When to unit-test vs manual test

Asked 15 years, 9 months ago Modified 6 years, 1 month ago

Viewed 13k times

40

While unit-testing seems effective for larger projects where the APIs need to be industrial strength (for example development of the .Net framework APIs, etc.), it seems possibly like overkill on smaller projects.

When is the automated TDD approach the best way, and when might it be better to just use manual testing techniques, log the bugs, triage, fix them, etc.

Another issue--when I was a tester at Microsoft, it was emphasized to us that there was a value in having the developers and testers be different people, and that the tension between these two groups could help create a great product in the end. Can TDD break this idea and create a situation where a developer might not be the right person to rigorously find their own mistakes? It may be automated, but it would seem that there are many ways to write the tests, and that it is questionable whether a given set of tests will "prove" that quality is acceptable.

unit-testing testing tdd

edited Mar 2, 2009 at 7:26

Share

Improve this question

Follow

Steve Rowe

19.4k ● 9 ● 53 ● 82

asked Mar 2, 2009 at 6:32

alchemical

13.9k ● 24 ● 84 ● 111

Unit tests can offer a form of documentation of code that may be very useful if someone other than the author of the code has to make a change to fix a bug or put in a new feature/enhancement. Developers should have to test their code, but those tests shouldn't be the only ones, IMO.
– JB King Mar 13, 2009 at 20:54

15 Answers

Sorted by: Highest score (default)

75

The effectiveness of TDD is independent of project size. I will practice the three laws of TDD even on the smallest programming exercise. The tests don't take much time to write, and they save an enormous amount of debugging time. They also allow me to refactor the code without fear of breaking anything.

TDD is a discipline similar to the discipline of dual-entry-bookkeeping practiced by accountants. It prevents errors in-the-small. Accountants will enter every transaction twice; once as a credit, and once as a debit. If no simple errors were made, then the balance sheet will sum to zero. That zero is a simple spot check that prevents the executives from going to jail.

By the same token programmers write unit tests in advance of their code as a simple spot check. In effect, they write each bit of code twice; once as a test, and once as production code. If the tests pass, the two bits of code are in agreement. Neither practice protects against larger and more complex errors, but both practices are nonetheless valuable.

The practice of TDD is not really a testing technique, it is a development practice. The word "test" in TDD is more or less a coincidence. As such, TDD is not a replacement for good testing practices, and good QA testers. Indeed, it is a very good idea to have experienced testers write QA test plans independently (and often in advance of) the programmers writing the code (and their unit tests).

It is my preference (indeed my passion) that these independent QA tests are also automated using a tool like FitNesse, Selenium, or Watir. The tests should be easy to read by business people, easy to execute, and utterly unambiguous. You should be able to run them at a moment's notice, usually many times per day.

Every system also needs to be tested manually. However, manual testing should never be rote. A test that can be scripted should be automated. You only want to put humans in the loop when human judgement is needed. Therefore humans should be doing exploratory testing, not blindly following test plans.

So, the short answer to the question of when to unit-test versus manual test is that there is no "versus". You

should write automated unit tests *first* for the vast majority of the code you write. You should have automated QA acceptance tests written by testers. And you should also practice strategic exploratory manual testing.

Share Improve this answer

Follow

edited Apr 17, 2018 at 7:20

TT.

16.1k ● 6 ● 50 ● 93

answered Mar 7, 2009 at 20:59

Uncle Bob

4,750 ● 30 ● 19

1 So nice to see you here *Dear Uncle Bob* ...+1 for your unique examples...like dual-entry-bookkeeping here. Thanks.
– Snesh Dec 19, 2014 at 10:26

7

Unit tests aren't meant to replace functional/component tests. Unit tests are really focused, so they won't be hitting database, external services, etc. Integration tests does that, but you can have them really focused. The bottom line, is that on the specific question, the answer is that they don't replace those manual tests. Now, automated functional tests + automated component tests can certainly replace manual tests. It will depend a lot of the project and the approach to it on who will actually do those.

Update 1: Note that if developers are creating automated functional tests you still want to review that those have the appropriate coverage, complementing them as

appropriate. Some developers create automated functional tests with their "unit" test framework, because they still have to do smoke tests regardless of the unit tests, and it really helps having those automated :)

Update 2: Unit testing isn't overkill for a small project, nor is automating the smoke tests or using TDD. What is overkill is having the team doing any of that for their first time on the small project. Doing any of those have an associated learning curve (specially unit testing or TDD), and not always will be done right at first. You also want someone who has been doing it for a while involved, to help avoid pitfalls and get pasts some coding challenges that aren't obvious when starting on it. The issue is that it isn't common for teams to have these skills.

Share Improve this answer Follow

edited Mar 2, 2009 at 7:05

answered Mar 2, 2009 at 6:37

eglasius

36k ● 5 ● 68 ● 110

4

TDD is the best approach whenever it is feasible to do so. TDD testing is automatic, quantifiable through code coverage, and reliable method of ensuring code quality.

Manual testing requires a huge amount of time (as compared to TDD) and suffers from human error.

There is nothing saying that TDD means only developers test. Developers **should** be responsible for coding a percentage of the test framework. QA should be responsible for a much larger portion. Developers test APIs the way they want to test them. QA tests APIs in ways that I really wouldn't have ever thought to and do things that, while seemingly crazy, are actually done by customers.

Share Improve this answer

Follow

answered Mar 2, 2009 at 6:36

JaredPar

753k ● 151 ● 1.3k ● 1.5k

3

I would say that unit-tests are a programmers aid to answer the question:

> Does this code do what I think it does?

This is a question they need to ask themselves alot. Programers like to automate anything they do alot where they can.

The separate test team needs to answer a different question:-

> Does this system do what I (and the end users) expect it to do? Or does it suprise me?

There are a whole massive class of bugs related to the programer or designers having a different idea about what is correct that unit tests will never pickup.

Share Improve this answer

Follow

answered Mar 2, 2009 at 6:37

WW.

24.3k ● 15 ● 97 ● 124

> This is how we work in safety critical software. The programmer proves their code is safe in all states/branches. The tester (system level) independently tests their interpretation of what is often a high level functional requirement. It's easy to write perfect code which doesn't actually meet the customers requirement. – MattP Aug 11, 2015 at 23:22

3

According to studies of various projects (1), Unit tests find **15..50%** of the defects (average of 30%). This doesn't make them the worst bug finder in your arsenal, but not a silver bullet either. There *are* no silver bullets, any good QA strategy consists of multiple techniques.

A test that is automated runs more often, thus it will find defects earlier and reduce total cost of these immensely - that is the true value of test automation.

Invest your ressources wisely and pick the low hanging fruit first.
I find that automated tests are easiest to write and to maintain for small units of code - isolated functions and classes. End user functionality is easier tested manually -

and a good tester will find many oddities beyond the required tests. Don't set them up against each other, you need both.

Dev vs. Testers Developers are notoriously bad at testing their own code: reasons are psychological, technical and last not least economical - testers are usually cheaper than developers. But developers can do their part, and make testing easier. TDD makes testing an intrinsic part of program construction, not just an afterthought, that is the true value of TDD.

Another interesting point about testing: There's no point in 100% coverage. Statistically, bugs follow an 80:20 rule - the majority of bugs is found in small sections of code. Some studies suggest that this is even sharper - and tests should focuse on the places where bugs turn up.

---

(1) Programming Productivity Jones 1986 u.a., quoted from Code Complete, 2nd. ed. But as others have said, *unit* tests are only one part of tests, integration, regression and system tests can be - at leat partially - automated as well.

My interpretation of the results: "many eyes" has the best defect detection, but only if you have some formal process that makes them actually look.

 edited May 8, 2017 at 13:08

Follow

answered Mar 7, 2009 at 23:16

peterchen

41.1k ● 22 ● 108 ● 193

Studies link is a dead link. – Adam Parkin May 6, 2017 at 16:48

@AdamParkin: Replaced with google search of common title.Let's see how long this one lasts (couldn't quickly find a reliable source that cites actual source of data) – peterchen May 8, 2017 at 13:09

2

Every application gets tested.

Some applications get tested in the form of does my code compile and does the code appear to **function**.

Some applications get tested with **Unit tests**. Some developers are religious about Unit tests, TDD and code coverage to a fault. Like everything, too much is more often than not bad.

Some applications are luckily enough to get tested via a **QA** team. Some QA teams automate their testing, others write test cases and manually test.

Michael Feathers, who wrote: Working Effectively with Legacy Code, wrote that code not wrapped in tests is legacy code. Until you have experienced The Big Ball of Mud, I don't think any developer truly understands the

benefit of good Application Architecture and a suite of well written Unit Tests.

Having different people test is a great idea. The more people that can look at an application the more likely all the scenarios will get covered, including the ones you didn't intend to happen.

TDD has gotten a bad rap lately. When I think of **TDD** I think of dogmatic developers meticulously writing tests before they write the implementation. While this is true, what has been overlooked is by writing the tests, (first or shortly after) the developer experiences the method/class in the shoes of the consumer. Design flaws and shortcomings are immediately apparent.

I argue that the **size of the project** is irrelevant. What is important **is the lifespan** of the project. The longer a project lives the more the likelihood that a developer other than the one who wrote it will work on it. Unit tests are documentation to the expectations of the application -- A manual of sorts.

Share Improve this answer

Follow

answered Mar 2, 2009 at 7:47

Chuck Conway

16.4k ● 11 ● 61 ● 102

1

Unit tests can only go so far (as can all other types of testing). I look on testing as a kind of "sieve" process. Each different type of testing is like a sieve that you are placing under the outlet of your development process.

The stuff that comes out is (hopefully) mostly features for your software product, but it also contains bugs. The bugs come in lots of different shapes and sizes.

Some of the bugs are pretty easy to find because they are big or get caught in basically any kind of sieve. On the other hand, some bugs are smooth and shiny, or don't have a lot of hooks on the sides so they would slip through one type of sieve pretty easily. A different type of sieve might have different shape or size holes so it will be able to catch different types of bugs. The more sieves you have, the more bugs you will catch.

Obviously the more sieves you have in the way, the slower it is for the features to get through as well, so you'll want to try to find a happy medium where you aren't spending too much time testing that you never get to release any software.

Share Improve this answer

Follow

answered Mar 2, 2009 at 6:40

1800 INFORMATION

135k ● 30 ● 163 ● 242

1

The nicest point (IMO) of automated unit tests is that when you change (improve, refactor) the existing code, it's easy to test that you didn't break it. It would be tedious to test everything manually again and again.

answered Mar 2, 2009 at 7:22

Joonas Pulakka
36.6k • 29 • 108 • 171

1

Your question seems to be more about automated testing vs manual testing. Unit testing is a form of automated testing but a very specific form.

Your remark about having separate testers and developers is right on the mark though. But that doesn't mean developers shouldn't do some form of verification.

Unit testing is a way for developers to get fast feedback on what they're doing. They write tests to quickly run small units of code and verify their correctness. It's not really testing in the sense you seem to use the word testing just like a syntax check by a compiler isn't testing. Unit testing is a development technique. Code that's been written using this technique is probably of higher quality than code written without but still has to go through quality control.

The question about automated testing vs manual testing for the test department is easier to answer. Whenever the project gets big enough to justify the investment of writing automated tests you should use automated tests. When you've got lots of small one-time tests you should do them manually.

answered Mar 7, 2009 at 21:15

Mendelt
37.5k 6 75 97

1

Having been on both sides, QA and development, I would assert that someone should always manually test your code. Even if you are using TDD, there are plenty of things that you as a developer may not be able to cover with unit tests, or may not think about testing. This especially includes usability and aesthetics. Aesthetics includes proper spelling, grammar, and formatting of output.

Real life example 1:

A developer was creating a report we display on our intranet for managers. There were many formulas, all of which the developer tested before the code came to QA. We verified that the formulas were, indeed, producing the correct output. What we asked development to correct, almost immediately, was the fact that the numbers were displayed in pink on a purple background.

Real life example 2:

I write code in my spare time, using TDD. I like to think I test it thoroughly. One day my wife walked by when I had a message dialog up, read it, and promptly asked, "What on Earth is that message supposed to mean?" I thought the message was rather clear, but when I reread it I realized it was talking about parent and child nodes in a tree control, and probably wouldn't make sense to the

average user. I reworded the message. In this case, it was a usability issue, which was not caught by my own testing.

Share Improve this answer Follow

answered Mar 13, 2009 at 20:48

ssakl

690 ● 2 ● 10 ● 23

1

> unit-testing seems effective for larger projects where the APIs need to be industrial strength, it seems possibly like overkill on smaller projects.

It's true that unit tests of a moving API are brittle, but unit-testing is also effective on API-less projects such as applications. Unit-testing is meant to test the units a project is made with. It allows ensuring every unit works as expected. This is a real safety net when modifying - refactoring - the code.

As far as the size of the project is concerned, It's true that writing unit-tests for a small project can be overkill. And here, I would define small project as a small program, that can be tested manually, but very easily and quickly, in no more than a few seconds. Also a small project can grow, in which case it might be advantageous to have unit tests at hand.

> there was a value in having the developers and testers be different people, and that the tension

> between these two groups could help create a great product in the end.

Whatever the development process, unit-testing is not meant to supersede any other stages of test, but to complement them with tests at the development level, so that developers can get very early feedback, without having to wait for an official build and official test. With unit-testing, development team delivers code that works, downstream, not bug-free code, but code that can be tested by the test team(s).

To sum up, I test manually when it's really very easy, or when writing unit tests is too complex, and I don't aim to 100% coverage.

Share Improve this answer Follow

edited Sep 1, 2009 at 11:39

answered Mar 2, 2009 at 12:46

philant

35.7k ● 11 ● 73 ● 113

1

I believe it is possible to combine the expertise of QA/testing staff (defining the tests / acceptance criteria), with the TDD concept of using a developer owned API (as oppose to GUI or HTTP/messaging interface) to drive an application under test.

It is still critical to have independent QA staff, but we don't need huge manual test teams anymore with modern test tools like FitNesse, Selenium and Twist.

Share Improve this answer

Follow

edited Apr 16, 2011 at 5:38

Cody Gray ♦

244k ● 52 ● 501 ● 581

answered Apr 15, 2011 at 20:39

Adam Smith

11 ● 1

0

Just to clarify something many people seem to miss:

TDD, in the sense of *"write failing test, write code to make test pass, refactor, repeat"* Is usually most efficient and useful when you write **unit** tests.

You write a unit test around just the class/function/unit of code you are working on, using mocks or stubs to abstract out the rest of the system.

"Automated" testing usually refers to higher level integration/acceptance/functional tests - you *can* do TDD around this level of testing, and it's often the only option for heavily ui-driven code, but you should be aware that this sort of testing is more fragile, harder to write test-first, and no substitute for unit testing.

Share Improve this answer

Follow

answered Mar 8, 2009 at 10:31

Korny

0

TDD gives me, as the developer, confidence that the change I am making to the code has the intended consequences and ONLY the intended consequences, and thus the metaphor of TDD as a "safety net" is useful; change any code in a system without it and you can have no idea what else you may have broken.

Engineering tension between developers and testers is really bad news; developers cultivate a "well, the testers are paid to find the bugs" mindset (leading to laziness) and the testers -- feeling as if they aren't being seen to do their jobs if they don't find any faults -- throw up as many trivial problems as they can. This is a gross waste of everyone's time.

The best software development, in my humble experience, is where the tester is also a developer and the unit tests and code are written together as part of a pair programming exercise. This immediately puts the two people on the same side of the problem, working together towards the same goal, rather than putting them in opposition to each other.

Share Improve this answer

Follow

answered Jan 19, 2013 at 16:40

Matt

1,678 ● 12 ● 23

0

Unit testing is not the same as functional testing. And as far as automation is concerned, it should normally be considered when the testing cycle will be repeated more than 2 or 3 times... It is preferred for regression testing. If the project is small or it will not have frequent changes or updates then manual testing is a better and less cost effective option. In such cases automation will prove to be more costly with the script writing and maintainence.

Share Improve this answer

Follow

answered Jul 17, 2014 at 12:05

IAmMilinPatel
445 ● 1 ● 12 ● 21