CONF-9510245--3

A Perspective on Computer Documentation: System Developer vs. Technical Editor

RECEIVED
JAN 17 1996
OSTI

Elithe Truett Carnes
University of Tennessee

and

Lorena F. Truett
Oak Ridge National Laboratory*

Prepared for
19th Annual Practical Conference on Communication
October 12-13, 1995
Oak Ridge, Tennessee

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

*Oak Ridge National Laboratory is managed by Lockheed Martin Energy Systems, Inc. for the U.S. Department of Energy under contract DE-AC05-84OR21400.

"The submitted manuscript has been authored by a contractor of the U.S. Government under contract No. DE-AC05-84OR21400. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes."

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

A Perspective on Computer Documentation: System Developer vs. Technical Editor

Elithe Truett Carnes and Lorena F. Truett

Between the computer-knowledgeable "techie" and the technical writer is a chasm created by differences in knowledge bases and skills. Although this gap is widened by misunderstandings and misconceptions of system development roles, it is bridged by mutual need and dual appreciation.

Often the editor/writer is "behind" from beginning to end. The writer normally joins the team after the programmers are well into system development and do not want to "waste time" discussing fundamentals. The writer is usually excluded from technical discussions because it is assumed that he/she would not understand anyway. Later in the system development cycle, the writer has no time to polish the documentation before a new version of the software is issued which implies that the documentation must be revised. Nevertheless, the editor/writer's product is critical for the end-user's appreciation of the software, a fact which promotes unity to complete the comprehensive package of software and documentation.

This paper explores the planks in the bridge that spans the chasm between developers and their fundamental PR agents, the technical editors/writers. This paper defines approaches (e.g., The Circling Theory) and techniques (Bold Thrust!) employed for effective communication -- between software developer and technical writer as well as between the software and the end-user.

Introduction

Cooperation among the members of a software development team is critical, as is an understanding and appreciation for the roles and tasks of other members of the team. Otherwise, the software may be very good but the documentation inadequate and ineffective.

Computer documentation historically has been the undesired stepchild of system development. In an article in InfoWorld, Pamela Beason (co-author of Technical Writing for Business and Industry, Scott, Foresman & Co., 1990) states that "The key flaw is that documentation is usually not written for the people who have to use it, [such as] either an end-user or another programmer. It's written by the developer for himself, and therefore written from a developer's point of view." Beason goes on to note that one of the main reasons users do not read documentation is because it is not written in such a way as to be useful to their purposes. J. G. Angus asserts that the reasons for poor

documentation cannot be attributed to one single fault, but most of the erring results from poor design and poor management (Angus, 1995). Ed Yourdon notes that "in the culture of development, documentation is always considered an add-on, not real work. It's not considered something you need to make a working program. Programmers will do whatever it takes to make a working program, as long as that whatever isn't documentation" (Angus, 1995).

In this paper we describe perspectives of both developers and writers, and we discuss the approaches and techniques employed for effective documentation. W. R. Dodson breaks the process of documentation into three subcategories: Storming, Norming, and Performing (Dodson, 1994). Storming sees members of the team vying for power (similar to the Circling Theory described in this paper). Norming occurs once the team is past the battle for control; it is a period of commonality and working toward a unified goal (similar to Iterative Progression). Performing "is the summit of team evolution," or the success of completed computer documentation.

In this paper, the Software Development Team is assumed to consist of a leader or manager, programmers (also referred to as "developers"), and a technical writer/editor (also referred to as "documenter"). Software Documentation, for the purposes of this paper, includes printed manuals, reports, and pamphlets that describe the system and/or assist the end-user; it also includes hypertext and on-line help files. Software documentation excludes comments embedded within the code, which are almost always completely written by programmers for programmers and are rarely viewed by the user community.

The Circling Theory

The Circling Theory assumes that a team is composed of individuals with differing skills (including tools usage), experiences, knowledge bases, and goals (Table 1). The theory states that the team members will attempt to establish a hierarchy of importance. To advance their own positions, team members will emphasize their own strengths and/or point out the failures and inadequacies of others. This behavior, which is pronounced during the formation of the team, may continue on a lesser scale until the team becomes highly successful or is disbanded.

Table 1. Strength sectors of the Circling Theory defined for a software development team

Sector	Programmer	Documenter
Skills	Programming logic/syntax; programming languages	Readability, grammar; word-processing tools
Experiences	Knowledgeable about system requirements, technical issues, team personnel -- an insider	Not familiar with system tools, system requirements, people involved, end-users -- an outsider
Knowledge bases	Mathematics, computer science	English, communications
Goals	Software functionality and performance; technical requirements	Documentation accuracy and completeness; readability

The programmers' skills emphasize logic and syntax within the code, and the writer concentrates on readability, grammar, and appearance of the documentation. Indeed, different tools for getting the job done (i.e., programming languages as opposed to word processing packets) foster an environment in which the developer and the writer circle one another with wary attitudes, failing to maximize time efficiently and blaming the other's differences for any problem encountered in the documentation process. With respect to experiences, although the technical writer may have already produced tomes on other projects, he/she is the "new kid on the block" for this project. The programmer's knowledge base is technical, founded in mathematics or computer science, while the documenter typically has a degree in English or communications. Commonly, the developer questions the writer's credentials, and vice versa. Developers are leery of the writer's urge to be creative. As Patricia Williams, co-author of a book on technical writing, notes, "It's programmer ego, ... Programmers are so coddled in most organizations, they think their work is so brilliant and easy to use that it doesn't need documentation" (Angus, 1995). These hesitancies of the programmer are met with writer-based exclamations centered around the grammatical acumen of the developer. J. O. Borchers states that most programmers view documentation with the perspective that if the software was hard to write, it should be just as difficult to understand. Therefore, if the developer writes the documentation it is "typographically dreadful" (Borchers, 1995).

The Circling Theory assumes that members of the formative team have an innate lack of trust for each other. The developer often assumes that the writer is not sufficiently intelligent to comprehend the software's intricacies. Similarly, the writer's reaction to the program which must be documented often prompts such responses as "They call this 'user friendly'?!" During circling maneuvers, very little advancement is made toward the real goals: software functionality AND software documentation accuracy and completeness.

Bold Thrust!!

In the typical software development environment, the technical writer joins the team after the programmers are well into code development. Circling diversions then waste a lot of time. Because of impending deadlines (which were almost certainly set prior to the technical writer's arrival), the documenter produces a first draft based almost entirely on his or her interpretation of what the software does (Bold Thrust!). The review of this first draft is usually extreme: either too little or too much. It may be completely ignored by other members of the team, which will cause a greater problem in the next iteration if the technical writer has made errors in interpretation. In other cases, this preliminary draft may receive too much attention, being reviewed not only by the team, but also by management, the system's sponsor, and the end-users. This extensive review can be disastrous if it occurs before the writer knows enough about the software to produce a reasonable draft.

Coordinating software development and software documentation is mandatory, but difficult to orchestrate until after the initial Bold Thrust. Bold Thrust can result in significant improvements to the team dynamics. For instance, one positive result of this technique finds the writer suddenly included in meetings and discussions (even if only as a listener). Another positive result is increased willingness among the Project Manager and the developers to hear and respond to questions posed by the writer. Of course, Bold Thrust can devastate the writer's ego. As a worst-case scenario, the documenter might leave the team.

Figure 1 illustrates the process of comparing the programmers' concepts of "what the code really does" (or "will do" since the code is unfinished) against this preliminary documentation. This merging can be a disaster, or it can result in a more cooperative team, working together on the next iteration.

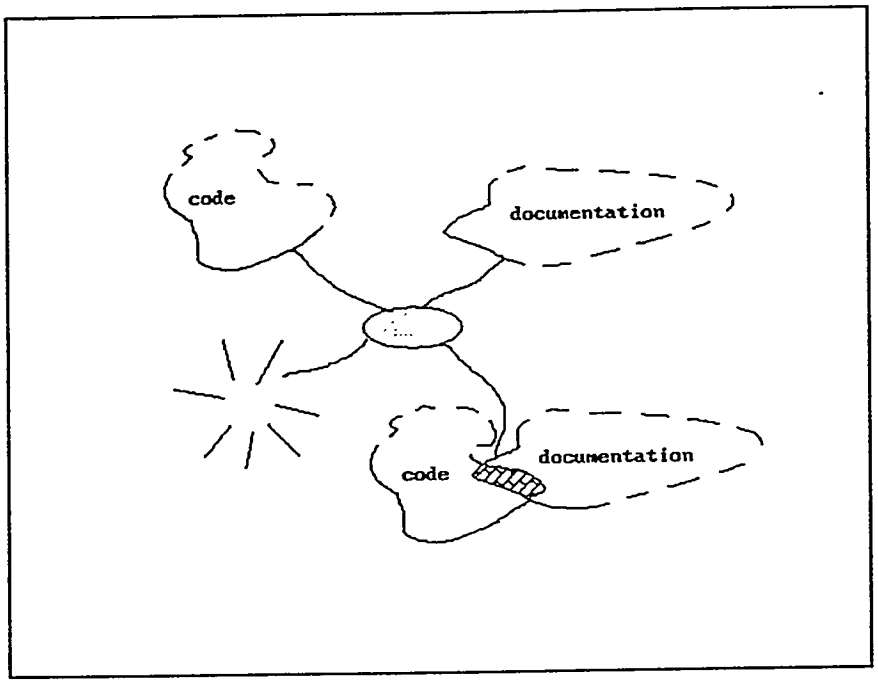

Fig. 1. Bold Thrust!

Iterative Progression

Iterative Progression is the longest stage in the documentation process, from a time perspective. From a task perspective, this stage accomplishes the most. This phase in the process is centered around cooperation and communication between the programmer and the writer. For instance, while the programmer arranges the document by logic flow, the technical writer arranges the document for the end-user. Developer and writer must agree on what is important to tell the users. Often, developers want just the facts; their drafts have little order, no formatting, and questionable grammar. The technical writer wants a nice appearance; their drafts are often verbose, fancy, eye-pleasing, and border-line superfluous.

Programmer and writer must also agree on an appropriate use of acronyms and system-specific words. Moderation is everything. Williams asserts that, "There's no technological fix ... for the problem of poor documentation. You only solve it by working with people, and most technologists don't like people; if they did, they probably wouldn't be technologists" (Angus, 1995). While this view is perhaps a bit biased, it is true that compromise and collaboration is everything in the process of creating effective computer documentation. The final product must be neither too technical nor too wordy. The programmers and writers must agree on who the end-users really are and what they need to know. Through an appropriate use of creativity, the documenter must decide how best to provide this information. The documentation does not necessarily need to follow the logic flow of the code, but the functionality of the code must be explained logically to the reader. For example, user's manuals and hypertext are arranged to guide a user (novice or advanced) through a system.

Because of its repetitive nature, this stage is characterized by persistence and patience. The writer does not need to understand all of the technical details of the software, but he/she must understand the system's functionality and must have a clear understanding of the user community. Similarly, the developer does not need to understand the writer's emphasis on format and design, but the "techie" must understand the functional requirements and must communicate the purpose of the software to the writer. Cooperation is critical, communication is essential, persistence is required, and patience makes the whole process more pleasant.

Unfortunately, during this stage, time pressures to complete the product and budget issues are mentioned more and more frequently. It would be wonderful to have sufficient time and budget to develop the world's most wonderful software with accompanying perfect documentation.

In addition to time and funding issues, about this time, the version crises occurs. Changes to code usually require changes to documentation. There is no time to polish the text because a new version is being prepared.

The Iterative Progression stage can be slow initially, but toward the end of this stage, when deadlines, budget crunches, and the eternal change cycle are all factors, it is important that programmers and writer cooperate fully. The developer has accepted the role of the technical writer as the P.R. agent

for the code/software. The technical writer must accept the reality of constant change (i.e., revisions and newer versions -- both to the code and to the documentation). This joint cooperation promotes team unity, and a good team produces a good product.

Figure 2 illustrates that as the code becomes more solidified, the version of the documentation becomes more in tune with the code.

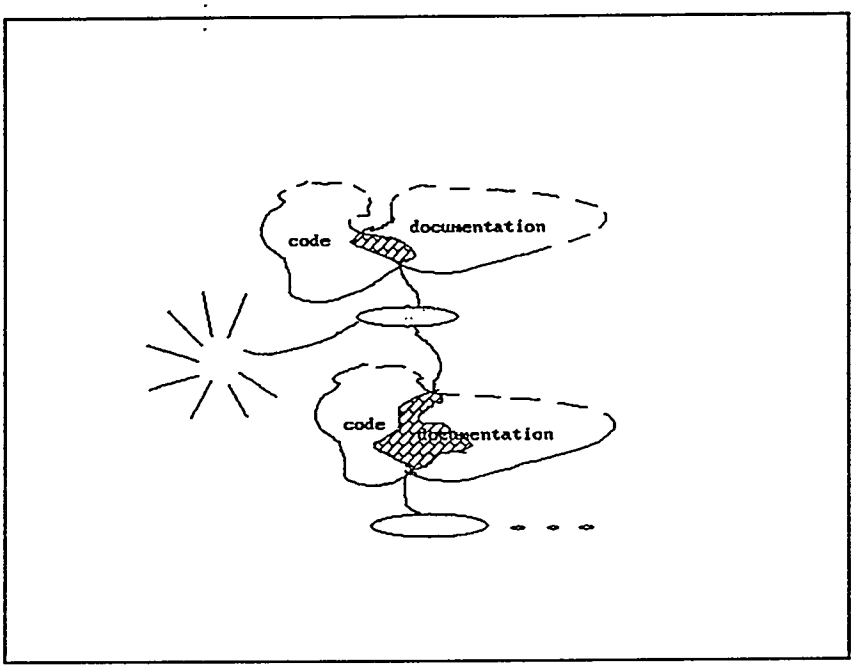

Fig. 2. Iterative Progression

Success!!

Effective teamwork produces a comprehensive package of software and documentation, as well as a more effective and efficient software development team. Through Circling maneuvers, a Bold Thrust, and Iterative Progression, the final product (though fleeting, because the next version is right behind it) arrives.

Computer documentation, like any formal writing, is never completed in "one fell swoop." The process of drafting and revising is continuous. Computer documentation differs from creative writing,

however, because it is a collaborative effort by individuals with different skills using different tools, from different backgrounds and experiences, with different knowledge bases, and having different goals. Although good documentation cannot make bad code any better, the technical writer's skill certainly elicits the end-user's appreciation of good code.

Works Cited

Angus, Jeffery Gordon. "Documentation Is Not for Dummies." InfoWorld Feb. 1995: 57+.

Borchers, Jan Oliver. "HyperSource: a Hypermedia Program Development and Documentation System." Karlsruhe, Germany: 1995 (Internet Article).

Dodson, William R. "Secrets of a High-Performing Team: Joint Application Design (JAD) Is Effective When Examined and Implemented in Components." Data Based Advisor Dec. 1994: 46+

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.