Commenting code

Beyond common wisdom

PyCon DE and PyData

Stefan Schwarzer, SSchwarzer.com
info@sschwarzer.com

Berlin, Germany, 2019-10-10

About me

- Using Python since 1999
- Full-time software developer since 2000
- Freelancer since 2005
- Book “Workshop Python”, Addison-Wesley,
  using the then brand new Python 2.2 ;-)
- About 15 conference talks
- Maintainer of ftputil (high-level FTP client library)
  since 2002

Overview

- Introduction
- Common wisdom
- Comment more – or less?
- Tips for good comments
- Comments vs. commit messages

Introduction

motivation, purposes of comments

Motivation for this talk

- Very interested in writing maintainable software
- Writing rather many, sometimes extensive comments
- Found online articles or statements that recommend using few or even no comments

How to resolve this contradiction?

Why comments?

Comments help with

- Understanding the software better
  $\rightarrow$ make changes more safely
- Understanding the software faster
  $\rightarrow$ make changes faster – but still correctly!

(even if the software design is already quite good)

Common wisdom

why vs. what/how, extract functions, better identifier names, comments become obsolete, comments deviate from code

“Document the ‘why’, not the ‘what/how’ ”

- Mostly yes
- “Section comments” can help, too
- Idea: describe a code section on a higher abstraction level, even if you could derive the same information from reading the code

“Document the ‘why’, not the ‘what/how’”

Example of a section comment

```
# Classify the object or its descriptor.
if isinstance(dict_obj, (staticmethod, types.BuiltinMethodType)):
    kind = "static method"
    obj = dict_obj
elif isinstance(dict_obj, (classmethod,
                           types.ClassMethodDescriptorType)):
    kind = "class method"
    obj = dict_obj
elif isinstance(dict_obj, property):
    kind = "property"
    obj = dict_obj
elif isroutine(obj):
    kind = "method"
else:
    kind = "data"
```

(from Python standard library, `inspect` module)

“Extract a function/method instead of writing comments”

Pro:

- Individual function is shorter, probably easier to read
- Extraction introduces new (hopefully meaningful) names for function/method and parameters

Con:

- Code becomes more fragmented
- Overall understanding doesn't necessarily get better
- Reader has to find out from where the extracted function/method is called

“Extract a function/method instead of writing comments”

```
class MyClass:

    ...

    # Is this called from anywhere else in the class?
    # Maybe from derived classes?
    def _new_method(self, foo, bar):
        ...
        return result

    def original_method(self):
        ...
        foo = ...
        ...
        bar = ...
        return self._new_method(foo, bar)

    ...
```

“Use better names instead of writing comments”

aka “Use a better name instead of commenting a bad name”

- Why not use a better name and comment?
- Improve design, including better names
- If something still needs clarification, add comments

“Comments easily become obsolete”

- Take comments as seriously as other code
  → Pay attention to updating comments
- “I can’t test comments”
  Does that mean you don’t take code seriously only because you can’t test it? ;-)
- Comments are more likely to become obsolete the further they are away from the code they apply to
  → If you can, prefer comments in functions/methods over comments on the class/module level

“Avoid comments because they could deviate from the code”

aka “Contradictory comments are worse than no comments”

- Pay attention to updating comments (see above)
- Since the comment should be on a higher abstraction level, it’s likely that the code is wrong and the comment tells you what was intended
- Sometimes deviations tell you something about design problems in your code
- Usually only a small fraction of comments ends up contradictory, so overall you’re better off with comments :-)

Comment more – or less?

don't repeat the code, development context,
developer knowledge, recommendation

Don't repeat the code

Do not comment when it would repeat the code
on the same abstraction level

```
# Increase index by 2.
index += 2

# Add client to clients list.
clients.append(client)
```

This is ok (higher abstraction level)

```
# Never expire
max_age = None
```

Development context

Reasonable amount and detailedness of comments depend on development context

- Developer knowledge
- Software size
- Software complexity
- Application vs. library
- Team size
- Team fluctuation
- Time between modifications
- ...

Developer knowledge

There are many ways a developer can be more or less experienced/knowledgeable

- General programming knowledge
- Programming language knowledge
- Problem domain knowledge
- Company knowledge
- Architecture knowledge
- Perhaps more?

Do pair programming and code reviews for knowledge transfer!

More or less?

- (If you really want) you can ignore existing comments
- But you can't conjure up comments which aren't there :-)

→ If in doubt, comment rather more than less!

Good comments

(in my opinion)

Tips for good comments

- Comments should make the code easier to understand
- Comment what a reader might want to know
- Comment what you found out while working on the code
- Comment what isn't obvious. Besides, what's obvious to you may not be obvious to others ;-)
- Check comments for ambiguity (pronouns, singular/plural, sentence structure, . . .)
- Comment to avoid "simplifications" that could break the code

Tips for good comments

Example for avoiding “simplification”

```
def _dir(self, path):
    """
    Return a directory listing as made by FTP's `LIST` command
    as a list of strings.
    """
    # Don't use `self.path.isdir` in this method because that
    # would cause a call of `(l)stat` and thus a call to `_dir`,
    # so we would end up with an infinite recursion.
    ...
```

(from ftputil library, `host` module)

Tips for good comments

- Start comments with an uppercase letter. End sentences with a period. This makes it easier to extend comments.
- Use XXX, TODO and FIXME comments. Don't pretend everything is fine when you know it's not the case.
- If necessary, adjust syntax highlighting so that comments are easy to read. This helps against comments getting outdated.
- If you add an empty line to separate code sections, consider adding a comment on the section instead :-)

Comments vs. commit messages

Comments vs. commit messages

- Comments are far more visible than commit messages
  $\rightarrow$ Comment what a reader must not miss
- Explain in commit messages why you did not do something (for example designs you considered and why you chose one over the others)
- Put information in commit messages a code reviewer might want to know (unless already covered in comments)

Literature

Literature

- *The art of readable code*
  Dustin Boswell, Trevor Foucher
- *A philosophy of software design*
  John Ousterhout
  see also his talk at
  https://www.youtube.com/watch?v=bmSAYIu0NcY
- *How to write a Git commit message*
  Chris Beams
  https://chris.beams.io/posts/git-commit
  (also applies to other VCSs)
- *Line by line – how to edit your own writing*
  Claire Kehrwald Cook

Thank you for your attention! :-)

Questions?

Remarks?

Discussion?

info@sschwarzer.com

https://sschwarzer.com