Computer-aided documentation

by SAUL ROSENBERG
Riverside Research Institute
New York, New York

ABSTRACT

Current standards for high-quality documentation of complex computer systems include many criteria, based on the application and user levels. Important points common to many systems are: targeting to specific user groups; being complete, concise, and structured; containing both tutorials and reference material; being field-tested; and being timely in appearance relative to the software delivery. To achieve these goals, uniform quality standards should be more vigorously applied, the documentation development cycle should be shortened, more documentation/software help should be available on line, and more user interaction should be solicited.

For future computer systems, the proposal is made that the documentation be “machine comprehensible.” This should be phased in, with the immediate goal being to facilitate user querying for information, and with an ultimate goal of providing a database for “programmer apprentice” artificial-intelligence programs that assist software development. This new functionality will be the result of several trends, including the drastically reduced cost of read-only online random-access storage via video optical disks, the ongoing successes of artificial-intelligence programs when applied to limited application areas, and the ever increasing cost of software programmers.
INTRODUCTION

This paper first discusses desirable standards for documentation that are realizable using current techniques and then discusses goals for documentation to be produced over the next five years using anticipated technological improvements. A quantum jump in documentation capabilities can be expected as the computer becomes able to aid the user in extracting relevant and timely information.

Good documentation is absolutely essential. It is not an accident that the hottest mini- and microcomputer products that have emerged over the past few years are distinguished from their competitors by higher-quality and easier-to-use documentation.

The paper is written from the perspective of a computer user and of a computer-center manager/systems programmer engaged daily in helping scientific/engineering users in a moderately sized computer center.

CURRENT DOCUMENTATION TECHNOLOGY

Good documentation requires careful planning and analysis on a level comparable to the software development and should be written in tandem with the code. Professional writers should be engaged and should have clearly defined responsibilities to ensure that the final product meets uniform standards.

Documentation must be readable. It should follow good English writing practices and flow smoothly. Detailed technical descriptions should be confined to the appropriate technical sections.

Documentation must be targeted at specific user groups such as students, casual users, experienced analysts, and so on. The authors should at all times be conscious of the user's level. Terminology and technical content should be adjusted as required.

Documentation must be complete, containing all the information the user will require. It must be concise, omitting information or background that is not required for a particular application. Extraneous detail clutters manuals, making them harder to use.

A set of manuals should have a common glossary and a combined index to the important keywords over all the manuals. These are especially important when faced with a transition to a new vendor and/or new system.

Documentation must be structured. Each manual must be well organized and have an overall plan that is apparent to the casual reader. The location of material in the document should be predictable to a new user familiar with the general subject matter. Sets of manuals should fit together in a cohesive, relatively non-overlapping fashion. Documentation put together without a guiding plan is often of uneven quality and information content.

Documentation should have an appropriate structure. Different uses of documentation can require radically different structures. In particular, tutorial introductions for beginners should be structured differently than reference material for systems programmers. Briefly:

- Tutorial material should gradually move from introductory concepts toward detailed descriptions. It should have a moderate number of examples in line and refer to a much larger set of examples in an appendix or auxiliary material, such as computer files distributed with the software. Conceptual diagrams should be strategically placed.

- Reference material should be written as an outline, permitting rapid answers to specific questions by concentrating information about particular topics. The reference section can contain many of the examples referred to in the tutorial material. It should also present a compact synopsis of the information for the frequent user.

- Internal-logic documentation should follow the conceptual structure of the program. This will vary based on the application area. For most programs, a top-down approach is most valuable for showing logic, but some programs in which data transactions are of primary importance may require a structure that traces data records and ignores the logical program order. High-level charts are important. The overview they provide is usually more important than the particular form (flow, HIPO, Warnier-Orr, etc.). Detailed flow charts should not be used because they are often too obsolete to be reliable.

References should be made both to specific code portions and to the formal reference document.

- Internal-code documentation must be readable and refer back to the internal-logic documentation. It must be concise and not obstruct the reading of the actual code. Major code sections should be clearly delineated. Unusual or particularly important code techniques should be emphasized. Many of the computer-science structured programming practices that have been developed over the past decade are also highly applicable to documentation.

Documentation must be timely in its appearance. Frequently, a good set of documentation covering a product will not be fully formed until several years after the product's initial introduction into the field. This imposes extra problems in using state-of-the-art software technology. It is recommended the documentation proceed more or less simultaneously with the software development. This helps assure its
timely appearance and helps the developers avoid inconsistencies that are not noticed until the rules are codified onto paper.

Documentation must be updated periodically, both to integrate new material and to correct mistakes. Areas that have been found to be confusing to many users (based on a user feedback mechanism) should be clarified or rewritten.

A new method for updates to documentation is required. Many manuals are used that are not the latest versions. The causes for this problem relate to the high distribution cost of paper, which makes it impractical to circulate entire new sets of manuals with new software releases. The update notification process is often haphazard, having to filter through several layers of people, from the vendor's site through the user’s. Finally, many users do not bother with regular insertion of the updates into manuals owing to the menial and time-consuming nature of the job.

The new update method should have a faster cycle. Examples and reference material in the documentation are sometimes incorrect. For most reputable vendors this is not a frequent situation, but when it does occur it can send users in circles until the problem is resolved. Long document-update cycles cause the same errors to be repeated by additional users even after a problem has been identified.

Documentation must be field-tested. This is standard practice for new software products and should be applied to new software documentation as well. Usually, new products are placed for Beta testing at experienced sites that can detect bugs in the code and produce their own workarounds. The problem is that these sites typically have person-to-person contact with the vendor and thus do not rely primarily on the written documentation. The suggestion is that software and documentation must also be placed at “inexperienced” sites, and records be kept of the user documentation queries.

Feedback from users must be encouraged. It is vital for vendor employees to be in periodic contact with real end users to learn what is required in the field. Feedback also helps focus attention on obscure sections and is a source of ideas for new manuals. If resources permit, all customer letters should be answered.

REALIZABLE NEW DOCUMENTATION TECHNIQUES

The preceding discussion mentioned many standards that good documentation must meet. Accomplishing this is a highly labor-intensive process, absorbing quality staff resources. Fortunately, continuing and new trends in computer hardware and software technology can make feasible in the near future (the next five years) techniques that will enhance the quality and speed the distribution of documentation.

Problems with Paper-Publishing Cycles

Current documentation techniques are built along standard paper-publishing technology. This leads to a relatively long documentation production and printing cycle before the material appears in the field and makes very expensive the updating and enhancement of the information. It also makes it impossible to tailor the documentation to a specific user's questions.

These problems are typically alleviated via the user’s resident systems programmer. His/her job involves reading most of the manuals, knowing where information is located, reading bulletins about updates and new features, and having a reservoir of experience and knowledge to draw on in applying software facilities. While systems programmers will be essential in the foreseeable future, many of the lower-level, straightforward tasks can and should be automated. The result will be friendlier computers that are more tolerant of user mistakes and are able to tailor information to the specific needs of the user.

Software and Hardware Advances Applicable to Documentation

Advances have been made in the following areas:

- Cheap bulk read-only online random-access storage
- Codification and successful implementation of artificial-intelligence techniques in circumscribed applications
- Development of programmer apprentice artificial-intelligence programs

Commercial companies are beginning to interface videodisk systems to computers. These are being used in educational-training centers and inventory-parts applications among others. Capacities of 15,000 to 50,000 pages of text seem achievable using current hardware. These pages can be randomly accessed and are “printed” on a cheap-to-reproduce vinyl disk. These should be used for distribution of system manuals.

Immediate Benefits—Up-to-Date Manuals

The immediate benefit is a consistent set of up-to-date manuals for all the products with each new release. Since videodisk manuals are relatively cheap (say $20-$50, compared to $500-$1000 per full set for many computer systems), they can be distributed automatically to all users.

Interim Benefits—Query Programs

The important interim benefit lies in the machine readability of the documentation. Many current systems are considered “friendly” because, among other things, they have consistently implemented online “Help” documentation throughout their command syntax. The information on line today is usually only a fraction of what could be available, considering that most new manuals are prepared using word processors. The cheap bulk storage approach can make most of this information available and make it amenable to analysis by programs.

Semi-intelligent query programs are already in use today in online database systems. An example is a medical pharmacological database that advises doctors on drug selection, tolerances, and side effects. The rapid growth that has occurred
in these services testifies to the viability and friendliness of query programs. It is time these techniques were applied to computer software. These query programs would allow a user to skim documentation and receive tailored extracts relevant to his/her immediate needs.

Note that general-purpose comprehension of all possible topics written in English is not required. Many artificial-intelligence programs perform very successfully in narrow fields of knowledge. Surely the computer field is eligible, considering the enormous efforts expended to make computer systems rigorous in their definitions, and predictable in their execution.

The query programs' capability to bypass irrelevant information and examples should not be undervalued. They can concentrate information based on what a particular user is requesting and based on his/her recent history of queries. Such programs also permit writing one "document" with all the information and having the query program present different levels of detail based on the applicability to the request on hand and the user's background. Currently, documentation writers must restrain the numbers and types of examples used to avoid cluttering the document and to avoid providing, for example, FORTRAN code to COBOL programmers. Query programs can select the appropriate ones.

Aspects of these query programs would be similar to Computer-Aided Instruction (CAI) programs, such as Control Data Corp's Plato. Indeed, it would seem reasonable that both types of programs could draw from the same database.

The online availability of examples makes it possible to produce programs that extract and retest the examples, to confirm both the continued functioning of the software and the correctness of the examples. This will help to lessen the historical problem of differences between the published reference-manual standard and the actual implementation standard.

System-error messages can refer to specific portions of reference material on code in error and, if requested, quote chapter and verse on possible causes.

Long-Term Benefits—Programmer Apprentice Programs

In the long term (five to ten years), the availability of documented structures for access by computer programs will form a database usable by high-level artificial-intelligence programs. These are being developed to aid an analyst in formulating programs and to perform much of the mechanics of coding. They will respond to high-level commands, such as "use a quick-sort on the credit records, and then merge them with the master file." These programs will undoubtedly contain high-level abstract descriptions of basic computer-science algorithms. It is unlikely they will be able to contain code fragments to implement all possible variations of them. When faced with applying a specific algorithm, these programs will have to formulate many of the same questions that a person would in selecting language features to implement an idea.

Automatic recording of the high-level commands during the program-generation process would also be a start toward the internal program documentation discussed earlier. Separation of the algorithm descriptions from the language-implementation details would allow use of new language features as they become available, allow people to give advice to the program on which features are more reliable, and permit comparison of new computer languages against previously developed realistic programs.

The online availability of documentation can also lead to programs that can respond to specific questions, such as "What will this language construct do when this parameter has this value?" or its inverse, "Given this behavior, what were the most likely inputs to this language construct?"

Implementation Considerations

These new features do not depend on immediate integration of video disks in particular. Another plausible candidate is the large-capacity "write-once" laser optical recording disk. More conventionally, rotating magnetic-disk storage costs have been dropping so consistently each year that dedicating say 5% of the total available (in typical moderate to large systems) would be sufficient to begin implementing many of these ideas. With time, this would apply to smaller systems also. The disk tradeoff would be amply repaid by the lowered software costs.

The full implementation of these methods will require considerable document writer/programmer/educator time. Furthermore, when operational, the system will use nontrivial CPU power. However, over a ten-year minimum life (for the more durable languages, such as FORTRAN, PASCAL, BASIC, C, and various well-known operating systems), such an investment would be repaid.

SUMMARY

The trend is clear toward low-cost bulk storage becoming commonly available. This fact, together with the increasing programmer shortage, should lead to the exploration and implementation of as many programmer aids as possible.

Good documentation is vital to the success of software products. By using well-known, current principles, vendors can improve their manuals. Design and writing of documentation should be an integral, simultaneous part of software development. Updates to manuals should be distributed on a regular basis, as frequently as needed.

New techniques that apply the computer's potential for artificial intelligence in specialized areas should make it feasible to query/abstract from documentation databases. This will instrumentally increase the friendliness and usability of systems and ultimately be a fundamental part of the knowledge base for programmer apprentice programs that assist in program development.

REFERENCES
