Comments are in theory supposed to be a good thing, but in practice they are often either abused or unused. **Kevlin Henney** offers comment on comments

Prefer code to comments

YOUR CODE READING SPEED IS AFFECTED by many factors, some of which depend on you, some of which depend on the code. For instance, your ability to parse and grasp code will be lower in the post-lunch siesta part of the afternoon than during the up-'n'-at-'em, caffeine-injected start of the day. You can grasp code more rapidly and accurately when it is based on familiar patterns rather than someone else's programming idiolect. Reading through long-winded or poorly constructed logic and control flow is more akin to wading through molasses than cutting through butter. However, super-clever code trickery or cryptic conventions keep you guessing. And then there are comments.

Comments *should* improve the readability of the code, but so often, the opposite is true. As with so many *shoulds* in life, it takes more than just saying it to get it right. Comments are an odd form of code: they are code that is not code. They are bound by a formal syntax but, with the exception of particular annotations for certain tools, comments are not read by compilers. Hmm, for that matter, they are not really read by programmers.

Goldilocks and the three programmers

When it comes to estimating the value of comments, there are three basic schools of thought: any comment is a good comment; any comment is a bad comment; only comments that communicate are good, all the rest are bad. This is a bit like Goldilocks with her porridge pilfering: too hot, too cold, just right.

The view that all code should be commented to death is often the result of either a poor formal process, which is typically document-driven rather than results-driven, or bad personal experience — "Yeah, I used to have to maintain this comment-free spaghetti nightmare. So if you're asking me whether you should write comments, my view is that programmers who don't write comments should be sacked."

The view that code should always be comment-free is sometimes derived from a notion of program purity, so that code should be unsullied by anything that the compiler does not use. The (worthy) ideal is that code should be self-documenting. In other cases, the ideal is one of laziness, a political stance based on some notion of individual programmer liberty, or both.

Inevitably, code and comments in the former style fan the flames of support for the latter style, and vice-versa. The third way on this is that code should indeed be free of most of the kinds of comment that satisfy the "any comment is a good comment" crowd. A constructive approach to making code self-documenting should be used rather than a polemical only-good-comment-is-a-dead-comment.

Commenting should be minimal. However, minimalism is not the same as nihilism. Code should be made self-documenting not so much by the omission of comments but by the adoption of practices that make intent clear and code brief. Labouring the point and using verbose coding practices do not make code clearer, just laboured and verbose. Most comments then become surplus to requirements (functional requirements, operational requirements or developmental requirements [1]).

Once this more minimalist approach based on communication is employed, comments can then fill the gap between what can be and is said by the code and the intent and model the programmer is trying to communicate to the reader. Adopt this approach and the role of comments then becomes clear, as does their value.

Mostly useless

A comment that is wrong is by definition lying to the reader. I don't think that it would be too

FACTS AT A GLANCE

- **Comments are a form of code that is not really code, so they are not subject to the same rigorous practice as executable or declarative code.**
- **Comments should be about communication, not bureaucracy or storytelling.**
- **Comments that get in the way, because they are either wrong or useless, are not harmless and should be removed.**

contentious to say that such comments should be deleted without further ado. Such summary execution takes care of a great many comments that you may encounter when dealing with old code.

Other comments seem less harmful because they are not misdirecting the reader, but at the same time, they are not exactly useful, while some that appear useful are useful for the wrong reasons. A comment with no use is, by definition, useless. Useless comments seem to fall into one of a few categories:

- *Multiple copyright messages or some other repeated legalese and admin:* One of these per file is both sensible and quite enough; one per method is ridiculous. And given that the one occurrence in a file is perfunctory, dull and often quite long, there is a good case to be made for putting it at the end of the source file, where it can be easily ignored, rather than at its head, where it becomes the first thing you see when you open the file.
- *Comments that include version information:* This is a job best left to, and information best left in, the version control system. If you are following a highly iterative development style, you will find that such comments rapidly come to dominate the source file. Sometimes comments can be revealing, but embarrassingly so: I recall a collection of source files that had "amended to allow compilation" as their first version change. However, if your house rules feel the need to mandate such versioning fluff in the source code, see if you can place it at the end of the file out of harm's way.
- *Comments that describe the intent of a method in its implementation:* The intent of a method is best described in its name and in any comment outside its implementation that presents the method to the outside world, e.g. a sketch of its contract [2].
- *Comments that describe the implementation of a method along with its intent:* Does this mean that the external comment will be updated every time the implementation of the method is changed, whilst leaving the intent intact? Not in my experience.
- *Comments that parrot the code:* Assume that the reader can read the language you are writing. If they cannot, your comments will be of minimal use unless they include a guide to the language and its idioms (and no, that is not a suggestion, although I have heard of it being done). A comment that documents a method as taking three arguments is a waste of space: that fact is obvious from the fact that its argument list has three arguments, or that fact is clearly not a fact if the method actually takes four. Likewise, documenting the type and name of each argument is a good way of repeating what is already in the code, but without the guarantee of correctness. A comment that tells you that an `if` statement on equality between two variables is comparing two variables for equality is similarly unhelpful.
- *Comments that tell you what someone has done:* At best this is gossip — "Joe added the following five lines" — and at worst it falls rapidly out of date — "Joe added the following five lines" ahead of ten lines of code... so which five did he write and which mystery individual wrote the other five?
- *Comments that tell you what to do:* As generated by various tools, TO DO comments suggest that there is something to be done. This is fine if there is something to be done. However, if you leave the comment in the code it means that it's not finished. When done, delete. In fact, even changing the comment to DONE might be preferable to leaving it as TO DO! Some programmers feel that they can't touch the comment because it was authored by an automatic code generator. Given the quality of most automatically generated code, I don't understand the source of such reverence.
- *Comments that describe what a piece of complex code does*: Sometimes such comments can prove useful, but more often than not, they are an indication that something needs to be re-factored and simplified. Such comments are glossing over something that should be fixed. For instance, if a method seems long and winding, don't comment it: split it up and make it shorter. Divide and conquer worked for Julius Caesar and Tony Hoare; it also works a treat when you are wondering what to re-factor and why.
- *Commented-out code*: The version control system is there to act as the archivist and historian of your system. Commented-out code does nothing, therefore it's not really code, therefore it will make no functional or operational difference if you delete it, but the developmental difference will be an improvement in readability and the removal of any lingering doubts in the reader's mind.

Heavy commenting often includes many of these kinds of comments, and I suspect that some of it comes from programmers who were not fully comfortable with the code. For some code the sentiment is understandable, but think for a moment: programmers who do not understand the code will invariably describe it incorrectly, so how good will their comments be? Pouring petrol onto a fire will not put it out.

In conclusion

Like gold, comments are valuable only when they are not found under every rock or on every surface, but at the same time, their existence must not be an impossibility. If comments flow as freely as air they are worthless, acting as speed bumps to comprehension rather than green lights. Prefer less to more, but not necessarily none [3]:

In other words, it's not just a case of prefer good code to bad comments: prefer good code to good comments. This recommendation is an easy one to put into practice: read your comments and if they give you news not trivia keep them, otherwise reach for Del boy.

If you want to know what is happening, look at the code. If you want to know why, then the gap between what can be read from the code and what you need to know is what offers space for documentation. Some of this may be in the form of comments. ■

References

1. Kevlin Henney, "Inside requirements", *Application Development Advisor*, May 2003.
2. Kevlin Henney, "Sorted", *Application Development Advisor*, July 2003.
3. Kevlin Henney, "The rest of the best", *Application Development Advisor*, June 2002.

Kevlin Henney is an independent software development consultant and trainer. He can be reached at http://www.curbralan.com.