What is unit testing? [closed]

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

Viewed 56k times

230

Closed. This question needs to be more focused. It is not currently accepting answers.

Want to improve this question? Update the question so it focuses on one problem only by editing this post.

Closed 5 years ago.

The community reviewed whether to reopen this question last year and left it closed:

> Original close reason(s) were not resolved

Improve this question

I saw many questions asking 'how' to unit test in a specific language, but no question asking 'what', 'why', and 'when'.

- What is it?
- What does it do for me?
- Why should I use it?
- When should I use it (also when not)?
- What are some common pitfalls and misconceptions

unit-testing

Share

Improve this question

Follow

edited Sep 12, 2020 at 2:04

Wicket
37.9k ● 9 ● 77 ● 188

asked Aug 4, 2008 at 16:27

Uberfuzzy
8,362 ● 11 ● 43 ● 50

Uber, there are a lot of resources available, and the Wikipedia article is certainly a great place to start. Once you've got a basic idea, perhaps some more specific questions can help you decide for yourself if you want to use unit testing. – palehorse Aug 4, 2008 at 16:34

Clean code talks: youtube.com/watch?v=wEhu57pih5w – Kieran Nov 24, 2010 at 0:16

1 That is not 1 specific question. That is 5 broad questions. – Raedwald Nov 30, 2013 at 12:37

20 Answers

Sorted by: Highest score (default)

220

Unit testing is, roughly speaking, testing bits of your code in isolation with test code. The immediate advantages that come to mind are:

- Running the tests becomes automate-able and repeatable

- You can test at a much more granular level than point-and-click testing via a GUI

Note that if your test code writes to a file, opens a database connection or does something over the network, it's more appropriately categorized as an integration test. Integration tests are a good thing, but should not be confused with unit tests. Unit test code should be short, sweet and quick to execute.

Another way to look at unit testing is that you write the tests first. This is known as Test-Driven Development (TDD for short). TDD brings additional advantages:

- You don't write speculative "I might need this in the future" code -- just enough to make the tests pass
- The code you've written is always covered by tests
- By writing the test first, you're forced into thinking about how you want to call the code, which usually improves the design of the code in the long run.

If you're not doing unit testing now, I recommend you get started on it. Get a good book, practically any xUnit-book will do because the concepts are very much transferable between them.

Sometimes writing unit tests can be painful. When it gets that way, try to find someone to help you, and resist the temptation to "just write the damn code". Unit testing is a lot like washing the dishes. It's not always pleasant, but it keeps your metaphorical kitchen clean, and you really want it to be clean. :)

Edit: One misconception comes to mind, although I'm not sure if it's so common. I've heard a project manager say that unit tests made the team write all the code twice. If it looks and feels that way, well, you're doing it wrong. Not only does writing the tests usually speed up development, but it also gives you a convenient "now I'm done" indicator that you wouldn't have otherwise.

Share Improve this answer Follow

edited Mar 15, 2015 at 6:35

Shashank Sawant
1,140 ● 5 ● 23 ● 41

answered Aug 4, 2008 at 16:36

Rytmis
32k ● 8 ● 61 ● 69

---

4 Would you expand on your point about how TDD speeds up development? – xyhhx Jul 29, 2016 at 20:20

Test-Driven Development is just invented to make sure that the unit tests are actually written :) – yılmaz Nov 16, 2021 at 9:16

73

I don't disagree with Dan (although a better choice may just be not to answer)...but...

Unit testing is the process of writing code to test the behavior and functionality of your system.

Obviously tests improve the quality of your code, but that's just a superficial benefit of unit testing. The real benefits are to:

1. Make it easier to change the technical implementation while making sure you don't change the behavior (refactoring). Properly unit tested code can be aggressively refactored/cleaned up with little chance of breaking anything without noticing it.
2. Give developers confidence when adding behavior or making fixes.
3. Document your code
4. Indicate areas of your code that are tightly coupled. It's hard to unit test code that's tightly coupled
5. Provide a means to use your API and look for difficulties early on
6. Indicates methods and classes that aren't very cohesive

You should unit test because its in your interest to deliver a maintainable and quality product to your client.

I'd suggest you use it for any system, or part of a system, which models real-world behavior. In other words, it's particularly well suited for enterprise development. I would not use it for throw-away/utility programs. I would not use it for parts of a system that are problematic to test (UI is a common example, but that isn't always the case)

The greatest pitfall is that developers test too large a unit, or they consider a method a unit. This is particularly true if you don't understand [Inversion of Control]() - in which case your unit tests will always turn into end-to-end integration testing. Unit test should test individual behaviors - and most methods have many behaviors.

The greatest misconception is that programmers shouldn't test. Only bad or lazy programmers believe that. Should the guy building your roof not test it? Should the doctor replacing a heart valve not test the new valve? Only a programmer can test that his code does what he intended it to do (QA can test edge cases - how code behaves when it's told to do things the programmer didn't intend, and the client can do acceptance test - does the code do what what the client paid for it to do)

Share Improve this answer Follow

edited Oct 29, 2016 at 15:51
leif.gruenwoldt
13.9k ● 5 ● 62 ● 66

answered Aug 4, 2008 at 16:41
Karl Seguin
21.8k ● 5 ● 45 ● 49

47

The main difference of unit testing, as opposed to "just opening a new project and test this specific code" is that it's *automated*, thus *repeatable*.

If you test your code manually, it may convince you that the code is working perfectly - *in its current state*. But what about a week later, when you made a slight modification in it? Are you willing to retest it again by hand whenever *anything* changes in your code? Most probably not :-(

But if you can **run your tests anytime, with a single click, exactly the same way, within a few seconds**, then they *will* show you immediately whenever something is broken. And if you also integrate the unit tests into your automated build process, they will alert you to bugs even in cases where a seemingly completely unrelated change broke something in a distant part of the codebase - when it would not even occur to you that there is a need to retest that particular functionality.

This is the main advantage of unit tests over hand testing. But wait, there is more:

- unit tests **shorten the development feedback loop** dramatically: with a separate testing department it may take weeks for you to know that there is a bug in your code, by which time you have already forgotten much of the context, thus it may take you hours to find and fix the bug; OTOH with unit tests, the feedback cycle is measured in seconds, and the bug fix process is typically along the lines of an "oh sh*t, I forgot to check for that condition here" :-)

- unit tests effectively **document** (your understanding of) the behaviour of your code
- unit testing forces you to reevaluate your design choices, which results in **simpler, cleaner design**

Unit testing frameworks, in turn, make it easy for you to write and run your tests.

Share Improve this answer Follow

edited Mar 14, 2010 at 17:41

answered Mar 14, 2010 at 17:20

Péter Török
116k ● 31 ● 276 ● 331

+1 In addition, my favourite part about test code (especially when given a new codebase): It demonstrates the expected use of the code under test. – Steven Evers Mar 14, 2010 at 18:15

39

I was never taught unit testing at university, and it took me a while to "get" it. I read about it, went "ah, right, automated testing, that could be cool I guess", and then I forgot about it.

It took quite a bit longer before I really figured out the point: Let's say you're working on a large system and you write a small module. It compiles, you put it through its paces, it works great, you move on to the next task. Nine months down the line and two versions later someone else makes a change to some *seemingly* unrelated part of the program, and it breaks the module. Worse, they test their changes, and their code works, but they don't test your module; hell, they may not even know your module *exists*.

And now you've got a problem: broken code is in the trunk and nobody even knows. The best case is an internal tester finds it before you ship, but fixing code that late in the game is expensive. And if no internal tester finds it...well, that can get very expensive indeed.

The solution is unit tests. They'll catch problems when you write code - which is fine - but you could have done that by hand. The real payoff is that they'll catch problems nine months down the line when you're now working on a completely different project, but a summer intern thinks it'll look tidier if those parameters were in alphabetical order - and then the unit test you wrote way back fails, and someone throws things at the intern until he changes the parameter order back. **That's** the "why" of unit tests. :-)

Share Improve this answer Follow

edited Oct 22, 2011 at 16:34

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Sep 19, 2008 at 12:45

Cody Hatch
8,927 ● 6 ● 31 ● 36

14

Chipping in on the philosophical pros of unit testing and TDD here are a few of they key "lightbulb" observations which struck me on my tentative first steps on the road to TDD enlightenment (none original or necessarily news)...

1. TDD does NOT mean writing twice the amount of code. Test code is typically fairly quick and painless to write and is a key part of your design process and critically.
2. TDD helps you to realize when to stop coding! Your tests give you confidence that you've done enough for now and can stop tweaking and move on to the next thing.
3. The tests and the code work together to achieve better code. Your code could be bad / buggy. Your TEST could be bad / buggy. In TDD you are banking on the chances of BOTH being bad / buggy being fairly low. Often its the test that needs fixing but that's still a good outcome.
4. TDD helps with coding constipation. You know that feeling that you have so much to do you barely know where to start? It's Friday afternoon, if you just procrastinate for a couple more hours... TDD allows you to flesh out very quickly what you think you need to do, and gets your coding moving quickly. Also, like lab rats, I think we all respond to that big green light and work harder to see it again!

5. In a similar vein, these designer types can SEE what they're working on. They can wander off for a juice / cigarette / iphone break and return to a monitor that immediately gives them a visual cue as to where they got to. TDD gives us something similar. It's easier to see where we got to when life intervenes...
6. I think it was Fowler who said: "Imperfect tests, run frequently, are much better than perfect tests that are never written at all". I interprete this as giving me permission to write tests where I think they'll be most useful even if the rest of my code coverage is woefully incomplete.
7. TDD helps in all kinds of surprising ways down the line. Good unit tests can help document what something is supposed to do, they can help you migrate code from one project to another and give you an unwarranted feeling of superiority over your non-testing colleagues :)

This presentation is an excellent introduction to all the yummy goodness testing entails.

Share Improve this answer

Follow

edited Jan 30, 2014 at 19:41

guntbert

536 ● 6 ● 19

answered Aug 24, 2008 at 21:58

reefnet_alex

9,745 ● 5 ● 34 ● 32

All of these answers really encourage me to start learning and using Unit Testing, even if it may take a lot of time. But, mate, your link doesn't work, if you could fix that, it would be awesome) – Isi Aug 16, 2021 at 8:16

7

I would like to recommend the xUnit Testing Patterns book by Gerard Meszaros. It's large but is a great resource on unit testing. Here is a link to his web site where he discusses the basics of unit testing. http://xunitpatterns.com/XUnitBasics.html

Share Improve this answer Follow

answered Mar 14, 2010 at 18:10

Paul Hildebrandt
2,754 ● 28 ● 26

5

I use unit tests to save time.

When building business logic (or data access) testing functionality can often involve typing stuff into a lot of screens that may or may not be finished yet. Automating these tests saves time.

For me unit tests are a kind of modularised test harness. There is usually at least one test per public function. I write additional tests to cover various behaviours.

All the special cases that you thought of when developing the code can be recorded in the code in the unit tests. The unit tests also become a source of examples on how to use the code.

It is a lot faster for me to discover that my new code breaks something in my unit tests then to check in the code and have some front-end developer find a problem.

For data access testing I try to write tests that either have no change or clean up after themselves.

Unit tests aren't going to be able to solve all the testing requirements. They will be able to save development time and test core parts of the application.

Share Improve this answer Follow

answered Sep 17, 2008 at 0:38

Leah
3,342 ● 2 ● 33 ● 33

4

This is my take on it. I would say unit testing is the practice of writing software tests to verify that your real software does what it is meant to. This started with jUnit in the Java world and has become a best practice in PHP as well with SimpleTest and phpUnit. It's a core practice of Extreme Programming and helps you to be sure that your software still works as intended after editing. If you have sufficient test coverage, you can do major refactoring, bug fixing or add features rapidly with much less fear of introducing other problems.

It's most effective when all unit tests can be run automatically.

Unit testing is generally associated with OO development. The basic idea is to create a script which sets up the environment for your code and then exercises it; you write assertions, specify the intended output that you should receive and then execute your test script using a framework such as those mentioned above.

The framework will run all the tests against your code and then report back success or failure of each test. phpUnit is run from the Linux command line by default, though there are HTTP interfaces available for it. SimpleTest is web-based by nature and is much easier to get up and running, IMO. In combination with xDebug, phpUnit can give you automated statistics for code coverage which some people find very useful.

Some teams write hooks from their subversion repository so that unit tests are run automatically whenever you commit changes.

It's good practice to keep your unit tests in the same repository as your application.

Share Improve this answer Follow

answered Aug 4, 2008 at 23:53

Polsonby
22.9k ● 19 ● 60 ● 74

4

LibrarIES like NUnit, xUnit or JUnit are just mandatory if you want to develop your projects using the TDD approach popularized by Kent Beck:

You can read *Introduction to Test Driven Development (TDD)* or Kent Beck's book *Test Driven Development: By Example*.

Then, if you want to be sure your tests cover a "good" part of your code, you can use software like NCover, JCover, PartCover or whatever. They'll tell you the coverage percentage of your code. Depending on how much you're adept at TDD, you'll know if you've practiced it well enough :)

Share Improve this answer Follow

edited Oct 22, 2011 at 16:41
Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Mar 14, 2010 at 17:22
pierroz
7,860 ● 9 ● 51 ● 60

3

Unit-testing is the testing of a unit of code (e.g. a single function) without the need for the infrastructure that that unit of code relies on. i.e. test it in isolation.

If, for example, the function that you're testing connects to a database and does an update, in a unit test you might not want to do that update. You would if it were an integration test but in this case it's not.

So a unit test would exercise the functionality enclosed in the "function" you're testing without side effects of the database update.

Say your function retrieved some numbers from a database and then performed a standard deviation calculation. What are you trying to test here? That the standard deviation is calculated correctly or that the data is returned from the database?

In a unit test you just want to test that the standard deviation is calculated correctly. In an integration test you want to test the standard deviation calculation and the database retrieval.

Share Improve this answer Follow

answered Aug 15, 2008 at 17:42

Guy
67.1k ● 101 ● 265 ● 331

3

Unit testing is about writing code that tests your application code.

The *Unit* part of the name is about the intention to test small units of code (one method for example) at a time.

xUnit is there to help with this testing - they are frameworks that assist with this. Part of that is automated test runners that tell you what test fail and which ones pass.

They also have facilities to setup common code that you need in each test before hand and tear it down when all tests have finished.

You can have a test to check that an expected exception has been thrown, without having to write the whole try catch block yourself.

Share Improve this answer Follow

answered Mar 14, 2010 at 17:19

Oded

498k ● 102 ● 893 ● 1k

3

I think the point that you don't understand is that unit testing frameworks like NUnit (and the like) will help you in *automating* small to medium-sized tests. Usually you can run the tests in a GUI (that's the case with NUnit, for instance) by simply clicking a button and then - hopefully - see the progress bar stay green. If it turns red, the framework shows you which test failed and what exactly went wrong. In a normal unit test, you often use assertions, e.g. `Assert.AreEqual(expectedValue, actualValue, "some description")` - so if the two values are unequal you will see an error saying "some description: expected <expectedValue> but was <actualValue>".

So as a conclusion unit testing will make testing faster and a lot more comfortable for developers. You can run all the unit tests before committing new code so that you don't break the build process of other developers on the same project.

Share Improve this answer Follow

answered Mar 14, 2010 at 17:25

AndiDog

70k ● 21 ● 163 ● 207

3

Use Testivus. All you need to know is right there :)

Share Improve this answer

Follow

edited Oct 22, 2011 at 16:30

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Sep 17, 2008 at 0:48

MetroidFan2002

29.9k ● 16 ● 65 ● 81

3

Unit testing is a practice to make sure that the function or module which you are going to implement is going to behave as expected (requirements) and also to make sure how it behaves in scenarios like boundary conditions, and invalid input.

xUnit, NUnit, mbUnit, etc. are tools which help you in writing the tests.

Share Improve this answer

Follow

edited Oct 22, 2011 at 16:37

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Mar 14, 2010 at 17:22

Rony

9,481 ● 2 ● 23 ● 22

Test Driven Development has sort of taken over the term Unit Test. As an old timer I will mention the more generic

2

definition of it.

Unit Test also means testing a single component in a larger system. This single component could be a dll, exe, class library, etc. It could even be a single system in a multi-system application. So ultimately Unit Test ends up being the testing of whatever you want to call a single piece of a larger system.

You would then move up to integrated or system testing by testing how all the components work together.

Share Improve this answer Follow

answered Aug 24, 2008 at 22:34

bruceatk
5,138 ● 2 ● 27 ● 36

2

First of all, whether speaking about Unit testing or any other kinds of automated testing (Integration, Load, UI testing etc.), the key difference from what you suggest is that it is automated, repeatable and it doesn't require any human resources to be consumed (= nobody has to perform the tests, they usually run at a press of a button).

Share Improve this answer Follow

answered Mar 14, 2010 at 17:22

Tomas Vana
18.8k ● 9 ● 56 ● 66

1

I went to a presentation on unit testing at FoxForward 2007 and was told never to unit test anything that works with data. After all, if you test on live data, the results are unpredictable, and if you don't test on live data, you're not actually testing the code you wrote. Unfortunately, that's most of the coding I do these days. :-)

I did take a shot at TDD recently when I was writing a routine to save and restore settings. First, I verified that I could create the storage object. Then, that it had the method I needed to call. Then, that I could call it. Then, that I could pass it parameters. Then, that I could pass it specific parameters. And so on, until I was finally verifying that it would save the specified setting, allow me to change it, and then restore it, for several different syntaxes.

I didn't get to the end, because I needed-the-routine-now-dammit, but it was a good exercise.

Share Improve this answer Follow

answered Aug 22, 2008 at 18:10

SarekOfVulcan
1,358 ● 5 ● 16 ● 36

1

> What do you do if you are given a pile of crap and seem like you are stuck in a perpetual state of cleanup that you know with the addition of any new feature or code can break the current set because the current software is like a house of cards?
>
> How can we do unit testing then?

You start small. The project I just got into had no unit testing until a few months ago. When coverage was that low, we would simply pick a file that had no coverage and click "add tests".

Right now we're up to over 40%, and we've managed to pick off most of the low-hanging fruit.

(The best part is that even at this low level of coverage, we've already run into many instances of the code doing the wrong thing, and the testing caught it. That's a huge motivator to push people to add more testing.)

Share Improve this answer Follow

edited Jun 20, 2020 at 9:12
Community Bot
1 ● 1

answered Aug 22, 2008 at 18:19
Adam V
6,336 ● 3 ● 41 ● 52

1

This answers why you should be doing unit testing.

The 3 videos below cover unit testing in javascript but the general principles apply across most languages.

Unit Testing: Minutes Now Will Save Hours Later - Eric Mann - https://www.youtube.com/watch?v=_UmmaPe8Bzc

JS Unit Testing (very good) - https://www.youtube.com/watch?v=-IYqgx8JxlU

Writing Testable JavaScript - https://www.youtube.com/watch?v=OzjogCFO4Zo

---

Now I'm just learning about the subject so I may not be 100% correct and there's more to it than what I'm describing here but my basic understanding of unit testing is that you write some test code (which is kept separate from your main code) that calls a function in your main code with input (arguments) that the function requires and the code then checks if it gets back a valid return value. If it does get back a valid value the unit testing framework that you're using to run the tests shows a green light (all good) if the value is invalid you get a red light and you then can fix the problem straight away before you release the new code to production, without testing you may actually not have caught the error.

So you write tests for you current code and create the code so that it passes the test. Months later you or someone else need to modify the function in your main code, because earlier you had already written test code for that function you now run again and the test may fail because the coder introduced a logic error in the function or return something completely different than what that function is supposed to return. Again without the test in place that error might be hard to track down as it can possibly affect other code as well and will go unnoticed.

Also the fact that you have a computer program that runs through your code and tests it instead of you manually doing it in the browser page by page saves time (unit testing for javascript). Let's say that you modify a function that is used by some script on a web page and it works all well and good for its new intended purpose. But, let's also say for arguments sake that there is another function you have somewhere else in your code that depends on that newly modified function for it to operate properly. This dependent function may now stop working because of the changes that you've made to the first function, however without tests in place that are run automatically by your computer you will not notice that there's a problem with that function until it is actually executed and you'll have to manually navigate to a web page that includes the script which executes the dependent function, only then you notice that there's a bug because of the change that you made to the first function.

To reiterate, having tests that are run while developing your application will catch these kinds of problems as you're coding. Not having the tests in place you'd have to manually go through your whole application and even then it can be hard to spot the bug, naively you send it out into production and after a while a kind user sends you a bug report (which won't be as good as your error messages in a testing framework).

It's quite confusing when you first hear of the subject and you think to yourself, am I not already testing my code?

And the code that you've written is working like it is supposed to already, "why do I need another framework?"... Yes you are already testing your code but a computer is better at doing it. You just have to write good enough tests for a function/unit of code once and the rest is taken care of for you by the mighty cpu instead of you having to manually check that all of your code is still working when you make a change to your code.

Also, you don't have to unit test your code if you don't want to but it pays off as your project/code base starts to grow larger as the chances of introducing bugs increases.

Share Improve this answer Follow

edited Jul 21, 2015 at 12:00

answered Jul 18, 2015 at 18:34

Stephen Brown
74 ● 5

0

Unit-testing and TDD in general enables you to have shorter feedback cycles about the software you are writing. Instead of having a large test phase at the very end of the implementation, you incrementally test everything you write. This increases code quality very much, as you immediately see, where you might have bugs.

Share Improve this answer Follow

answered May 3, 2017 at 9:33

aap
55 ● 10

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.