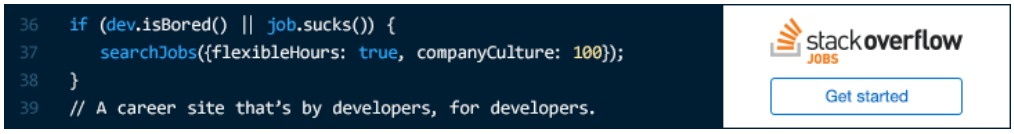
Is there hard evidence of the ROI of unit testing?



Unit testing sounds great to me, but I'm not sure I should spend any time really learning it unless I can convince others that is has significant value. I have to convince the other programmers and, more importantly, the bean-counters in management, that all the extra time spent learning the testing framework, writing tests, keeping them updated, etc.. will pay for itself, and then some.

What proof is there? Has anyone actually developed the same software with two separate teams, one using unit testing and the other not, and compared the results? I doubt it. Am I just supposed to justify it with, "Look it up on the Internet, everybody's talking about it, so it must be the right thing to do"?

Where is the hard evidence that will convince the laymen that unit testing is worth the effort?

unit-testing tdd

edited Nov 13 '16 at 20:28
Dariusz Woźniak
4,629 3 31 56

asked Oct 25 '08 at 20:59
raven
12.4k 12 64 103

11 Answers

Yes. This is a link to a study by Boby George and Laurie Williams at NCST and a another by Nagappan et al. I'm sure there are more. Dr. Williams publications on testing may provide a good starting point for finding them.

[EDIT] The two papers above specifically reference TDD and show 15-35% increase in initial development time after adopting TDD, but a 40-90% decrease in pre-release defects. If you can't get at the full text versions, I suggest using Google Scholar to see if you can find a publicly available version.

edited Jul 5 '11 at 19:31
raven
12.4k 12 64 103

answered Oct 25 '08 at 21:46
tvanfosson
368k 71 589 708

7 The first study compares agile+TDD against waterfall projects, it's results would be more relevant if it had compared two agile teams. The second study mentions other studies that found little to no quality bonus for TDD projects. And when you compare management's estimates about needed extra time for TDD it's significantly estimated higher for the two teams with a high domain expertise, yet they also have a 20% lower test coverage. This confirms my own experience, I find assurance much more important in systems I haven't worked with yet, while testing is a hindrance for everything else. – LearnCocos2D Oct 14 '13 at 17:45

Neither of the studies compares comparable process model with only the testmethofology change. That is is spending the time used on UT actually better spent on eg. system testing. As it stands it might as well be " if we test smarter does that help" study. – Rune FS Mar 23 '14 at 8:40

So what if the cost of fixing the post release bugs is 0.01% of total development? TDD would be a terrible investment in that case. And if the bugs are few? These %s mean nothing without context. To be fair I am yet to read the whole study. But as it stands your post is useful (good links) but does not answer the question regarding ROI, IMO. – Instine Jun 13 '14 at 10:56

@Instine Luckily (?) there is good evidence that this is not the case. Fixing post-release bugs is exponentially more expensive than bugs found early in development (which is what TDD does). In that context, a cost of 0.01% of total development for all post-release bugs seems unlikely. (For details see *Code Complete*, in particular Boehm *& al.*, “Understanding and Controlling Software Costs”, IEEE Trans Softw Eng (1988)). – Konrad Rudolph Nov 4 '14 at 10:52

" I have to convice the other programmers and, more importantly, the bean-counters in management, that all the extra time spent learning the testing framework, writing tests, keeping them updated, etc.. will pay for itself, and then some."

Why?

Why not just do it, quietly and discretely. You don't have to do it all at once. You can do this in little tiny pieces.

The framework learning takes very little time.

Writing one test, just one, takes very little time.

Without unit testing, all you have is some confidence in your software. With one unit test, you still have your confidence, plus proof that at least one test passes.

That's all it takes. No one needs to know you're doing it. Just do it.

answered Oct 25 '08 at 22:03

S.Lott
274k 55 387 665

8 The bean counters couldn't tell a unit test from the rest of the code if their lives depended on it. I support the suggestion to just do it. There's one caveat, though: If you are not alone, you need your fellow developers to embrace this practice. If not, they will unintentionally break your tests. – Thomas Eyde Oct 25 '08 at 23:54

Just do it and don't tell them, and sell the idea to your colleges at the coffee break ;-) – Johan Feb 17 '09 at 20:06

Well... they could say that "even if unit test works for you but it's not working for me because of my projects are more difficult than you" – Graviton Nov 5 '09 at 14:52

2 Because you'd get fired when you consistently didn't hit your deadlines? – Andrew Jan 16 '10 at 5:09

3 @Neko: Unit tests don't add a "bit of overhead". They *reduce* the overall workload by preventing a whole flood of dumb mistakes. The work does not grow; it simply shifts in nature from bad code to good unit tests and good code. – S.Lott Jan 29 '12 at 14:19

I take a different approach to this:

What assurance do you have that your code is correct? Or that it doesn't break assumption X when someone on your team changes func1()? Without unit tests keeping you 'honest', I'm not sure you have much assurance.

The notion of keeping tests updated is interesting. The tests themselves don't often have to change. I've got 3x the test code compared to the production code, and the test code has been changed *very* little. It is, however, what lets me sleep well at night and the thing that allows me to tell the customer that I have confidence that I can implement the Y functionality without breaking the system.

Perhaps in academia there is evidence, but I've never worked anywhere in the commercial world where anyone would pay for such a test. I can tell you, however, that it has worked well for me, took little time to get accustomed to the testing framework and writing test made me **really** think about my requirements and the design, far more than I ever did when working on teams that wrote no tests.

Here's where it pays for itself: 1) You have confidence in your code and 2) You catch problems earlier than you would otherwise. You don't have the QA guy say "hey, you didn't bother bounds-checking the xyz() function, did you? **He** doesn't get to find that bug because **you** found it a month ago. That is good for him, good for you, good for the company and good for the customer.

Clearly this is anecdotal, but it has worked wonders for me. Not sure I can provide you with spreadsheets, but my customer is happy and that is the end goal.

answered Oct 25 '08 at 21:21

itsmatt
24.5k 9 80 147

My QA guy was pretty sharp but he wasn't looking at code, but it was easy to tell the bounds weren't checked. – itsmatt Oct 25 '08 at 22:04

Totally agreed about unit testing forcing you to think more about your design and correctness rather than code recklessly – chakrit Oct 25 '08 at 22:28

4 Customers don't pay us to write tests. Then again, they don't pay us to write code, either. They pay us to solve their problems, and when confronted, I bet they also want the problems to stay solved. Given the evidence, it's unbelievable customers don't want to secure their investment. – Thomas Eyde Oct 25 '08 at 23:47

We've demonstrated with hard evidence that it's possible to write crappy software without Unit Testing. I believe there's even evidence for crappy software with Unit Testing. But this is not the point.

Unit Testing or Test Driven Development (TDD) is a Design technique, not a test technique. Code that's written test driven looks completely different from code that is not.

Even though this is not your question, I wonder if it's really the easiest way to go down the road and answer questions (and bring evidence that might be challenged by other reports) that might be asked wrong. Even if you find hard evidence for your case - somebody else might find hard evidence against.

Is it the business of the bean counters to determine how the technical people should work? Are they providing the cheapest tools in all cases because they believe you don't need more expensive ones?

This argument is either won based on trust (one of the fundamental values of agile teams) or lost based on role power of the winning party. Even if the TDD-proponents win based on role power I'd count it as lost.

answered Oct 25 '08 at 21:11

Olaf Kock
31.5k 6 35 72

10 hear, hear :) A lot of the hard evidence for TDD also comes from very experienced teams that were already getting good results without it. TDD just improved their results rather than creating them out of thin air. The real ROI is hiring decent coders and letting them decide how to do things. – workmad3 Oct 25 '08 at 21:17

"Is it the business of the bean counters to determine how the technical people should work?" --> all business decisions come down to money. Still, good answer, +1 – jcollum Feb 17 '09 at 17:23

@jcollum but how you perform your job has nothing to do with money and if you wish dome one to be accountable you let them decide HOW they do the WHAT you asked of them – Rune FS Mar 23 '14 at 8:50

Here's a great and entertaining read of a guy changing his company from within. It's not limited to TDD. http://jamesshore.com/Change-Diary/ Note that he didn't persuade the "bean counters" for quite some time and did "guerilla tactics" instead.

answered Oct 27 '08 at 19:48

Epaga
16.6k 47 123 218

the link looks interesting... worth checking out re: changing organisations work processes... – nasty pasty Feb 18 '09 at 5:13

More about TDD than strictly unit testing, here is a link to the Realizing quality improvement through test driven development: results and experiences of four industrial teams paper, by Nagappan, E. Michael Maximilien, Thirumalesh Bhat, and Laurie Williams. paper published by Microsoft Empirical Software Engineering and Measurement (ESM) group and already mentionned here.

The team found was that the TDD teams produced code that is between 60% and 90% percent better (in terms of defect density) than non-TDD teams. *However* TDD teams took between 15% and 35% longer to complete their projects.

answered Oct 20 '09 at 6:58


philant
24k 10 49 92

Well, there are some large companies that require you to use unit testing but if you are a small company why mimic large ones?

For me when I started with unit testing , many years ago,(today we mostly use behavior model) it was because I could not control all the path in one application.

I was used to bottom first programming and a REPL so when I got Unit Test (One Test for Every Function) it was like bringing back a REPL to languages that where very much compile. It brought the fun back to every line of code I wrote. I felt god. I liked it. I didn't need a report to tell me that I begun writing better code faster. My boss didn't need a report to notice that because we where doing crazy stuff we suddenly never missed a deadline. My boss didn't need a report to notice that the number of "plain" bugs drop from (to many) to nearly nil because of this very strange thing of writing non-productive code.

As another poster already wrote, you don't use TDD to Test (verify). You write it to capture the specification, the behaviour of what your unit(object, module, function, class, server, cluster) works.

There are lot of failures and success stories of switching to a different model of developing software in a lot of companies.

I just started to use it whenever I had something new to write. There is a old saying that goes somewhat hard for me to translate to english but:

> Start with something so simple that you don't notice that you do it. When training for a marathon, start by walking 9 meters and run 1 meter, repeat.

answered Oct 25 '08 at 21:29


Jonke
5,521 2 16 34

So, I should just do it? It's guaranteed to work, and it doesn't matter if no one else does it with me? – raven Oct 25 '08 at 21:39

Actually, this is a Joel Test: joelonsoftware.com/articles/fog0000000043.html. It sounds to me that you may have more problem than a lack of the Nobel Prize Award Study On Unit Test – Jonke Oct 26 '08 at 0:07

There are statistics that prove that fixing a bug found in the unit/integration test costs many times less than fixing once it's on the live system (they are based on monitoring thousand of real life projects).

Edit: for example, as pointed out, the book "Code Complete" reports on such studies (paragraph 20.3, "Relative Effectiveness of Quality Techniques"). But there is also private research in the consulting field that proves that as well.

edited Oct 25 '08 at 22:36

answered Oct 25 '08 at 21:03


Gabriele D'Antona
5,312 3 29 68

Can you point me to these statistics? – raven Oct 25 '08 at 21:05

1 This is covered in Steve McConnell's *Code Complete*, which is a book you probably want to have on your bookshelf for other reasons. – Robert Rossney Oct 25 '08 at 21:27

That's not related to the test method but to when in the process a bug is reported and further the time would be better spent finding bugs in the specifications since the cost of fixing them when finding them when developing is reported upto 1000 fold as expensive (a factor of 10 per development phase) – Rune FS Mar 23 '14 at 8:46

If you are also interested in evidence against unit testing here is one well researched and thought out article:

Why Most Unit Testing is Waste By James O Coplien (lean and agile guru)

answered Aug 17 '15 at 0:13


mmm
667 5 17

I do have one set of data points for this - from an experience that sold me on unit tests.

Many moons ago I was a fresh graduate working on a large VB6 project and had occasion to write a large body of stored procedure code. Of the subsystem I was writing it made up about 1/4 of the whole code base - around 13,000 LOC out of 50K or so.

I wrote a set of unit tests for the stored procedures but unit testing VB6 UI code is not really feasible without tools like Rational Robot; at least it wasn't back then.

The statistics from QA on the piece were that about 40 or 50 defects were raised on the whole subsystem, of which ***two*** originated from the stored procedures. That's *one defect per 6,500 lines of code* vs. 1 per 1,000-1,200 or so across the whole piece. Bear in mind also, that about 2/3 of the VB6 code was boilerplate code for error handling and logging, identical across all of the procedures.

Without too much handwaving you can ascribe at least an order-of-magnitude improvement in defect rates to the unit testing.

answered Feb 10 '09 at 14:04

ConcernedOfTunbridge Wells

46.1k 12 113 176

Just to add more information to these answers, there are two meta-analysis resources that may help out figuring out productivity & quality effects on academic and industry background:

Guest Editors' Introduction: TDD—The Art of Fearless Programming [link]

> All researchers seem to agree that TDD encourages better task focus and test coverage. The mere fact of more tests doesn't necessarily mean that software quality will be better, but the increased programmer attention to test design is nevertheless encouraging. If we view testing as sampling a very large population of potential behaviors, more tests mean a more thorough sample. To the extent that each test can find an important problem that none of the others can find, the tests are useful, especially if you can run them cheaply.
>
> Table 1. A summary of selected empirical studies of test-driven development: industry participants*

Family of studies	Type	Development time analyzed	Legacy project?	Organization studied	Software built	Software size	No. of participants	Language	Productivity effect	Quality effect
Sanchez et al.[6]	Case study	5 years	Yes	IBM	Point-of-sale device driver	Medium	9–17	Java	Increased effort 19%	40%†
Bhat and Nagappan[7]	Case study	4 months	No	Microsoft	Windows networking common library	Small	6	C/C++	Increased effort 25–35%	62%†
	Case study	≈7 months	No	Microsoft	MSN Web services	Medium	5–8	C++/C#	Increased effort 15%	76%†
Canfora et al.[8]	Controlled experiment	5 hours	No	Soluziona Software Factory	Text analyzer	Very small	28	Java	Increased effort by 65%	Inconclusive based on quality of test
Damm and Lundberg[9]	Multi-case study	1–1.5 years	Yes	Ericsson	Components for a mobile network operator application	Medium	100	C++/Java	Total project cost increased by 5–6%	5–30% decrease in fault-slip-through rate; 55% decrease in avoidable fault costs
Melis et al.[10]	Simulation	49 days (simulated)	No	Calibrated using Klondike-Team and Quinary data	Market information project	Medium	4‡	Smalltalk	Increased effort 17%	36% reduction in residual defect density
Mann[11]	Case study	8 months	Yes	PetroSleuth	Windows-based oil and gas project management with statistical modeling elements	Medium	4–7	C#	n/a	81%§; customer and developers' perception of improved quality
Geras et al.[12]	Quasi-controlled experiment	3 hours	No	Various companies	Simple database-backed business information system	Small	14	Java	No effect	Inconclusive based on failure rates; improved based on no. of tests and frequency of execution
George and Williams[13]	Quasi-controlled experiment	4.75 hours	No	John Deere, Role Model Software, Ericsson	Bowling game	Very small	24	Java	Increased effort 16%	18%#
Ynchausti[14]	Case study	8.5 hours	No	Monster Consulting	Coding exercises	Small	5	n/a	Increased effort 60–100%	38–267%†

Table 2. A summary of selected empirical studies of TDD: academic participants*

Family of studies	Type	Development time analyzed	Legacy project?	Organization studied	Software built	Software size	No. of participants	Language	Productivity effect	Quality effect
Flohr and Schneider[15]	Quasi-controlled experiment	40 hours	Yes	University of Hannover	Graphical workflow library	Small	18	Java	Improved productivity by 27%	Inconclusive
Abrahamsson et al.[16]	Case study	30 days	No	VTT	Mobile application for global markets	Small	4	Java	Increased effort by 0% (iteration 5) to 30% (iteration 1)	No value perceived by developers
Erdogmus et al.[17]	Controlled experiment	13 hours	No	Politecnico di Torino	Bowling game	Very small	24	Java	Improved normalized productivity by 22%	No difference
Madeyski[18]	Quasi-controlled experiment	12 hours	No	Wroclaw University of Technology	Accounting application	Small	188	Java	n/a	–25 to –45%†
Melnik and Maurer[19]	Multi-case study	4-month projects over 3 years	No	University of Calgary/ SAIT Polytechnic	Various Web-based systems (surveying, event scheduling, price consolidation, travel mapping)	Small	240	Java	n/a	73% of respondents perceive TDD improves quality
Edwards[20]	Artifact analysis	2–3 weeks	No	Virginia Tech	CS1 programming assignment	Very small	118	Java	Increased effort 90%	45%†
Pančur et al.[21]	Controlled experiment	4.5 months	No	University of Ljubljana	4 programming assignments	Very small	38	Java	n/a	No difference
George[22]	Quasi-controlled experiment	1-3/4 hours	No	North Carolina State University	Bowling game	Very small	138	Java	Increased effort 16%	16%†
Müller and Hagner[23]	Quasi-controlled experiment	10 hours	No	University of Karlsruhe	Graph library	Very small	19	Java	No effect	No effect, but better reuse and improved program understanding

The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis [link]

Abstract:

> This paper provides a systematic meta-analysis of 27 studies that investigate the impact of Test-Driven Development (TDD) on external code quality and productivity.

> The results indicate that, in general, TDD has a small positive effect on quality but little to no discernible effect on productivity. However, subgroup analysis has found both the quality improvement and the productivity drop to be much larger in industrial studies in comparison with academic studies. A larger drop of productivity was found in studies where the difference in test effort between the TDD and the control group's process was significant. A larger improvement in quality was also found in the academic studies when the difference in test effort is substantial; however, no conclusion could be derived regarding the industrial studies due to the lack of data.
>
> Finally, the influence of developer experience and task size as moderator variables was investigated, and a statistically significant positive correlation was found between task size and the magnitude of the improvement in quality.

answered Nov 13 '16 at 20:27

Dariusz Woźniak
4,629 3 31 56