What is self-documenting code and can it replace well documented code? [closed]

Asked 16 years, 2 months ago Modified 2 years, 10 months ago

Viewed 83k times

277

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, visit the help center for guidance.

Closed 12 years ago.

I have a colleague who insists that his code doesn't need comments, it's "self documenting."

I've reviewed his code, and while it's clearer than code which I've seen others produce, I still disagree that self-documenting code is as complete and useful as well commented and documented code.

Help me understand **his** point of view.

- What is self documenting code

- Can it really replace well commented and documented code
- Are there situations where it's better than well documented and commented code
- Are there examples where code cannot possibly be self-documenting without comments

Maybe it's just my own limitations, but I don't see how it can be a good practice.

This is not meant to be an argument - please don't bring up reasons why well commented and documented code is high priority - there are many resources showing this, but they aren't convincing to my peer. I believe I need to more fully understand his perspective to convince him otherwise. Start a new question if you must, but don't argue here.

Also, those of you who are arguing against self documenting code -this is primarily to help me understand the perspective (ie, positive aspects) of self-documenting code evangelists.

documentation **comments**

edited Feb 3, 2022 at 8:27

community wiki

7 revs, 5 users 74%
Adam Davis

126 You know what really impresses me? You disagree with the guy but you're asking to understand him, not for more ammunition against him. – kajaco Dec 8, 2008 at 0:50

19 As a story of an extremely opposite case, I have a co-worker who writes tons of documentation: in every cpp file she includes a *manual* with at least a couple dozen pages on the implementation and usage of the functions provided. However, she writes disastrously long and complicated functions (single functions with 8000 lines of code), counter-intuitive identifiers for variables and functions, etc. Compared to her, I'd take someone striving to write self-documenting code that slacks on the comments any day, provided his code is well-organized with small functions that are easy to understand. – stinky472 Jul 8, 2010 at 1:08

Related: thedailywtf.com/Articles/… – Calmarius Oct 3, 2012 at 9:48

2 In short, one can avoid most comments that explain **how** the code works and make the code self-documenting in that respect. But it is often also required to explain **why** the code works the way it does, e.g. when you work around a limitation in a library. You usually need comments to explain the why. – Lutz Prechelt Feb 1, 2015 at 15:36

2 I used to work with someone who over-commented everything, but typically with useless comments, like `i++; // increment i` - but with no explanation about *why* `i` should be incremented at that point in the function.
– nnnnnn Jul 18, 2016 at 5:53

1

I think its a matter of the right amount of documentation, rather than all or none. If the parameters to a function are well named, you often don't have to say exactly what they are, e.g. char *CustomerName is pretty obvious. If you use assert value ranges for parameters, you don't have to document those ranges as well. IMO, documentation should cover everything which is less than obvious and hence needs some explanation, and most code needs some documentation. Personally, I'd rather see an illustrative example of how a given function works than descriptive documentation, in most cases.

Documentation for documentations sake can be a waste of time, as the documentation needs maintenance to be kept up to date with the code base. If no one will benefit from reading it, don't produce it.

Share Improve this answer Follow

answered Oct 16, 2008 at 15:34

community wiki
SmacL

I'd turn this around.

1

Ask yourself what you don't understand in his code and then ask him to document those. And maybe you could also tell us some.

Share Improve this answer Follow

answered Oct 16, 2008 at 15:35

community wiki
iny

1

This is an excellent question. It traces back to the first programming language that allowed comments, I'm sure. The code certainly should be as self-documenting as possible. Comments that point out the obvious, should be eliminated. Comments that make it easier to understand the intent, purpose, and use of a given method or section of code can be invaluable to those of us dolts that may be less familiar with the language or code in question. Structured comments that allow for the generation of API documentation are a good example. Just don't comment an IF statement that checks to see if a checkbox is checked and tell me that you're checking to see if the checkbox is checked. Restating the obvious in a comment is the worst waste keystrokes in our universe.

```
//For example, the above text deals with what is a
useful comment
```

answered Oct 16, 2008 at 15:38

Share Improve this answer

Follow

community wiki
Tyler Jensen

1

Self documenting code is code that is trivially easy to understand. Variable naming goes a long way to making code self documenting, but i find the best tactic is to break any complicated logic down into tiny little chunks and refactor that information into seperate methods with verbose and informative names. Then your complicated methods become simply a list of steps to be performed. The tiny private helper methods then are documented sufficiently by their own method name and the complicated methods are documented as a sequence of abstract steps to be performed. In practice this strategy cannot always be applied perfectly so comments are still very useful. Plus you should never completely abandon any tool which will help you write code that is easier to understand.

Share Improve this answer

Follow

answered Oct 16, 2008 at 15:51

community wiki
Spencer Stejskal

If your code isn't completely clear without comments, then there's room to improve your code.

1

I'm not saying "don't comment unclear code". I'm saying "make your code clear".

If you end up leaving your code unclear in some way, *then* use comments to compensate.

Share Improve this answer Follow

answered Oct 16, 2008 at 20:38

community wiki
Jay Bazuzi

1

Self documenting code is silliness. Anyone who's had to re-visit their code after weeks, months or *gasp* years knows that (or days, in my case). (Perhaps the guy who is promoting this idea is still wet behind the ears!?!?!)

Use meaningful, descriptive data names, factor your code intelligently, and leave yourself hints as to why the heck you did what you did and you will live a richer, more fullfilling life.

Although...I did read a quote once attributed to Bill Gates: "The code IS the documentation."

Go figure.

Share Improve this answer Follow

answered Oct 16, 2008 at 21:38

community wiki
Chuck Wright

I disagree on the "Self documenting code is silliness thing". "Use meaningful, descriptive data names, factor your code intelligently" - Is part of self documenting code. – AshtonKJ Oct 17, 2008 at 11:08

True, but that wasn't my point, which is wrapped up in the "and leave yourself hints as to why the heck you did what you did" comment (no pun intended). – Chuck Wright Oct 17, 2008 at 14:22

1

Some perspectives from the non-commenting camp.

"well commented" (verbose) code is harder to read and understand. For one thing, there is simply more text to scan. It increases the cognitive effort in understanding a CodeBase - the nonfunctional text takes up screen space that could be used to show code.

Another big problem with comments is that they are unreliable - especially on older code bases, comment rot sets in faster than bit rot.

And then of course there is the effort involved in writing comments. With the exception of the occasional one line clarifier, every time I start commenting code I get one of two guilty feelings

1. this info needs to go in overall supporting documentation

2. I need to clean up my code

Share Improve this answer Follow

answered Oct 17, 2008 at 0:46

community wiki
Scott Weinstein

1

Regardless of purely self-documenting code is achievable, there are some things that come to mind one should do anyway:

- Never have code that is "surprising". Ie. don't use silly macro's to redefine things etc. Don't misuse operator overloading, don't try to be smart on this.
- Split away code at the right point. Use proper abstractions. Instead of inlining a rolling buffer (a buffer with fixed length, with two pointers that gets items added at one end and removed at the other), use an abstraction with a proper name.
- Keep function complexity low. If it gets too long or complex, try to split it out into other other functions.

When implementing specific complex algorithms, add documentation (or a link to) describing the algorithm. But in this case try to be extra diligent in removing unneeded complexity and increasing legibility, as it is too easy to make mistakes.

Share Improve this answer Follow

answered Oct 17, 2008 at 8:07

community wiki
Paul de Vrieze

1

Very mixed inputs here it seems :)

I use the Pseudo code Programming Process for new developments, which virtually makes my code self documenting. I start to write Pseudo code only when writing new code then extend on it. Im not saying this is best practice or anything like that, i'm just highlighting one technique I find useful if you know you want a lot of documentation for your code if its going to a third party, reviewer, etc... it also occasionally highlights problems for me before ive even written a line of code.

```
' check database is available
  ' if it is then allow the procedure
  ' if it isnt roll back and tidy up
' move onto something else
```

becomes;

```
' check database is available
  if checkDBStateResult(currentDB) = Open then
     ' if it is then allow the procedure
         proc.Ok = True
  else
     ' if it isnt roll back
         proc.Ok = False
```

```
            CleanUp()
    end if
```

Share Improve this answer Follow

answered Oct 17, 2008 at 8:39

community wiki
CaRDiaK

1

I once worked with a guy who was about to sell a financial suite to a large company. They insisted he document the source code, to which he produced a 30+ page assembler routine and said 'it is documented, look' - then he flipped to page 13 and there was a comment 'bump counter by one'. Great product, great implementor, but...

Anyway to my mind the inportant comments above are to set the context. This snippet was stated as self-documented:

```
> from BeautifulSoup import
> BeautifulSoup, Tag def
> replace_a_href_with_span(soup):
>     links = soup.findAll("a")
>     for link in links:
>         tag = Tag(soup, "span", [("class",
"looksLikeLink")])
>         tag.contents = link.contents
>         link.replaceWith(tag)
```

But, I for one, need a context to understand it fully.

Share Improve this answer Follow

answered Nov 20, 2008 at 21:14

community wiki
jsfain

1

The point has been made that comments should capture intent, but I would go a little further.

I think for any class of problems, there is an ideal (or nearly so) vocabulary and syntax to describe it, and you can see it if you just ask the person who has such problems to describe them (assuming that person can think clearly).

If that vocabulary and syntax maps easily (by defining classes, methods, etc.) onto code in a computer language, then that code can be self-documenting. Also, IMO, a domain-specific language has been created. (And that is my rough-hewn definition of "declarative".)

Failing this ideal, if the problem does not map so directly onto the computer code, then you need something to link the two together. IMO, that is the purpose of comments.

That way, when the problem changes, you can find the corresponding parts of the code to change.

EDIT: I am not, by the way, arguing in favor of the OOP methodology where every noun becomes a class and

every verb a method. I've seen enough bloatware built **that** way.

Share Improve this answer Follow

edited Dec 11, 2008 at 3:15

community wiki
2 revs
Mike Dunlavey

1

Good design structure helps to point out that some functions are for general use and some for random business logic even though you don't have a comment saying "this function is general".

We should not forget about design and specification documentation though. Those have or at least should have much of the texts that are not necessarily needed in comments. Software often have also user manuals and other description documents, and those should be in sync with what the program does. The situation is not great if the user has to find out what the software does from the source code instead of a manual. So self documenting code still doesn't mean that the actual software has been documented.

Think about traceability of features, too. When you have your manual, then you should be able to trace the features to source code and back for better maintainability. Manuals and specifications are not that much about programming, but they are about software

engineering. The bigger the software, the more engineering is needed.

Share Improve this answer Follow

answered Aug 12, 2009 at 8:18

community wiki
Silvercode

1

Here are my best answers to your questions.

Self documenting code is code that's written clearly with class, method, function, and variable names that make it's intent and function easily understood. If done well, it is the documentation.

It *can* replace code that's well commented and documented, but I've rarely seen it. Too many times, programmers believe that they are good enough to do this, but the best way to knock them down is to start asking questions. If they have to start explaining too much, then their code wasn't clear enough. You shouldn't have to read the code to know what it does.

There may be some situations where it's better. If the code is small and simple, then you may clutter things up by adding documentation.

Code that includes an algorithm should contain comments. Most times, even the original programmers

can't remember what the heck they were thinking a few months ago when they wrote a long function.

Share Improve this answer Follow

edited Oct 10, 2013 at 17:46

community wiki
2 revs
wandercoder

0

Self documented code is code that is so clear that a comment would be unnecessary. I'll give a small example:

```
//iterate from 0 to 100
for(int i=0; i < 100; i++) {
   println i
}
```

The comment is pretty useless, because the code is clear. Documentation is a good practice, but extra documentation can add unecessary noise to the code. What your colleague needs to know is that not everyone can read other's code and acknowledge all it's details.

```
int calc(int a, int b) {
   return sqrt(a*a + b*b); //pythagoras theorem
}
```

The last snippet would be extra hard to decipher without the comment. You can imagine other examples that are

more contrived.

Share Improve this answer Follow

answered Oct 16, 2008 at 15:35

community wiki
Miguel Ping

I think you mean "Pythagorean" theorem? Why not just name the function that? – ine Oct 16, 2008 at 15:47

That first comment is not only redundant, it can also be argued that it is wrong because it does not state the side-effect that something is getting printed - even though this particular example is rather contrived. – Chris Vest Oct 16, 2008 at 16:31

The first comment is also wrong since it iterates from 0 to 99. A fate too many comments experiences. – Berserk Oct 16, 2008 at 16:52

The second example should be something like "int calcHypotenuseLength(int side1, int side2)" or "int pythagoreanHypotenuseLength(...)". Self-documenting. – ColinD Oct 16, 2008 at 17:17

calcHypotenuseLength conveys the algorithm. This violates information hiding. Let the implementation keep track of that, maybe a method called vektorSum(int xVektor, int yVektor) if the mathematical properties are important, or something closer to the domain like distance(int pointX, int pointY) – John Nilsson Oct 16, 2008 at 19:25