Programming Pragmatically:

A comment on comments

Aaron K. Nelson

Morehead State University

Introduction

About Programming

The ability to convey meaning—to distill complex and abstract ideas into transmittable symbols—has always been the linchpin of human society. Opposable thumbs may have been an important catalyst for the emergence of mankind as the dominant species, but our advanced communication is what ensured our continued growth.

It is now the 21st century, and that growth is exponential. Relatively recent developments in technology have created entirely new media for communication with fewer constraints. A scant 200 years ago, language barriers and proximity constituted insurmountable limitations. Today, a child can talk to a stranger on the other side of the world almost instantaneously, while computers do the heavy lifting of instantaneous message transmission and translation.

The computer programs written for tasks like these are a new world for communication. Programming languages are several layers of abstraction above the nearly incomprehensible binary and assembly languages that machines speak. Though programming languages are international and universal, they overwhelmingly embrace English as the *lingua franca* for operational words and function names. Take a very simple example:

```
if(condition) {
      do something...
} else  {
      do something different...
}
```

In this example, the italicized words are completely open-ended; the variable names, function names, and anything else written by the programmer could be any national language. But the important construct framing it all—the “if” and “else” universal to all the well-known and widely adopted programming languages—are borrowed from English. Numerous other words such as “for” and “while” are used as well. Bear in mind that major coding tutorial sites like W3Schools, online code repositories like GitHub, and social question-and-answer, problem-and-solution sites like StackOverflow all use English.

About Programming Comments

Every major programming language provides a means for the author to convey information about the program: comments. Comments are superfluous text added to code, not written to the machine, but to the self or to other people. They are completely natural language framed in a way that the code’s compiler does not read and attempt to execute it. Comments are used to give clarity to code, provide a space for multiple collaborators to discuss the code, and allow for quick attention-grabbing to critical parts of code. Given the minor prevalence of English in the code’s structure and the dominance of English in the programming community, comments in code samples pulled from online repositories is almost always English as well.

The nature of inter-programmer communication via code comments is not unlike other forms of communication. Just like workplace conversation or Internet chat, comment text is governed by some amount of social mores, professional courtesy, and grammatical conventions. But the frequency and purposes for commenting is a matter of great debate among programmers. Some parties, such as programming blogger and Stack Exchange network co-founder Jeff Atwood, contend that code should be self-explanatory, and utilize descriptive names for variables and functions to make the code clearer.

"You should always write your code as if comments didn't exist," he argues. "[Developers] should be relying on the *code* to tell the story." (Atwood, 2008).

The converse opinion is that any meta-information about the code—written to be easily human-readable—is welcome. Frequent, thorough comments replace the need to figure out how a piece of code works, why it works, and what purpose it serves in the program.

Their open-ended possibilities mean comments can be added to claim authorship of a piece of code, provide an in-program revision history, or a stage for collaborating developers to crack jokes. How acceptable these and other uses of commenting is completely at the discretion of the programmer.

Studying Comments as a Realm of Communication

Programming language comments are a relatively unstudied domain of communication, though they are ripe subject for analysis. If a program is a sterile means to an end, comments are the human counterpoint—an organic means for the programmer to communicate and a respite from the unnatural process of writing code.

Though they can serve any purpose or no purpose at all, comments "in the wild"—meaning those found in open-source programs available online for copy, distribution, and use in anyone's project—often serve one primary goal: to clarify the code, making it easier to hop into someone else's code and know how it works. Therefore, while it would be fascinating to explore ethnocentrism or androcentrism in this context, it would serve us to first study their efficacy. How well do comments work to help programmers become familiar with a given piece of code? How does their complete exclusion hamper inter-programmer communication? I hypothesize that the presence of thoughtful, meaningful comments will reliably and demonstrably help new programmers become familiar faster than relying on code to be self-explanatory.

Computers are not an emerging technology—they are here today and here to stay. The growth of programming as a career and as a hobby is accelerating to meet the demands of our changing culture, and programming itself is an area of communication that deserves the attention of researchers.

Review of Literature

Communication scholars are quick to deconstruct supermarket conversation, media advertising, and schoolyard banter as prime texts for how people share ideas. Programming languages sometimes get lost by the wayside to more mainstream fields of research.

Madeo (1990) discusses an experiment held to determine if students who were allowed to define the language elements of a programming language in their own terms would be able to adopt it easier and have an easier time writing a simple program. His results indicated exactly the opposite. Perhaps, then, it is best to have a rigid programming language with a steep learning curve, and reserve the more approachable language for comments.

Burgos, Ryan, and Murphree (2007) have some additional insight. In analyzing the way programmers approach and maintain legacy code (very old code, often wherein the original developers are long gone), one programmer—in explaining how their model was correct—mentions that the process starts with “external sources”—talking to the original developer if possible, of course, and if not, analyzing the way the code is used and any comments in the code. This creates a superficial understanding, which is later filled in by a deeper internal analysis of the code (p. 222).

Method

The Programs

In order to facilitate a study that did not rely on only one language *and* find participants who would know several languages, I chose a field that utilizes several various languages that are somewhat dependent on each other: dynamic website design. In brief, the languages involved:

- HTML (Hypertext Markup Language) – Not a programming language at all. HTML is the markup around the information on a page, denoting the structure of the information—e.g. this is a paragraph, this is a heading, this is a list, etc.
- CSS (Cascading Style Sheets) – Also not a programming language. CSS is a lightweight language designed to be used in conjunction with HTML to simplify the process of styling content—e.g. this should be 12 pt. Times New Roman font, this should be black, etc.
- JavaScript (sometimes "JS")– A programming language that is executed on the client's computer. JavaScript is used for detecting user actions, making calculations on a user's computer, and animating and changing the content of a web page.
- PHP (PHP Hypertext Processor) – A programming language executed on the web server. PHP deals with information kept in databases and complex mechanisms on the server, like tracking that a user is logged in or reading and writing cookies.

Four projects were collected from the online code repository GitHub for use in the study. All were openly licensed for such use. All were originally heavily commented, some even including "read me" text files with additional information. For each project, I created a challenge—a completely arbitrary and relatively simple set of changes to be made on the existing code. Every project utilized some amount of HTML and CSS. They were:

1. A PHP login system, allowing for users to have a site "remember" them next time they visit. Programmers were asked to further sanitize the username and password fields and include a third field throughout the script—a generic "user ID number."

2. A JavaScript slideshow script. Given a set of HTML images, JavaScript hides and shows them in turn at the click of buttons on the page. Programmers were asked to change the rate at which the images changed.
3. A PHP/JavaScript slideshow script. Given a set of files in a folder on the server, PHP creates the HTML images and passes along information about them to JavaScript (their size and a field of caption text for each one). Programmers were asked to create an arbitrary new array of information about the images, and have JavaScript output the information in a popup.
4. A JavaScript employee schedule editor. The user has a spreadsheet-like form in which they can enter times for employees to work. JavaScript takes their numbers and calculates total hours worked in a week, and creates a graph of the data for easy viewing. Programmers were asked to have the system automatically add an hour for employees who work on Tuesdays.

The Programmers

Messages asking for volunteers proficient in programming were posted on a technology-themed online message-board. Readers were asked if they were interested in donating their time to the study of communication between programmers. At no point before or during the study was it made clear that the focus of the study would be code comments.

Four volunteers participated, every one of them reporting varying levels of proficiency in the languages used. They were asked their age and gender to provide some anonymous context to their results. An overview of the participants, in the order that they volunteered:

1. Programmer 1 is a 26-year-old male. He claims to have five years of experience in all four languages.
2. Programmer 2 is a 31-year-old male. He claims to be proficient in all but PHP, which he is in the process of learning.

3. Programmer 3 is a 25-year-old male. He claims to be proficient in all but PHP, which he is only tangentially familiar with, but is proficient in numerous other similar programming languages.
4. Programmer 4 is a 22-year-old male. He claims to be proficient in all but PHP, which he is not at all familiar with.

Each program was duplicated into two copies—one richly-commented original and one completely stripped of all comments (except licensing information). Each programmer was provided with two commented programs and two uncommented programs, and two text documents—one with the instructions and the challenges for each program, and one with instructions for after they finish. Participants were asked not to search online for help, nor to seek out the documentation about the programs. For each program, participants were asked to time (to the nearest second) how long it took to complete the challenge set forth. At the end (in the post-study document), participants were asked to think back to each challenge and explain their process to the solution, any difficulties they had with each, and what changes to the code would have made the task easier or harder. With this information turned in, I revealed to the participants that the purpose of the research was to study the efficacy of comments, and I thanked for their time and contribution.

This method permits a scientific and balanced analysis illuminating both a.) how a population of programmers can amend a given program varied only by the inclusion of comments and b.) how a given programmer performs with a set of various programs, with and without the inclusion of comments.

The programs were to be distributed evenly, though randomly. It fell as follows:

	PHP Login	JS Slideshow	PHP/JS Slideshow	JS Schedule editor
Programmer 1	C	C	UC	UC
Programmer 2	C	UC	UC	C

Programmer 3	UC	C	C	UC
Programmer 4	UC	UC	C	C

Before proceeding, it would serve the study to highlight the limitations of the study, the foremost being sample size. Given so few programmers who saw the study through to the end, and so few programs readily available that lend themselves well to being adapted in such a way, the results of this study are far from being conclusive. There is also a dearth of female programmers and programmers outside the 18-35 age demographic . Given how differently men and women communicate, and how people of different ages, ethnicities, and cultures might communicate—the results here cannot be extrapolated very far.

When considering some respondents were not entirely fluent in all languages, it is of real importance to study more deeply how the inclusion or absence of comments do or do not help those who are still learning a language's concepts. With these scant numbers, it was impossible to distribute commented/uncommented samples in such a way as to analyze this.

Additionally, all participants timed themselves and the study depended on the honor system as to whether the challenges were actually completed at all. I touched on this fact in their introduction document, stressing that since the study was completely anonymous and that there was absolutely nothing to gain by false reports, the study depended entirely upon their honesty.

Data

The results of the study were as follows. Time is given in mm:ss format.

	PHP Login	JS Slideshow	PHP/JS Slideshow	JS Schedule editor

Programmer 1	C / 3:08	C / 0:26	UC / 1:56	UC / 4:36
Programmer 2	C / 14:15	UC / 1:02	UC / 3:40	C / 2:32
Programmer 3	UC / 6:19	C / 0:23	C / 2:21	UC / 1:58
Programmer 4	UC / N/A	UC / 1:43	C / 6:40	C / 2:44

If we analyze the difficulty of each program's challenge, based on time taken, the PHP login system comes out as hardest—taking 3 participants 23 minutes and 42 seconds while the fourth did not attempt it at all, citing its difficulty. The JavaScript slideshow would be easiest, taking all four participants just 3 minutes, 34 seconds total to complete.

Getting at the heart of the matter, however, the commented code does not always win the day. Though it seemed to help a great deal for the JS Slideshow, the results for the PHP Login and PHP/JS Slideshow were inconclusively close or tipped in the favor of uncommented code. More likely, however, Programmer 2 and Programmer 4's lack of PHP experience could be skewing the data—P2's lengthy time in the Login project and P4's time in the PHP/JS slideshow are likely to blame.

The old adage may not be correct. The numbers could be lying. A couple of their responses seem to corroborate my shrugging off of the numbers. For Programmer 4, staring at what is admittedly a somewhat complex PHP script, the sheer fact that no comments were provided may have intimidated him away from trying some trial-and-error: "I tried, but I just have no idea what I'm looking at. No comments anywhere, all the functions sound the same, I don't even know what I'm supposed to be doing."

On the other hand, he also makes mention of the dubious function names. This is true—the various functions sprinkled throughout which check username and password data against the database,

against the existing session variables, and against cookies all use ambiguous names. When a function is just called “checkLogin,” that does not bode well for someone trying to find its exact purpose.

Programmer 2 was the only other respondent who explicitly mentioned comments in his response. In that, the Schedule editor script, a comment took him directly to the bit of code he needed to tweak.

Conclusions

I set out to test the efficacy of programming comments. I more conclusively answered that familiarity with a language matters more than anything else, and no amount of comments can save a programmer who simply does not know a language’s syntax.

The results did indicate some small support for my hypothesis—staring at a long block of foreign code without any help or context, some programmers will yearn for comments to help. When presented with a new block of code, some will immediately seek out keywords; even if the programming language does not use words like “weekday,” perhaps the author included a mention of it in his comments.

Again, the small sample size and the varied skillsets of the respondents prevents anything by way of a truly conclusive result. A qualitative analysis turns up a couple bare mentions of comments, but from the beginning, the counterpoint has been that descriptive variable and function names can often serve the same purpose.

References

Atwood, J. (2008). “Coding Without Comments.” Retrieved from http://www.codinghorror.com/blog/2008/07/coding-without-comments.html

Burgos, C. L., Ryan, J. H., & Murphree, E. (2007). Through a mirror darkly: How programmers understand legacy code. *Information Knowledge Systems Management*, *6*(3), 215-234.

Madeo, L. A. (1990). The effect of user-specified language elements on learning to program. *Journal Of Research On Computing In Education*, *22*(1), 336.

Appendix

PROGRAMMER 1 RESPONSES

RESPONSE TO “PHP Login”: To sanitize, I did a search for $_POST, since the form data would likely be turned in that way. I found the functions being run on it and added the regex one.
To make the third field, I just followed the $_SESSION variables for username and password and copied them in every instance with a new one.

RESPONSE TO “JS Slideshow”: I looked through the "setTimeout" calls to find the one responsible for how long the images hang and changed it.

RESPONSE TO “PHP/JS Slideshow”: I copied the existing array for captions and printed it into a javascript array just like the captions. Instead of $("#caption").text() I just called alert() in the image change function.

RESPONSE TO “JS Schedule editor”: I had to find the foreach loop where the function broke it down by weekday and added it there.

PROGRAMMER 2 RESPONSES

RESPONSE TO “PHP Login”: this was some pretty advanced php. i did the regex on the form data (not sure it was right) and all i could do for the third session thing was to copy/paste the variables already there.

RESPONSE TO “JS Slideshow”: took me a while to find the right number for the transition. would have been easier if the var names were clearer.

RESPONSE TO “PHP/JS Slideshow”: copied and pasted the stuff from caption, but used window.alert to make the popups.

RESPONSE TO “JS Schedule editor”: found a comment about the loop that breaks up the weekdays, so i added an “if” there, hours++.

PROGRAMMER 3 RESPONSES

RESPONSE TO “PHP Login”: I did the regex thing for the POST data and just made a third session variable for the userID.

RESPONSE TO “JS Slideshow”: Just changed the one number from 6 seconds to 8.

RESPONSE TO “PHP/JS Slideshow”: PHP does arrays a little differently than I’m used to, but I just had it spit out a Javascript array and put the alert in with the rest of the stuff.

RESPONSE TO “JS Schedule editor”: I tracked it to the point where it broke up that massive result set by weekday and put a conditional increment in there.

PROGRAMMER 4 RESPONSES

RESPONSE TO “PHP Login”: I tried, but I just have no idea what I’m looking at. No comments anywhere, all the functions sound the same, I don’t even know what I’m supposed to be doing.

RESPONSE TO “JS Slideshow”: Changed the numbers in the functions until I found the one that controlled the slideshow speed.

RESPONSE TO “PHP/JS Slideshow”: I just borrowed the captions array and made it do popups instead of captions.

RESPONSE TO “JS Schedule editor”: Saw a part where it says “break up into weekdays” and told it to up the hour by one if its a Tuesday.