What are your "hard rules" about commenting your code? [closed]

Asked 16 years, 2 months ago Modified 3 years, 6 months ago

Viewed 4k times

11

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 2 years ago.

Improve this question

I have seen the other questions *but I am still not satisfied with the way this subject is covered*.

I would like to extract a distiled list of things to check on comments at a code inspection.

I am sure people will say things that will just cancel each other. But hey, maybe we can build a list for each camp. For those who don't comment at all the list will just be very short :)

coding-style comments

Share

Improve this question

Follow

edited Jan 16, 2018 at 0:37

en casa
135 ● 2 ● 10

asked Sep 27, 2008 at 2:58

alfinoba
723 ● 1 ● 6 ● 14

21 Answers

Sorted by: Highest score (default)

36

I have one simple rule about commenting: Your code should tell the story of what you are doing; your comments should tell the story of why you are doing it.

This way, I make sure that whoever inherits my code will be able to understand the intent behind the code.

Share Improve this answer

Follow

answered Sep 27, 2008 at 3:19

Franci Penov

75.9k ● 18 ● 135 ● 171

1 If you can't tell what the intention is, it isn't commented well enough. Totally agree. – Jeff Yates Oct 1, 2008 at 21:26

Agreed, if you start writing a comment there is a good chance your code isn't clear enough. Refactor first and then if it's still too complex then comment. – David Hayes Oct 1, 2008 at 21:31

16

1. I comment public or protected functions with meta-comments, and usually hit the private functions if I remember.
2. I comment why any sufficiently complex code block exists (judgment call). The *why* is the important part.
3. I comment if I write code that I think is not optimal but I leave it in because I cannot figure out a smarter way or I know I will be refactoring later.
4. I comment to remind myself or others of missing functionality or upcoming requirements code not

present in the code (TODO, etc).

5. I comment to explain complex business rules related to a class or chunk of code. I have been known to write several paragraphs to make sure the next guy/gal knows why I wrote a hundred line class.

Share Improve this answer Follow

edited Apr 25, 2016 at 9:24
jotik
17.9k ● 16 ● 65 ● 128

answered Sep 27, 2008 at 3:13
Jason Jackson
17.2k ● 8 ● 51 ● 75

12

If a comment is out of date (does not match the code), delete it or update it. Never leave an inaccurate comment in place.

Share Improve this answer Follow

answered Sep 27, 2008 at 3:00
Jason Etheridge
6,897 ● 5 ● 31 ● 33

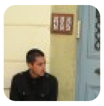
6

> Documentation is like sex; when it's good, it's very, very good, and when it's bad, it's better than nothing

Share Improve this answer

answered Sep 27, 2008 at 3:01

Follow


Pablo Fernandez
105k ● 58 ● 195 ● 233

4 And when it's wrong, you'd wish you'd never tried in the first place? – Rob Howard Nov 4, 2008 at 0:40

6

Write readable code that is self-explanatory as much as possible. Add comments whenever you have to write code that is too complex to understand at a glance. Also add comments to describe the business purpose behind code that you write, to make it easier to maintain/refactor it in the future.

Share Improve this answer

Follow

answered Sep 27, 2008 at 3:02

JoshL
11k ● 11 ● 58 ● 62

5

The comments you write can be revealing about the quality of your code. Countless times I've removed comments in my code to replace them with better, clearer code. For this I follow a couple of anti-commenting rules:

- If your comment merely explains a line of code, you should either let that line of code speak for itself or split it up into simpler components.
- If your comment explains a block of code within a function, you should probably be explaining a new function instead.

Those are really the same rule repeated for two different contexts.

The other, more normal rules I follow are:

- When using a dynamically-typed language, document the expectations that important functions make about their arguments, as well as the expectations callers can make about the return values. Important functions are those that will ever have non-local callers.
- When your logic is dictated by the behavior of another component, it's good to document what your understanding and expectations of that component are.

Share Improve this answer Follow

answered Sep 27, 2008 at 3:11

Logan
1,924 ● 1 ● 12 ● 11

4

When implementing an RFC or other protocol specification, comment state machines / event handlers / etc with the section of the spec they correspond to. Make sure to list the version or date of the spec, in case it is revised later.

Share Improve this answer Follow

answered Sep 27, 2008 at 3:04

DGentry
16.3k ● 8 ● 53 ● 66

3

I usually comment a method before I write it. I'll write a line or two of comments for each step I need to take within the function, and then I write the code between the comments. When I'm done, the code is already commented.

The great part about that is that it's commented *before* I write the code, so there are not unreasonable assumptions about previous knowledge in the comments; I, myself, knew nothing about my code when I wrote them. This means that they tend to be easy to understand, as they should be.

Share Improve this answer Follow

answered Sep 27, 2008 at 3:54

Lucas Oman
15.9k ● 2 ● 46 ● 45

Yup and it is an extra sanity check in the process.
– Toon Krijthe Sep 27, 2008 at 11:24

3

There are no hard rules - hard rules lead to dogma and people generally follow dogma when they're not smart enough to think for themselves.

The **guidelines** I follow:

1/ Comments tell what is being done, code tells how it's being done - don't duplicate your effort.

2/ Comments should refer to blocks of code, not each line. That includes comments that explain whole files, whole functions or just a complicated snippet of code.

3/ If I think I'd come back in a year and not understand the code/comment combination then my comments aren't good enough yet.

Share Improve this answer Follow

answered Sep 27, 2008 at 4:17

paxdiablo
880k ● 241 ● 1.6k ● 2k

2

A great rule for comments: if you're reading through code trying to figure something out, and a comment somewhere would have given you the answer, *put it there when you know the answer*.

Only spend that time investigating *once*.

Eventually you will know *as you write* the places that you need to leave guidance, and the places that are sufficiently obvious to stand alone. Until then, you'll spend time trawling through your code trying to figure out why you did something :)

Share Improve this answer Follow

answered Sep 27, 2008 at 6:28

Rich
2,893 ● 1 ● 20 ● 20

1

I document every class, every function, every variable within a class. Simple DocBlocks are the way forward.

I'll generally write these docblocks more for automated API documentation than anything else...

For example, the first section of one of my PHP classes

```
/**
 * Class to clean variables
 *
 * @package     Majyk
 * @author      Martin Meredith
<martin@sourceguru.net>
 * @licence     GPL (v2 or later)
 * @copyright   Copyright (c) 2008 Martin Meredith
<martin@sourceguru.net>
 * @version     0.1
 */
class Majyk_Filter
{
    /**
     * Class Constants for Cleaning Types
     */
    const Integer            = 1;
    const PositiveInteger    = 2;
    const String             = 3;
    const NoHTML             = 4;
    const DBEscapeString     = 5;
    const NotNegativeInteger = 6;

    /**
     * Do the cleaning
     *
     * @param   integer Type of Cleaning (as
defined by constants)
     * @param   mixed   Value to be cleaned
     *
     * @return  mixed   Cleaned Variable
```

```
      *
      */
```

But then, I'll also sometimes document significant code (from my init.php

```
// Register the Auto-Loader
spl_autoload_register("majyk_autoload");

// Add an Exception Handler.
set_exception_handler(array('Majyk_ExceptionHandler'
'handle_exception'));

// Turn Errors into Exceptions
set_error_handler(array('Majyk_ExceptionHandler',
'error_to_exception'), E_ALL);

// Add the generic Auto-Loader to the auto-loader
stack
spl_autoload_register("spl_autoload");
```

And, if it's not self explanatory why something does something in a certain way, I'll comment that

Share Improve this answer Follow

answered Sep 27, 2008 at 3:22

Mez
24.9k ● 14 ● 74 ● 93

Okay but isn't it redundant? // Add an Exception Handler. set_exception_handler(array('Majyk_ExceptionHandler', 'handle_exception')); – Daniel Sep 27, 2008 at 3:28

Quite possibly.... :D but more in the fact that the comment needs more expanding "Add global exception handler for all unhandled but thrown errors, or Exceptions bubbled up to the top level" – Mez Sep 29, 2008 at 7:24

1

The only guaranteed place I leave comments: **TODO** sections. The best place to keep track of things that need reworking is right there in the code.

Share Improve this answer Follow

answered Sep 27, 2008 at 6:31

Rich
2,893 ● 1 ● 20 ● 20

1

I create a comment block at the beginning of my code, listing the purpose of the program, the date it was created, any license/copyright info (like GPL), and the version history.

I often comment my imports if it's not obvious why they are being imported, especially if the overall program doesn't appear to need the imports.

I add a docstring to each class, method, or function, describing what the purpose of that block is and any additional information I think is necessary.

I usually have a demarcation line for sections that are related, e.g. widget creation, variables, etc. Since I use SPE for my programming environment, it automatically highlights these sections, making navigation easier.

I add TODO comments as reminders while I'm coding. It's a good way to remind myself to refactor the code once it's verified to work correctly.

Finally, I comment individual lines that may need some clarification or otherwise need some metadata for myself in the future or other programmers.

Personally, I hate looking at code and trying to figure out what it's supposed to do. If someone could just write a simple sentence to explain it, life is easier. Self-documenting code is a misnomer, in my book.

Share Improve this answer Follow

edited Sep 27, 2008 at 6:59

answered Sep 27, 2008 at 6:51

crystalattice
5,089 ● 12 ● 43 ● 57

1

I focus on the **why**. Because the **what** is often easy readable. TODO's are also great, they save a lot of time.

And i document interfaces (for example file formats).

Share Improve this answer Follow

answered Sep 27, 2008 at 11:27

Toon Krijthe
53.4k ● 38 ● 149 ● 202

A really important thing to check for when you are checking header documentation (or whatever you call the

1

block preceding the method declaration) is that directives and caveats are easy to spot.

Directives are any "do" or "don't do" instructions that affect the client: don't call from the UI thread, don't use in performance critical code, call X before Y, release return value after use, etc.

Caveats are anything that could be a nasty surprise: remaining action items, known assumptions and limitations, etc.

When you focus on a method that you are writing and inspecting, you'll see everything. When a programmer is using your method and thirty others in an hour, you can't count on a thorough read. I can send you research data on that if you're interested.

Share Improve this answer Follow

answered Oct 1, 2008 at 21:27

Uri

89.6k ● 51 ● 226 ● 322

0

Pre-ambles only; state a class's Single Responsibility, any notes or comments, and change log. As for methods, if any method needs substantial commenting, it is time to refactor.

Share Improve this answer Follow

answered Sep 27, 2008 at 3:16

moffdub

5,314 ● 2 ● 37 ● 32

0

When you're writing comments, stop, reflect and ask yourself if you can change the code so that the comments aren't needed. Could you change some variable, class or method names to make things clearer? Would some `assert`s or other error checks codify your intentions or expectations? Could you split some long sections of code into clearly named methods or functions? Comments are often a reflection of our inability to write (a-hem, code) clearly. It's not always easy to write clearly with computer languages but take some time to try... because code never lies.

P.S. The fact that you use quotes around "hard rules" is telling. Rules that aren't enforced aren't "hard rules" and the only rules that are enforced are in code.

Share Improve this answer Follow

answered Sep 27, 2008 at 3:24

Pat Notz

214k ● 31 ● 94 ● 92

0

I add 1 comment to a block of code that summarizes what I am doing. This helps people who are looking for specific functionality or section of code.

I comment any complex algorithm, or process, that can't be figured out at first glance.

I sign my code.

Share Improve this answer

answered Sep 27, 2008 at 4:34

Follow

J.J.
4,892 ● 1 ● 26 ● 29

0

In my opinion, TODO/TBD/FIXME etc. are ok to have in code which is currently being worked on, but when you see code which hasn't been touched in 5 years and is full of them, you realize that it's a pretty lousy way of making sure that things get fixed. In short, **TODO notes in comments tend to stay there**. Better to use a bugtracker if you have things which need to be fixed at some point.

Hudson (CI server) has a great plugin which scans for TODOs and notes how many there are in your code. You can even set thresholds causing the build to be classified as unstable if there are too many of them.

My favorite rule-of-thumb regarding comments is: **if the code and the comments disagree, then both are likely incorrect**

Share Improve this answer

Follow

answered Oct 1, 2008 at 21:21

JesperE
64.3k ● 22 ● 142 ● 199

Because bug reports are more expensive than todos, there's research showing that people will memorize instead of going for a full blown bug report. They also want to avoid making their todos public, in some cases. I use a tool that highlights calls to methods with todos; easier to catch. – Uri Oct 1, 2008 at 21:29

Well, then you could call it a lightweight issue tracker. The problem is when people just add a todo, and then forget about it. – JesperE Oct 1, 2008 at 22:16

0

We wrote an article on comments (actually, I've done several) here: http://agileinaflash.blogspot.com/2009/04/rules-for-commenting.html

It's really simple: Comments are written to tell you what the code cannot.

This results in a simple process: - Write any comment you want at first. - Improve the code so that the comment becomes redundant - Delete the now-redundant comment. - Only commit code that has no redundant comments

Share Improve this answer Follow

answered Nov 17, 2011 at 22:06

Tim Ottinger
1,452 ● 9 ● 5

0

I'm writing a Medium article in which I will present this rule: when you commit changes to a repository, each comment must be one of these three types:

- A license header at the top
- A documentation comment (e.g., Javadoc), or
- A `TODO` comment.

The last type should not be permanent. Either the thing gets done and the `TODO` comment is deleted, or we decide the task is not necessary and the `TODO` comment gets deleted.

Share Improve this answer Follow

answered Jun 22, 2021 at 20:18

Alonso del Arte

1,025 ● 1 ● 9 ● 20