Noah Wilson  
Professor Whitney Reuschling  
Technical Writing  
18 September, 2022

Technical Writing in Computer Science

Computer Science as a term encompasses a wide range of related fields, which employ a wide variety of skills. While it would be over-generalizing to make too broad of a statement about all of these fields, I think I can safely say that all will require some degree of technical writing skills. There are three main types of technical writing used in computer science fields which I will be discussing today. These are documents related to project planning and organization, code documentation, and the act of programming itself.

Modern software projects are a large undertaking, requiring the expertise and coordination of tens, hundreds, or even thousands of individuals. Like any undertaking on this scale, coordination between different levels of management, departments, and teams is absolutely crucial. Project planning involves drafting and revising reams of documents and technical specifications. These need to be written precisely so that expectations can be set clearly and performance can be appraised fairly. If one team demands too much from another without understanding the unique challenges they face, their view of their coworkers can become misinformed, leading to worsened relations and organizational integrity. If we step up a level, we will be concerned not only with internal company collaboration, but client relations as well. The agile software development model, a modern approach for accelerating the development of  software solutions, emphasizes this relationship between the firm and the client. The Manifesto for Agile Software Development states on their website (agilemanifesto.org):

...Through this work we have come to value:  
Individuals and interactions over processes and tools  
Working software over comprehensive documentation  
Customer collaboration over contract negotiation  
Responding to change over following a plan...

This development methodology has become extremely popular in the modern tech industry, making technical writing skills an integral part of performing one’s duties in this field. Oftentimes, the client does not know what the solution is to their problem, but they are looking for the expertise of someone (in this case, the software developer) who will be able to find that solution and deliver on it. Communication between these two parties is critical, and both parties need to be able to adjust to obstacles as they arise and constantly adjust their expectations to the situation at hand. Much like the friction that can arise with misunderstandings between teams, the same can occur between firm and client.

Until now I have primarily addressed more business-related forms of technical writing, but this is far from the only manner in which technical writing skills are useful in software development. Although it may not be immediately apparent to some, writing software is itself a form of technical writing. Computer code is little more than precise language communicating the instructions necessary to accomplish a certain task. In order for these instructions to be read by the computer, they must be translated into byte code which is incomprehensible to the human eye, but a good programmer is not paid simply to produce this byte code. The real value lies in having source code which not only works now, but can be maintained and built upon in perpetuity. Almost all code is written with the intention of being read and modified by others at some point in the future, especially in a workplace setting. The best code communicates to both the computer (compiler) and to other programmers. While documentation is an important part of writing code, it is more valuable to other programmers that the source code itself is readable and understandable. But what makes code readable and maintainable? Well I could go on and on about the conflicts of various development methodologies, but that is not necessary. Despite disagreements about specific styles of programming, there are many widely agreed-upon do’s and don’ts  for writing readable code. One such example is the POSIX (Portable Operating System Interface) standards, which define a whole set of guidelines for writing APIs for Unix-based operating systems. The official POSIX certifications, along with many other standards certifications, are established by the IEEE (Institute of Electrical and Electronics Engineers), which have helped to push for common standards in a diverse array of technological fields (ieee.org). Additionally, most organizations will provide extensive guidelines for code style and structure, so programmers will know exactly what is expected of them. Well written, these guidelines should still give some wiggle room for individual expressiveness. These rules should provide a structure for programmers so that they can be more productive, but should not go so far as to inhibit their work by restricting unnecessarily what they can write.

Lastly, I would like to touch on writing documentation for software. Documentation comes in many forms, from direct code documentation, to Human interface design and API descriptions. It may be written for internal use only or for public use and the intended audience will impact the form and function of the documentation. Additionally, there may be separate documentation written for programmers and end-users. Often, a user will only want to know how to use the user-interface of the software, while a programmer will want to know how to inter-operate with the source code directly. This should be kept in mind, and the writer should be aware of who will be reading their work. While the documentation written by a single software developer will likely not be seen by anyone outside the organization, its importance should not be understated, since it will form the basis for everything that is built upon it. The writing will undergo many stages of revision before reaching the public, meaning it will be seen and influenced by many individuals along the way. Writing ethically in this environment is essential because any mistakes can have far-reaching effects. Biases and assumptions made in one phase can seep all the way down the chain and end up in the final product which is delivered to consumers. These can begin as small, implicit ideas, but become systemic in the work as it is iterated upon and expanded.

I hope that this brief explanation of technical writing in the field of computer science (or more specifically, software development) has given you at least some insight into how these skills manifest in this environment.

Works Cited

“Manifesto for Agile Software Development.” Agilemanifesto.org, 2019, agilemanifesto.org/iso/en/manifesto.html.

“IEEE SA - P1003.1.” IEEE Standards Association, standards.ieee.org/ieee/1003.1/7700/#Standard. Accessed 18 Sept. 2022.