Do setup/teardown hurt test maintainability?

Asked 15 years, 5 months ago Modified 15 years, 5 months ago

Viewed 6k times

18

This seemed to spark a bit of conversation on another question and I thought it worthy to spin into its own question.

The DRY principle seems to be our weapon-of-choice for fighting maintenance problems, but what about the maintenance of **test code**? Do the same rules of thumb apply?

A few strong voices in the developer testing community are of the opinion that setup and teardown are harmful and should be avoided... to name a few:

- James Newkirk
- Jay Fields, [2]

In fact, xUnit.net has removed them from the framework altogether for this very reason (though there are ways to get around this self-imposed limitation).

What has been your experience? Do setup/teardown hurt or help test maintainability?

UPDATE: do more fine-grained constructs like those available in JUnit4 or TestNG (@BeforeClass, @BeforeGroups, etc.) make a difference?

unit-testing dry fixtures factory-pattern maintainability

Share

Improve this question

Follow

edited May 23, 2017 at 10:29

Community Bot

1 • 1

asked Jul 6, 2009 at 14:35

cwash

4,245 • 5 • 46 • 53

Here's another set of answers in a similar vein: stackoverflow.com/questions/235025/… – Chris S Jul 6, 2009 at 16:36

@Chris - Thanks -- can you clarify your comment? Not sure I see a lot of similarity between the two other than questioning conventional wisdom for testing practices. If we are talking readability or maintainability of tests, testing more than one thing at a time will most definitely hurt that IMO – cwash Jul 6, 2009 at 17:03

@cwash specifically Jon Skeet's answer – Chris S Jul 7, 2009 at 8:23

@Chris I'm definitely a fan of one assertion per test - unit test methods are cheap. The problem as I see it is how to organize and name the tests, remove duplication, sure, but keep things focused and readable. – cwash Jul 7, 2009 at 17:18

9 Answers

Sorted by: Highest score (default)

11

The majority (if not all) of **valid** uses for setup and teardown methods can be written as factory methods which allows for DRY without getting into issues that seem to be plagued with the setup/teardown paradigm.

If you're implementing the teardown, typically this means you're not doing a unit test, but rather an integration test. A lot of people use this as a reason to not have a teardown, but IMO there should be both integration and unit test. I would personally separate them into separate assemblies, but I think a good testing framework should be able to support both types of testing. Not all *good* testing is going to be unit testing.

However, with the setup there seems to be a number of reasons why you need to do things before a test is actually run. For example, construction of object state to prep for the test (for instance setting up a Dependency Injection framework). This is a valid reason for a setup, but could just as easily be done with a factory.

Also, there is a distinction between class and method level setup/teardown. That needs to be kept in mind when considering what you're trying to do.

My biggest problem that I have had with using the setup/teardown paradigm is that my tests don't always follow the same pattern. This has brought me into using

factory patterns instead, which allows me to have DRY while at the same time being readable and not at all confusing to other developers. Going the factory route, I've been able to have my cake and eat it to.

Share Improve this answer Follow

edited Jul 6, 2009 at 14:47

answered Jul 6, 2009 at 14:41

Joseph

25.5k ● 8 ● 82 ● 127

Thanks.. Agree with your 2nd paragraph 100% - I've written a blog post about this before. Unit testing requires pure component isolation. Not all automated testing developers do is unit testing. Just because you're using a unit testing framework doesn't always mean you are writing a unit test. This is why I prefer to refer to it as developer testing.
– cwash Jul 6, 2009 at 14:46

@cwash I'm totally with you on that one. There needs to be a push to redefine this testing idea to "developer testing". I get a lot of push back when I use the phrase "unit testing", but if I call it "developer testing" I get a much wider acceptance.
– Joseph Jul 6, 2009 at 14:49

Also, agree with your assessment re: factory methods. But these have maintenance issues of their own. I'm a fan of test data builders for initializing test data. – cwash Jul 6, 2009 at 14:50

@cwash Yeah I usually have a completely separate project for building out my Mock and/or Stub frameworks, which handles all the data initialization and mock/sub behavior, but I didn't want to go that deep down the rabbit hole for this answer. – Joseph Jul 6, 2009 at 14:57

1 @Chris The concept of a "unit" is purposely ambiguous, because it will mean different things to different people. However, there should be a clean way to separate out functionality in your 'core' to be able to test each piece independently, including database operations. If you are having difficultly with this, then the code base most likely violates Single Responsibility Principle. – Joseph Jul 6, 2009 at 17:42

2

They've really helped with our test maintainability. Our "unit" tests are actually full end-to-end integration tests that write to the DB and check the results. Not my fault, they were like that when I got here, and I'm working to change things.

Anyway, if one test failed, it went on to the next one, trying to enter the same user from the first test in the DB, violating a uniqueness constraint, and the failures just cascaded from there. Moving the user creation/deletion into the [Fixture][SetUp|TearDown] methods allowed us to see the one test that failed without everything going haywire, and made my life a lot easier and less stabby.

Share Improve this answer

Follow

answered Jul 6, 2009 at 14:42

Chris Doggett

20.7k ● 4 ● 62 ● 87

I think they're a must have for integration tests. There is too much to do for these kinds of tests, and it's mostly repetitive. But as you mention they're not unit tests. For unit tests I think they do bloat the text context and make things harder to maintain and slower to run. – cwash Jul 6, 2009 at 14:48

2

I think the DRY principle applies just as much for tests as it does for code, however its application is different. In code you go to much greater lengths to literally not do the same thing in two different parts of the code. In tests the need to do that (do a lot of the same setup) is certainly a smell, but the solution is not necessarily to factor out the duplication into a setup method. It may be make the state easier to set up in the class itself or to isolate the code under test so it is less dependent on this amount of state to be meaningful.

Given the general goal of only testing one thing per test, it really isn't possible to avoid doing a lot of the same thing over and over again in certain cases (such as creating an object of a certain type). If you find you have a lot of that, it may be worth rethinking the test approach, such as introducing parametrized tests and the like.

I think setup and teardown should be primarily for establishing the environment (such as injections to make the environment a test one rather than a production one), and should not contain steps that are part and parcel of the test.

Share Improve this answer

Follow

answered Jul 6, 2009 at 14:43

Yishai

91.8k • 31 • 192 • 264

2

I agree with everything Joseph has to say, especially the part about tearDown being a sign of writing integration tests (and 99% of the time is what I've used it for), but in addition to that I'd say that the use of setup is a good indicator of when tests should be logically grouped together and when they should be split into multiple test classes.

I have no problem with large setup methods when applying tests to legacy code, but the setup should be common to *every test in the suite*. When you find yourself having the setup method really doing multiple bits of setup, then it's time to split your tests into multiple cases.

Following the examples in "Test Driven", the setup method comes about *from* removing duplication in the test cases.

Share Improve this answer

Follow

answered Jul 6, 2009 at 15:03

tddmonkey

21.2k • 10 • 59 • 69

TD is a great reference. We classically think of su/td as ways to remove duplication, but the question is should we always aggressively remove duplication from our test code?
– cwash Jul 6, 2009 at 15:13

I perhaps won't be as "aggressive" as I am with production code, but I do want all common functionality in a single place (setup) so that each test case can just show how it differs from the happy-day scenario rather than having alot of setup code – tddmonkey Jul 7, 2009 at 8:29

1

I use setup quite frequently in Java and Python, frequently to set up collaborators (either real or test, depending). If the object under test has no constructors or just the collaborators as constructors I will create the object. For a simple value class I usually don't bother with them.

I use teardown very infrequently in Java. In Python it was used more often because I was more likely to change global state (in particular, monkey patching modules to get users of those modules under test). In that case I want a teardown that will guaranteed to be called if a test failed.

Integration tests and functional tests (which often use the xunit framework) are more likely to need setup and teardown.

The point to remember is to think about fixtures, not only DRY.

Share Improve this answer

Follow

answered Jul 6, 2009 at 15:06

Kathy Van Stone

26.3k ● 3 ● 33 ● 42

1

I don't have an issue with test setup and teardown methods per se.

The issue to me is that if you have a test setup and teardown method, it implies that the same test object is being reused for each test. This is a potential error vector, as if you forget to clean up some element of state between tests, your test results can become order-dependent. What we really want is tests that do not share any state.

xUnit.Net gets rid of setup/teardown, because it creates a new object for each test that is run. In essence, the constructor becomes the setup method, and the finalizer becomes the teardown method. There's no (object-level) state held between tests, eliminating this potential error vector.

Most tests that I write have some amount of setup, even if it's just creating the mocks I need and wiring the object being tested up to the mocks. What they don't do is share any state between tests. Teardown is just making sure that I don't share that state.

Share Improve this answer

Follow

answered Jul 7, 2009 at 3:33

kyoryu

13k ● 2 ● 30 ● 33

I haven't had time to read both of what you posted, but I in particular liked this comment:

0

> each test is forced to do the initialization for what it needs to run.

Setup and tear down are convenience methods - they shouldn't attempt to do much more than initialize a class using its default constructor, etc. Common code that three tests need in a five test class shouldn't appear there - each of the three tests should call this code directly. This also keeps tests from stepping on each other's toes and breaking a bunch of tests just because you changed a common initalization routine. The main problem is that this will be called before all tests - not just specific tests. Most tests should be simple, and the more complex ones will need initialization code, but it is easier to see the simplicity of the simple tests when you don't have to trace through a complex initialization in set up and complex destruction in tear down while thinking about what the test is actually supposed to accomplish.

Share Improve this answer

Follow

answered Jul 6, 2009 at 14:42

MetroidFan2002

29.9k ● 16 ● 65 ● 81

1 Personally, if I have three tests that need the same SetUp in a group of 5, I would extract them into a separate fixture as they are obviously testing something not related to the other two tests. – Sekhat Jul 6, 2009 at 14:50

Well, I was assuming unit tests here - tests for only one class. Some methods may not require state that the whole class requires for operation, so its best not to assume in your

setup that all that stuff needs to be injected.
– MetroidFan2002 Jul 6, 2009 at 16:44

0

Personally, I've found setup and teardown aren't always evil, and that this line of reasoning is a bit dogmatic. But I have no problem calling them a code smell for unit tests. I feel their use should be justified, for a few reasons:

1. Test code is procedural by its nature. In general, setup/teardown **do** tend to reduce test readability/focus.
2. Setup methods tend to initialize more than what is needed for any single test. When abused they can become unwieldy. Object Mothers, Test Data Builders, perhaps frameworks like FactoryGirl seem better at initializing test data.
3. They encourage "context bloat" - the larger the test context becomes, the less maintainable it will be.

To the extent that my setup/teardown doesn't do this, I think their use is warranted. There will always be some duplication in tests. Neal Ford states this as *"Tests can be wet but not soaking..."* Also, I think their use is more justified when we're not talking about unit tests specifically, but integration tests more broadly.

Working on my own, this has never really been a problem. But I've found it very difficult to maintain test suites in a team setting, and it tends to be because we don't understand each other's code immediately, or don't

want to have to step through it to understand it. From a test perspective, I've found allowing some duplication in tests eases this burden.

I'd love to hear how others feel about this, though.

Share Improve this answer Follow

edited Jul 6, 2009 at 15:32

answered Jul 6, 2009 at 14:43

cwash

4,245 ● 5 ● 46 ● 53

0

If you need setup and teardown to make your unit tests work, maybe what you *really* need is mock objects?

Share Improve this answer Follow

answered Jul 7, 2009 at 3:43

SingleNegationElimination

156k ● 35 ● 268 ● 306

Not sure I follow. There is duplication involved in mocking as well? – cwash Jul 7, 2009 at 17:40

Also, the question is not whether su/td is needed to make the tests work. The question is whether they make the test code less maintainable. – cwash Jul 9, 2009 at 15:27