you using a newer testing framework like TestNG or JUnit4? They allow you to remove duplication in tests by providing you with more fine-grained hooks into the test lifecycle.
- Investigate using test doubles (mocks, fakes, stubs) to reduce the size of the test context.
- Investigate the Test Data Builder pattern.

Remove duplication from tests, but make sure they retain focus.

You probably won't be able to remove all duplication, but still try to remove it where it's causing pain. Make sure you don't remove so much duplication that someone can't come in and tell what the test does at a glance. (See Paul Wheaton's "Evil Unit Tests" article for an alternative explanation of the same concept.)

- No one will want to fix a test if they can't figure out what it's doing.
- Follow the Arrange, Act, Assert Pattern.
- Use only one assertion per test.

Test at the right level to what you're trying to verify.

Think about the complexity involved in a record-and-playback Selenium test and what could change under you versus testing a single method.

- Isolate dependencies from one another.
- Use dependency injection/inversion of control.
- Use test doubles to initialize an object for testing, and make sure you're testing single units of code in isolation.
- Make sure you're writing relevant tests
    - "Spring the Trap" by introducing a bug on purpose and make sure it gets caught by a test.
- See also: Integration Tests Are A Scam

Know when to use State Based vs Interaction Based Testing

True unit tests need true isolation. Unit tests don't hit a database or open sockets. Stop at mocking these interactions. Verify you talk to your collaborators correctly, not that the proper result from this method call was "42".

Demonstrate Test-Driving Code

It's up for debate whether or not a given team will take to test-driving all code, or writing "tests first" for every line of code. But should they write at least some tests first? Absolutely. There are scenarios in which test-first is undoubtedly the best way to approach a problem.

- Try this exercise: TDD as if you meant it (Another Description)
- See also: Test Driven Development and the Scientific Method

Resources:

- *Test Driven* by Lasse Koskela
- *Growing OO Software, Guided by Tests* by Steve Freeman and Nat Pryce
- *Working Effectively with Legacy Code* by Michael Feathers
- *Specification By Example* by Gojko Adzic
- Blogs to check out: Jay Fields, Andy Glover, Nat Pryce
- As mentioned in other answers already:
  - XUnit Patterns
  - Test Smells
  - Google Testing Blog
  - "OO Design for Testability" by Miskov Hevery
- "Evil Unit Tests" by Paul Wheaton
- "Integration Tests Are A Scam" by J.B. Rainsberger
- "The Economics of Software Design" by J.B. Rainsberger

- "Test Driven Development and the Scientific Method" by Rick Mugridge
- "TDD as if you Meant it" exercise originally by Keith Braithwaite, also workshopped by Gojko Adzic

Share Improve this answer Follow

edited Sep 12, 2015 at 14:59

community wiki
13 revs
cwash

1 why is this question not getting more upvotes? This is some good stuff! – Epaga Jun 2, 2009 at 7:18

1 This was a stub blog entry of mine... I'll probably reiterate the question and post it soon. – cwash Jun 2, 2009 at 15:32

2 Seriously, one assertion per test? How much time do you spend writing test method signatures? – erikkallen Sep 28, 2009 at 19:43

1 Also, why don't you like static methods? I often find it much easier to test if I have a bunch of static methods (of course, no side effects) and then have mutating instance methods be of the form "mem = f1(mem1, mem2); mem2 = f2(mem1);", and then just test the static method. – erikkallen Sep 28, 2009 at 19:45

1 @erikkallen - w/r/t static methods, static methods are a black box. i can't change anything out for the purposes of testing. they tend to propagate the use of global state and therefore make testing more difficult. if you write a static method that doesn't use global state, odds are all the state used in the method comes in via arguments. why not write that as an instance method? see:

googletesting.blogspot.com/2008/12/… again, this is not a law of the universe, but a good idea if you are concerned with testability or maintainable tests. – cwash Sep 28, 2009 at 21:36

19

Are you testing small enough units of code? You shouldn't see too many changes unless you are fundamentally changing everything in your core code.

Once things are stable, you will appreciate the unit tests more, but even now your tests are highlighting the extent to which changes to your framework are propogated through.

It is worth it, stick with it as best you can.

Share Improve this answer

Follow

answered May 28, 2009 at 14:03

cjk

46.4k ● 9 ● 82 ● 113

I agree with ck, for what I've seen of unit testing integration in a couple of company, unit testing frameworks are more often used for functional testing rather that unit testing. – Nicolas May 29, 2009 at 13:10

I wrote a blog about this topic a few months back: cwash.org/2009/02/17/dont-unit-test-anymore-no-really – cwash May 29, 2009 at 16:13

That is, how "developer testing" terminology has been confused with "unit testing" terminology and the implications of the confusion causes... – cwash May 29, 2009 at 16:15

Without more information it's hard to make a decent stab at why you're suffering these problems. Sometimes it's inevitable that changing interfaces etc. will break a lot of things, other times it's down to design problems.

It's a good idea to try and categorise the failures you're seeing. What sort of problems are you having? E.g. is it test maintenance (as in making them compile after refactoring!) due to API changes, or is it down to the behaviour of the API changing? If you can see a pattern, then you can try to change the design of the production code, or better insulate the tests from changing.

If changing a handful of things causes untold devastation to your test suite in many places, there are a few things you can do (most of these are just common unit testing tips):

- Develop small units of code and test small units of code. Extract interfaces or base classes where it makes sense so that units of code have 'seams' in them. The more dependencies you have to pull in (or worse, instantiate inside the class using 'new'), the more exposed to change your code will be. If each unit of code has a handful of dependencies (sometimes a couple or none at all) then it is better insulated from change.
- Only ever assert on what the test needs. Don't assert on intermediate, incidental or unrelated state. Design by contract and test by contract (e.g. if you're testing

a stack pop method, don't test the count property after pushing -- that should be in a separate test).

I see this problem quite a bit, especially if each test is a variant. If any of that incidental state changes, it breaks everything that asserts on it (whether the asserts are needed or not).

- Just as with normal code, use factories and builders in your unit tests. I learned that one when about 40 tests needed a constructor call updated after an API change...
- Just as importantly, use the front door first. Your tests should always use normal state if it's available. Only used interaction based testing when you have to (i.e. no state to verify against).

Anyway the gist of this is that I'd try to find out why/where the tests are breaking and go from there. Do your best to insulate yourself from change.

Share Improve this answer

Follow

answered May 28, 2009 at 14:25

Mark Simpson

23.4k ● 2 ● 46 ● 44

8

One of the benefits of unit testing is that when you make changes like this you can prove that you're not breaking your code. You do have to keep your tests in sync with your framework, but this rather mundane work is a lot easier than trying to figure out what broke when you refactored.

Share Improve this answer

Follow

answered May 28, 2009 at 14:03

Jon B

51.8k ● 31 ● 136 ● 163

4

I would insists you to stick with the TDD. Try to check your Unit Testing framework do one RCA (Root Cause Analysis) with your team and identify the area.

Fix the unit testing code at suite level and do not change your code frequently specially the function names or other modules.

Would appreciate if you can share your case study well, then we can dig out more at the problem area?

Share Improve this answer

Follow

answered May 28, 2009 at 14:12

Sourabh

1,605 ● 1 ● 16 ● 22

4

Good question!

Designing good unit tests is hard as designing the software itself. This is rarely acknowledged by developers, so the result is often hastily-written unit tests that require maintenance whenever the system under test changes. So, part of the solution to your problem could be spending more time to improve the design of your unit tests.

I can recommend one great book that deserves its billing as [The Design Patterns of Unit-Testing]

HTH

Share Improve this answer

Follow

answered May 28, 2009 at 14:38

azheglov

5,513 ● 1 ● 23 ● 30

definately a good book that likely addresses the issue the questioner has – Frank Schwieterman May 28, 2009 at 20:12

1 A lot of books help you get started with TDD and unit testing. The book above can help when you get stuck. – Pete TerMaat May 29, 2009 at 1:52

Indeed. If you get stuck, you just need to open it on the right page! – azheglov Jun 5, 2009 at 18:52

4

If the problem is that your tests are getting out of date with the actual code, you could do one or both of:

1. Train all developers to not pass code reviews that don't update unit tests.
2. Set up an automatic test box that runs the full set of units tests after every check-in and emails those who break the build. (We used to think that that was just for the "big boys" but we used an open source package on a dedicated box.)

Share Improve this answer

Follow

answered May 28, 2009 at 14:54

Robert Gowland
7,917 ● 6 ● 42 ● 58

3

Well if the logic has changed in the code, and you have written tests for those pieces of code, I would assume the tests would need to be changed to check the new logic. Unit tests are supposed to be fairly simple code that tests the logic of your code.

Share Improve this answer

Follow

answered May 28, 2009 at 14:04

CSharpAtl
7,502 ● 8 ● 41 ● 54

3

Your unit tests are doing what they are supposed to do. Bring to the surface any breaks in behavior due to changes in the framework, immediate code or other external sources. What this is supposed to do is help you determine if the behavior did change and the unit tests need to be modified accordingly, or if a bug was introduced thus causing the unit test to fail and needs to be corrected.

Don't give up, while its frustrating now, the benefit will be realized.

Share Improve this answer

Follow

answered May 28, 2009 at 14:30

David Yancey
2,030 ● 1 ● 16 ● 21

2

I'm not sure about the specific issues that make it difficult to maintain tests for your code, but I can share some of my own experiences when I had similar issues with my tests breaking. I ultimately learned that the lack of testability was largely due to some design issues with the class under test:

- Using concrete classes instead of interfaces
- Using singletons
- Calling lots of static methods for business logic and data access instead of interface methods

Because of this, I found that usually my tests were breaking - not because of a change in the class under test - but due to changes in other classes that the class under test was calling. In general, refactoring classes to ask for their data dependencies and testing with mock objects (EasyMock et al for Java) makes the testing much more focused and maintainable. I've really enjoyed some sites in particular on this topic:

- Google testing blog
- The guide to writing testable code

Share Improve this answer

Follow

answered May 28, 2009 at 14:58

Kyle Krull

1,708 ● 2 ● 18 ● 26

+1 The google stuff is really very good. googletesting.blogspot.com/2008/11/… is an excellent

presentation for those of us who do not like reading that much ;-) – jens Jun 3, 2009 at 6:27

2

Why should you have to change your unit tests every time you make changes to your framework? *Shouldn't this be the other way around?*

If you're using TDD, then you should first decide that your tests are testing the wrong behavior, and that they should instead verify that the desired behavior exists. Now that you've fixed your tests, your tests fail, and you have to go squish the bugs in your framework until your tests pass again.

Share Improve this answer

Follow

answered May 29, 2009 at 0:10

SingleNegationElimination

156k ● 35 ● 268 ● 306

1

Everything comes with price of course. At this early stage of development it's normal that a lot of unit tests have to be changed.

You might want to review some bits of your code to do more encapsulation, create less dependencies etc.

When you near production date, you'll be happy you have those tests, trust me :)

Share Improve this answer

answered May 28, 2009 at 14:09

Follow

Gerrie Schenck
22.4k ● 21 ● 69 ● 96

We are in the release cycle at the minute and this is the problem as we cannot afford to spend time fixing tests when we have Site Acceptance bugs to fix. – Burt May 28, 2009 at 14:31

2 @Burt This doesn't make any sense. You write unit test for behaviour that you don't *want* to change in the first place. If you want behaviour to change every other day, then don't write unit tests. What is a bit odd is to write them and then complain that they fail. You need unit tests if only if you want code with stable, repeatable behaviour. If you want your software to keep changing what it does, then don't unit test it. In this scenario, good luck with your manual testing, though. – Daniel Daranas May 28, 2009 at 14:58

How are we meant to evolve the framework without changing it? It is infrastructure code, I worked with the castle stack for a while working off the trunk code and it changed quite a lot so I presume it is common for a framework to improve by refactoring and changing when new requirements are identified. – Burt May 28, 2009 at 21:24

Either (A) You test it automatically, then when the behaviour changes you have to modify the test; (B) You test it manually, with a Word document or word-of-mouth expectations on the behaviour, then when you test it you have to tell the tester "Hey Joe, from now on when you do X and Y, Z must happen, instead of W", or (C) You don't test it. It looks like for some intermediate layer you are basculating between (A) and (C). Both have costs and advantages. – Daniel Daranas May 29, 2009 at 10:47

1

Aren't your unit tests too black-box oriented ? I mean ... let me take an example : suppose you are unit testing some sort of container, do you use the get() method of the container to verify a new item was actually stored, or do you manage to get an handle to the actual storage to retrieve the item directly where it is stored ? The later makes brittle tests : when you change the implementation, you're breaking the tests.

You should test against the interfaces, not the internal implementation.

And when you change the framework you'd better off trying to change the tests first, and then the framework.

Share Improve this answer

Follow

answered May 28, 2009 at 14:47

philant

35.7k ● 11 ● 73 ● 113

1

I would suggest investing into a test automation tool. If you are using continuous integration you can make it work in tandem. There are tools aout there which will scan your codebase and will generate tests for you. Then will run them. Downside of this approach is that its too generic. Because in many cases unit test's purpose is to break the system. I have written numerous tests and yes I have to change them if the codebase changes.

There is a fine line with automation tool you would definatelly have better code coverage.

However, with a well wrttien develper based tests you will test system integrity as well.

Hope this helps.

Share Improve this answer

Follow

answered May 28, 2009 at 14:58

Boris Kleynbok

327 ● 2 ● 5 ● 16

1

If your code is really hard to test and the test code breaks or requires much effort to keep in sync, then you have a bigger problem.

Consider using the extract-method refactoring to yank out small blocks of code that do one thing and only one thing; without dependencies and write your tests to those small methods.

Share Improve this answer

Follow

answered May 28, 2009 at 19:24

sal

23.6k ● 15 ● 69 ● 85

1

> The extra code seems to have become a maintenance headache as when our internal Framework changes we have to go around and fix any unit tests that hang off it.

The alternative is that when your Framework changes, you don't test the changes. Or you don't test the

Framework at all. Is that what you want?

You may try refactoring your Framework so that it is composed from smaller pieces that can be tested independently. Then when your Framework changes, you hope that either (a) fewer pieces change or (b) the changes are mostly in the ways in which the pieces are composed. Either way will get you better reuse of both code and tests. But real intellectual effort is involved; don't expect it to be easy.

Share Improve this answer

Follow

answered May 28, 2009 at 23:43

Norman Ramsey

202k ● 62 ● 371 ● 541

> Ideally our framework should have been interface driven meaning that we could mock a lot of it out but it was decided inheritance was a better option, it is the inheritance that was causing the issues and making functionality difficult to test.
> – Burt Aug 30, 2010 at 15:38

1

I found that unless you use IoC / DI methodology that encourages writing very small classes, and follow Single Responsibility Principle religiously, the unit-tests end up testing interaction of multiple classes which makes them very complex and therefore fragile.

My point is, many of the novel software development techniques only work when used together. Particularly MVC, ORM, IoC, unit-testing and Mocking. The DDD (in

the modern primitive sense) and TDD/BDD are more independent so you may use them or not.

Share Improve this answer

Follow

answered May 29, 2009 at 0:00

Andriy Volkov

18.9k ● 9 ● 69 ● 84

0

Sometime designing the TDD tests launch questioning on the design of the application itself. Check if your classes have been well designed, your methods are only performing one thing a the time ... With good design it should be simple to write code to test simple method and classes.

Share Improve this answer

Follow

answered May 28, 2009 at 21:13

sptremblay

595 ● 1 ● 5 ● 9

0

I have been thinking about this topic myself. I'm very sold on the value of unit tests, but not on strict TDD. It seems to me that, up to a certain point, you may be doing exploratory programming where the way you have things divided up into classes/interfaces is going to need to change. If you've invested a lot of time in unit tests for the old class structure, that's increased inertia against refactoring, painful to discard that additional code, etc.

Share Improve this answer

Follow

answered May 28, 2009 at 21:36

Anon

12.5k ● 3 ● 25 ● 19