# What can you do to a legacy codebase that will have the greatest impact on improving the quality? [closed]

Asked 16 years, 2 months ago Modified 27 days ago Viewed 5k times

37

**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 27 days ago.

Improve this question

As you work in a legacy codebase what will have the greatest impact over time that will improve the quality of the codebase?

- Remove unused code
- Remove duplicated code
- Add unit tests to improve test coverage where coverage is low

- Create consistent formatting across files
- Update 3rd party software
- Reduce warnings generated by static analysis tools (i.e.Findbugs)

The codebase has been written by many developers with varying levels of expertise over many years, with a lot of areas untested and some untestable without spending a significant time on writing tests.

refactoring legacy legacy-code

Share

Improve this question

Follow

edited Mar 28, 2019 at 9:50

Dariusz Woźniak

**10.3k** ● 7 ● 63 ● 80

asked Sep 28, 2008 at 22:49

Craig Angus

**23.1k** ● 18 ● 57 ● 64

## 11 Answers

Sorted by: Highest score (default)

**37**

- Read Michael Feather's book "Working effectively with Legacy Code"

This is a GREAT book.

If you don't like that answer, then the best advice I can give would be:

- First, stop making new legacy code[1]

[1]: Legacy code = code without unit tests and therefore an unknown

Changing legacy code without an automated test suite in place is dangerous and irresponsible. Without good unit test coverage, you can't possibly know what affect those changes will have. Feathers recommends a "stranglehold" approach where you isolate areas of code you need to change, write some basic tests to verify basic assumptions, make small changes backed by unit tests, and work out from there.

NOTE: I'm not saying you need to stop everything and spend weeks writing tests for everything. Quite the contrary, just test around the areas you need to test and work out from there.

Jimmy Bogard and Ray Houston did an interesting screen cast on a subject very similar to this: http://www.lostechies.com/blogs/jimmy_bogard/archive/2008/05/06/pablotv-eliminating-static-dependencies-screencast.aspx

Share Improve this answer Follow

answered Sep 28, 2008 at 22:57

chadmyers

**3,810** ● 23 ● 30

---

I work with a legacy 1M LOC application written and modified by about 50 programmers.

> - Remove unused code

Almost useless... just ignore it. You wont get a big Return On Investment (ROI) from that one.

> - Remove duplicated code

Actually, when I fix something I always search for duplicate. If I found some I put a generic function or comment all code occurrence for duplication (sometime, the effort for putting a generic function doesn't worth it). The main idea, is that I hate doing the same action more than once. Another reason is because there's always someone (could be me) that forget to check for other occurrence...

> - Add unit tests to improve test coverage where coverage is low

Automated unit tests is wonderful... but if you have a big backlog, the task itself is hard to promote unless you have stability issue. Go with the part you are working on and hope that in a few year you have decent coverage.

> - Create consistent formatting across files

IMO the difference in formatting is part of the legacy. It give you an hint about who or when the code was written.

This can gave you some clue about how to behave in that part of the code. Doing the job of reformatting, isn't fun and it doesn't give any value for your customer.

> - Update 3rd party software

Do it only if there's new really nice feature's or the version you have is not supported by the new operating system.

> - Reduce warnings generated by static analysis tools

It can worth it. Sometime warning can hide a potential bug.

Share Improve this answer

Follow

edited Nov 22 at 17:16

TylerH

**21.2k** ● 76 ● 79 ● 110

answered Oct 19, 2008 at 23:31

Hapkido

**1,421** ● 1 ● 9 ● 13

---

6

I'd say 'remove duplicated code' pretty much means you have to pull code out and abstract it so it can be used in multiple places - this, in theory, makes bugs easier to fix because you only have to fix one piece of code, as opposed to many pieces of code, to fix a bug in it.

Share Improve this answer

Follow

answered Sep 28, 2008 at 22:55

James Inman

**1,002** ● 4 ● 17 ● 32

---

6

Add unit tests to improve test coverage. Having good test coverage will allow you to refactor and improve functionality without fear.

There is a good book on this written by the author of CPPUnit, Working Effectively with Legacy Code.

Adding tests to legacy code is certianly more challenging than creating them from scratch. The most useful concept I've taken away from the book is the notion of "seams", which Feathers defines as

> "a place where you can alter behavior in your program without editing in that place."

Sometimes its worth refactoring to create seams that will make future testing easier (or possible in the first place.) The google testing blog has several interesting posts on the subject, mostly revolving around the process of Dependency Injection.

Share Improve this answer

Follow

edited Sep 28, 2008 at 23:48

answered Sep 28, 2008 at 23:06

Gordon Wilson
**26.4k** ● 11 ● 59 ● 61

> Its really difficult to add tests to the code as you end up mocking a lot out before you can even get near the code you want to test! its a bit chicken and egg, can't re-factor without the tests / cant write good tests without re-factoring
> – Craig Angus Sep 28, 2008 at 23:10

**4**

I can relate to this question as I currently have in my lap one of 'those' old school codebase. Its not really legacy but its certainly not followed the trend of the years.

I'll tell you the things I would love to fix in it as they bug me every day:

- Document the input and output variables
- Refactor the variable names so they actually mean something other and some hungarian notation prefix followed by an acronym of three letters with some obscure meaning. CammelCase is the way to go.
- I'm scared to death of changing any code as it will affect hundreds of clients that use the software and someone WILL notice even the most obscure side effect. Any repeatable regression tests would be a blessing since there are zero now.

The rest is really peanuts. These are the main problems with a legacy codebase, they really eat up tons of time.

Share Improve this answer

Follow

answered Sep 28, 2008 at 23:02

Caerbanog

1,335 ● 1 ● 12 ● 18

---

2

I'd say it largely depends on what you want to do with the legacy code...

If it will indefinitely remain in maintenance mode and it's working fine, doing nothing at all is your best bet. "If it ain't broke, don't fix it."

If it's not working fine, removing the unused code and refactoring the duplicate code will make debugging a lot easier. However, I would only make these changes on the erring code.

If you plan on version 2.0, add unit tests and clean up the code you will bring forward

Share Improve this answer

Follow

answered Sep 28, 2008 at 23:19

Austin Salonen

50.2k ● 15 ● 111 ● 140

---

2

Good documentation. As someone who has to maintain and extend legacy code, that is the number one problem. It's difficult, if not downright dangerous to change code you don't understand. Even if you're lucky enough to be handed documented code, how sure are you that the documentation is right? That it covers all of the implicit

knowledge of the original author? That it speaks to all of the "tricks" and edge cases?

Good documentation is what allows those other than the original author to understand, fix, and extend even bad code. I'll take hacked yet well-documented code that I can understand over perfect yet inscrutable code any day of the week.

Share Improve this answer

Follow

answered Sep 29, 2008 at 0:34

Josh Segall

**4,283** ● 4 ● 32 ● 28

> Unit tests ARE the documentation, the problem is that legacy code (as M. Feathers defines it in "Working Effectively with Legacy Code") doesn't have them. – Igor Popov Jul 3, 2010 at 10:14

---

1

The single biggest thing that I've done to the legacy code that I have to work with is to build a real API around it. It's a 1970's style COBOL API that I've built a .NET object model around, so that all the unsafe code is in one place, all of the translation between the API's native data types and .NET data types is in one place, the primary methods return and accept DataSets, and so on.

This was immensely difficult to do right, and there are still some defects in it that I know about. It's not terrifically efficient either, with all the marshalling that goes on. But on the other hand, I can build a DataGridView that round-trips data to a 15-year-old application which persists its

data in Btrieve (!) in about half an hour, and it works. When customers come to me with projects, my estimates are in days and weeks rather than months and years.

Share Improve this answer

Follow

answered Dec 8, 2008 at 0:59

Robert Rossney

96.6k • 24 • 148 • 218

---

1

As a parallel to what Josh Segall said, I would say comment the hell out of it. I've worked on several very large legacy systems that got dumped in my lap, and I found the biggest problem was keeping track of what I already learned about a particular section of code. Once I started placing notes as I go, including "To Do" notes, I stopped re-figuring out what I already figured out. Then I could focus on how those code segments flow and interact.

Share Improve this answer

Follow

answered Dec 8, 2008 at 1:34

dj_segfault

12.4k • 4 • 32 • 37

---

1

I would say just leave it alone for the most part. If it's not broken then don't fix it. If it is broken then go ahead and fix and improve the portion of the code that is broken and its immediately surrounding code. You can use the pain of the bug or sorely missing feature to justify the effort and expense of improving that part.

I would not recommend any wholesale kind of rewrite, refactor, reformat, or putting in of unit tests that is not guided by actual business or end-user need.

If you do get the opportunity to fix something, then do it right (the chance of doing it right the first time might have already passed, but since you are touching that part again might as well do it right time around) and this includes all the items you mentioned.

So in summary, there's no single or just a few things that you should do. You should do it all but in small portions and in an opportunistic manner.

Share Improve this answer

Follow

answered Jul 15, 2011 at 0:37

Omen

88 ● 1 ● 4

---

1

Late to the party, but the following may be worth doing where a function/method is used or referenced often:

- Local variables often tend to be poorly named in legacy code (often owing to their scope expanding when a method is modified, and not being updated to reflect this). Renaming these in line with their actual purpose can help clarify legacy code.
- Even just laying out the method slightly differently can work wonders - for instance, putting all the clauses of an `if` on one line.

- There might be stale/confusing code comments there already. Remove them if they're not needed, or amend them if you absolutely have to. *(Of course, I'm not advocating removal of useful comments, just those that are a hindrance.)*

These might not have the massive headline impact you're looking for, but they are low risk, particularly if the code can't be unit tested.



answered Feb 25, 2013 at 15:25

craigmcnulty

**31** ● 3