BLOG @CACM

The *Communications* Web site, <http://cacm.acm.org>, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.

Follow us on Twitter at <http://twitter.com/blogCACM>

DOI:10.1145/3193752

<http://cacm.acm.org/blogs/blog-cacm>

Commenting on Code, Considering Data's Bottleneck

Edwin Torres considers the enduring value of code comments, while Walid Saba wonders if we have overreacted to the knowledge acquisition bottleneck.

Edwin Torres Why Code Comments Still Matter

<http://bit.ly/2FgllP9>

February 26, 2018

In computer science, you are taught to comment your code. When you learn a new language, you learn the syntax for a comment in that language. Although the compiler or interpreter ignores all comments in a program, comments are valuable. However, there is a recent viewpoint that commenting code is bad, and that you should avoid all comments in your programs. In the 2013 article *No Comment: Why Commenting Code Is Still a Bad Idea*, Peter Vogel continued this discussion.

Those who believe commenting code is a bad idea argue that comments add unnecessary maintenance; when code changes, you must also modify comments to keep them in sync. They argue it is the responsibility of the programmer to write really obvious code, eliminating the need for comments. Although these are valid reasons to avoid

commenting code, the arguments are simplistic and general; comments are *necessary* for a variety of reasons:

1. Not all programmers can write really obvious code. Beginning programmers are just happy to write a correct program; they are still mastering the craft. Even experienced programmers write sloppy code. Programs are unique like fingerprints, so judging whether code is obvious is a subjective call.

2. It can be tedious to comment too much, but some comments are like titles and subtitles in articles; they guide, provide context, and convey overall meaning.

3. Comments are not just for code; they can document important program information such as author, date, license, and copyright details.

4. Some programming languages are cryptic, like the Glass programming language. This sample program (http://esolangs.org/wiki/Glass#Fibonacci_sequence) is hard to decipher, but prints a Fibonacci sequence. Is the meaning of this program clear to you? It may be possible to write it in a more obvious way, but

a comment could convey its meaning.

5. Some companies require employees to comment their code. Google's programming style guides specify how to write comments in programming languages like Java, JavaScript, and C++.

6. Specialized comments allow tools like javadoc, JSDoc, and apiDoc to generate professional, thorough, and consistent documentation for programs.

7. Comments can be placeholders for future work, a useful way to create an outline for a large program. The Eclipse Integrated Development Environment (IDE) creates a TODO comment when it generates a main method, a reminder to add the starting code of a program.

Commenting may be tedious or overwhelming, but it is valuable in many situations. Even if you think you write obvious code, try reading your code months or years later; will it still be obvious to you, or would you wish for comments?

Comments

*A key characteristic of comments is with respect to narration, as Ward Cunningham has pointed out. It can be important to distinguish what the code is *for*, not just what it is, and what the key assumptions and constraints might be. It is valuable to develop a grasp for what the requirements are, and code is rarely a substitute for that.*

—Dennis Hamilton

Dennis — That is a good point. There are times when you just need a quick overview of the code, without spending time to trace through it. Comments help here, assuming they are correct.

—Edwin Torres

There are many things we agree on. I should, for example, point out that my objection is to comments “in” code, not to comments at, for example, the start of a method, that include the name of the author, date created, and so on (though often, source control can automate that work).

I also definitely agree with you that code at the start of a method should describe what the method is “for”—why the method exists or was written. This is something even really obvious code often cannot communicate. At best, really obvious code can communicate the “how” of what the code does (though the name of the method can sometimes help address what this code is “for”).

We even agree about the need to comment cryptic code. I will, however, suggest that the problem should not be first addressed by writing a comment: it should be first addressed by writing obvious code. If there is a bug in that Fibonacci generator or there's a need to enhance it, clear code will help you find the bug or enhance the code in a way that a comment cannot. I suggest we consider a comment, in this scenario, as an apology from the original programmer to the next one: “I did the best I could but, for various reasons, I still ended up with this unfortunate code. Here's what I can do to help.”

I like the idea of comments as headings to guide a programmer through the code. As a part-time technical writer, that especially appeals to me. Of course, I'm going to suggest those comments (like headings) be only two or three words in length and, perhaps, evidence that this method should be refactored into several methods with their names reflecting those titles. But, in many cases, I can see that being overkill.

In fact, I will disagree with you in only one place: the idea that a programmer who can't write obvious code is, for some reason, capable of writing a comment that is (a) obvious, (b) accurate, and (c) complete. We hire programmers, after all, for their ability to write code, not for their ability as technical writers. And if the comment isn't accurate, well, to paraphrase “The Elements of Programming Style,” the code and its comments provide two descriptions of the processing logic. If they disagree, only the code is true. At this point, code that goes beyond describing what a method is for creates a maintenance burden of fixing the comments to keep them in line with the code. Programmers are busy enough.

—Peter Vogel

Peter — My goal was to highlight some additional needs for comments. I agree that the “code doesn't lie.” Also, too many comments can be overwhelming and distracting. I find it interesting that a discussion on comments even exists today. Who would've thought?

—Edwin Torres

That's a great list of reasons to comment, to which I would add one more: what is obvious to *you*, the author of the code, probably isn't obvious to *me*, the reader. If you've been working on the feature you are building for the last week, you have spent that week building a mental model of that area of the problem domain and its mapping onto the software system. I do not have that model. Developers should try to empathize with the developer who understands software, but is new to *this problem*, and help improve their understanding.

—Graham Lee

Graham — Great point. One of my earliest lessons in programming was that it is much harder to change someone else's program than create your own. When used effectively, comments can help here.

—Edwin Torres

Walid Saba Did We Just Replace the ‘Knowledge Bottleneck’ With a ‘Data Bottleneck’?

<http://bit.ly/2tdSHfS>

February 26, 2018

One of the main reasons behind the quantitative and data-driven revolution that took artificial intelligence (AI) by a storm in the early 1990s was the brittleness of symbolic (logical) systems and their never-ending need for carefully crafted rules. The rationale was that there is a knowledge acquisition *bottleneck* in the quest to build intelligent systems. The new cliché? Let the system ‘discover’ the logic/rules by crunching as much data as you can possibly get your hands on. With powerful machine learning techniques, the system will ‘discover’ an approximation of the probability distribution function and will ‘learn’ what the data is, and what it means, and will be ready for any new input hereafter. It all sounded good; too good to be true, in fact.

Notwithstanding the philosophical problems with this paradigm (for one

thing, that induction is not a sound inference methodology—outside of mathematical induction, that is), in practice, it seems that avoiding the knowledge acquisition bottleneck has not resulted in any net gain. In the world of data science, it seems data scientists spend more than half of their time not on the science (models, algorithms, inferences, etc.), but on preparing, cleaning, massaging, and making sure the data is ready to be pushed to the data analysis machinery—whether the machinery was SVM, deep neural networks, or what have you.

Some studies indicate data scientists spend almost 80% of their time on preparing data, and even after that tedious and time-consuming process is done, unexpected results are usually blamed by the data ‘scientist’ on the inadequacy of the data, and another long iteration of data collection, data cleaning, transformation, massaging, and more, goes on. Given that data scientists are some of the most highly paid professionals in the IT industry today, isn't 80% of their time on cleaning and preparing the data to enter the inferno something that should raise some flags—or, at least, some eyebrows?

Such techniques, even after the long, tedious process of data cleaning and data preparation, still will be vulnerable. These models can be fooled by data that is similar, yet it will cause these models to erroneously classify them. The problem of adversarial data is getting too much attention, without a solution in sight. It has been shown that any machine learning model can be attacked with adversarial data (whether an image, an audio signal, or text) and can make the classifier decide *anything the attacker wants the classification to be*, often by changing one pixel, one character, or one audio signal—changes otherwise unnoticeable for a human.

Maybe not everything we want is in some data distribution? Maybe we are in a (data) frenzy? Maybe we went a bit too far in our reaction to the knowledge acquisition bottleneck?

Edwin Torres is a full-time software engineer at The MITRE Corporation and an adjunct professor of computer science at Monmouth University. Walid Saba is Principal AI Scientist at Astound.ai, where he works on Conversational Agents technology.

© 2018 ACM 0001-0782/18/5 \$15.00