When does a code base become large and unwieldy? [closed]

Asked 16 years ago Modified 7 years, 4 months ago Viewed 739 times

3

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 5 years ago.

Improve this question

When do you start to consider a code base to be getting too large and unwieldy?

project-management

Share

Improve this question

Follow

edited Aug 16, 2017 at 8:50

Vadim Kotov

8,274 ● 8 ● 50 ● 63

asked Dec 3, 2008 at 16:01

Stephane Grenier

15.9k ● 41 ● 121 ● 204

12 Answers Sorted by: Highest score (default)

5

-when a significant amount of your coding time is devoted to "where do I put this code?"

-when reasoning about side-effects starts to become really hard.

-when there's a significant amount of code that's just "in there", and nobody knows what it does or if it's still running but it's too scary to remove

-when lots of team members spend significant chunks of their time chasing down intermittent bugs caused by some empty string somewhere in the data where it wasn't expected, or something that you think would usually be caught in a well-written application, in some edge case

-when, in considering how to implement a new feature, "complete rewrite" starts to seem like a good answer

-when you dread looking at the mess of code you need to maintain and wish you could find work building something clean and logical instead of dumpster diving through the detritus of someone else's poorly organized thinking

Share Improve this answer

Follow

answered Dec 3, 2008 at 16:09

Steve B.

57.2k ● 12 ● 97 ● 134

4

When it's over 100 lines. Joke. This is probably the hardest question to answer, because it's very individual.

But if you structure the application well and use different layers for i.e. interfaces, data, services and front-end you will automaticly get a nice "base"-structure. Then you can dividie each layer into different classes and then inside the classes you point out the appropriet methods for the class.

However, there's not an "x amount of lines per method is bad" but think of it more like this, if there is possibility of replication, split it from the current peice and make it re-usable.

Re-using code is the basics of all good structure.

And splitting up into different layers will help the base to become more and more flexible and modular.

Share Improve this answer

Follow

answered Dec 3, 2008 at 16:06

Filip Ekberg

36.3k ● 20 ● 128 ● 184

> The only thing I'd add is "or when it starts to feel that way." After you have enough experience working with good and bad code, you can ascertain the relative health of code pretty easily. Is only as complex as it needs to be? Does it have lots of code smells? Is there a clear design? – Matt Green Dec 3, 2008 at 16:09

2

There exist some calculable metrics if that's what you're searching for. Static code analysis tools can help with that:

Here's one list: http://checkstyle.sourceforge.net/config_metrics.html

Other factors can be the time it takes to change/add something.

Other non-calculable factors can be

- the risk associated to changes
- the level intermingling of features.
- if the documentation can keep up with the features / code
- if the documentation represent the application.
- the level of training needed.
- the quantity of repeat instead of reuse.

Share Improve this answer

Follow

answered Dec 3, 2008 at 16:10

Loki

30.8k ● 9 ● 51 ● 62

I would put the risk of change at #1 too. If you become afraid to add or change features because everything else breaks, then your code has become too unwieldy.
– Joris Timmermans Dec 3, 2008 at 16:19

2

Ah, the god-program anti-pattern.

- When you can't remember at least the outline of sections of it.
- When you have to think about how changes will affect itself or dependencies.
- When you can't remember all the things it's dependant on or depend on it.
- When it takes more than a few minutes(?) to download the source or compile.
- When you have to worry about how to deploy new versions.
- When you encounter classes which are functionally identical to other classes elsewhere in the app.

So many possible signs.

Share Improve this answer

Follow

answered Dec 3, 2008 at 16:11

annakata

75.7k • 18 • 115 • 180

2

I think there are many thoughts to why some code base is too large.

1. It is hard to remain in a constant naming convention. If classes/methods/atributes can't be named consistently or can't be found consistently, then it's time to reorganize.

2. When your programmers are surfing the web and going to lunch in order to compile. Keeping compiling/linking time to a minimum is important for management. The last thing you want is a programmer to get distracted by twiddling their thumbs for too long.
3. When small changes start to affect many MANY other places of code. There is a benefit to consolidation of code, but there is also a cost. If a small change to fix one bug causes a dozen more, and this is commonly happens, then your code base needs to be spread out (versioned libraries) or possibly unconsolidated (yes, duplicate code).
4. If the learning curve of new programmers to the project is obviously longer than acceptable (usually 90 days), then your code base/training isn't set up right.

..There are many many more, I'm sure. If you think about it from these three perspectives:

1. Is it hard to support?
2. Is it hard to change?
3. Is it hard to learn?

...Then you will have an idea if your code fits the "large and unwieldy" category

answered Dec 3, 2008 at 16:12

Jeremiah
5,516 • 9 • 40 • 46

2

For me, code becomes unwieldy when there's been a lot of changes made to the codebase that weren't planned for when the program was initially written or last refactored significantly. At this point, stuff starts to get fitted into the existing codebase in odd places for expediency and you start to get a lot of design artifacts that only make sense if you know the history of the implementation.

Share Improve this answer

Follow

answered Dec 4, 2008 at 2:51

dsimcha
68.6k • 55 • 219 • 340

1

Short answer: it depends on the project.

Long answer:

A codebase doesn't have to be large to be unwieldy - spaghetti code can be written from line 1. So, there's not really a magic tripping point from good to bad - it's more of a spectrum of great <---> awful, and it takes daily effort to keep your codebase from heading in the wrong direction. What you generally need is a lead developer that has the ability to review others' code objectively, and keep an eye on the architecture and design of the code as a whole - no one line developer can do that.

Share Improve this answer

Follow

answered Dec 3, 2008 at 16:09

Greg Hurlman

17.8k • 6 • 55 • 87

1

When I can't remember what a class does or what other classes it uses off the top of my head. It's really more a function of my cognitive capacity coupled with the code complexity.

Share Improve this answer

Follow

answered Dec 3, 2008 at 16:10

tvanfosson

532k • 102 • 699 • 798

1

I was trying to think of a way of deciding based on how your collegues perceive it to be.

During my first week at a gig a few years ago, I said during a stand-up that I had been tracking a white rabbit around the ContainerManagerBean, the ContainerManagementBean and the ContextManagerBean (it makes me shudder just recalling these words!). At least two of the developers looked at their shoes and I could see them keeping in a snigger.

Right then and there, I knew that this was not a problem with my lack of familiarity with the codebase - all the developers perceived a problem with it.

Share Improve this answer

Follow

answered Dec 3, 2008 at 18:26

jamesh
20.1k ● 14 ● 58 ● 96

0

If over years of development different people code change requests and bug fixes you will sooner or later get parts of code with duplicated functionality, very similar classes, some spaghetti etc. This is mostly due to the fact that a fix is needed fast and the "new guy" doesn't know the code base. So he happily codes away something which is already there.

But if you have automatic checks in place checking the style, unit test code coverage and similar you can avoid some of it.

Share Improve this answer

Follow

answered Dec 3, 2008 at 16:11

Drejc
14.3k ● 16 ● 75 ● 108

Even better than automatic checks is having code reviews - because they will actually *teach* the new guys how to do it right, instead of correcting them after the fact.
– Joris Timmermans Dec 3, 2008 at 16:37

Completly agree but sometimes there is no one to teach and tutor, or managers don't allow it because of time/money limits. Have seen it all. – Drejc Dec 3, 2008 at 19:28

0

A lot of the things that people have identified as indicating problems don't really have to do with the raw size of the codebase, but rather its comprehensibility. How does size relate to comprehensibility? If at all...

I've seen very short programs that are just a mess -- easier to throw away and redo from scratch. I've also seen very large programs whose structure is transparent enough that it is comprehensible even at progressively more detailed views of it. And everything in between...

I think look at this question from the standpoint of an entire codebase is a good one, but it probably pays to work up from the bottom and look first at the comprehensibility of individual classes, to multi-class components, to subsystems, and finally up to an entire system. I would expect the answers at each level of detail to build on each other.

For my money, the simplest benchmark is this: Can you explain the essence of what X does in one sentence? Where X is some granularity of component, and you can assume an understanding of the levels immediately above and below the component.

Share Improve this answer

Follow

answered Dec 3, 2008 at 18:37

Jeff Kotula

2,134 ● 12 ● 17

0

- When you come to need a utility method or class, and have no idea whether someone else has already implemented it or have any idea where to look for one.
- Related: when several slightly different implementations of the same functionality exist, because each author was unaware of other authors' work.

Share Improve this answer

Follow

answered Dec 3, 2008 at 18:49

Dan Vinton

26.8k ● 9 ● 41 ● 82