**SYSTEM DOCUMENTATION**

**INTRODUCTION**

All large software development projects, irrespective of application, generate a large amount of associated documentation. For moderately sized systems, the documentation will probably fill several filing cabinets; for large systems, it may fill several rooms. A high proportion of software process costs is incurred in producing this documentation. Furthermore, documentation errors and omissions can lead to errors by end-users and consequent system failures with their associated costs and disruption. Therefore, managers and software engineers should pay as much attention to documentation and its associated costs as to the development of the software itself.

The documents associated with a software project and the systems being developed have a number of associated requirements:

1. They should act as a communication medium between members of the development team.

2. They should be a system information repository to be used by maintenance engineers.

3. They should provide information for management to help them plan, budget and schedule the software development process.

4. Some of the documents should tell users how to use and administer the system.

Satisfying these requirements requires different types of document from informal working documents through to professionally produced user manuals.

Software engineers are usually responsible for producing most of this documentation although professional technical writers may assist with the final polishing of externally released information.

**Process and Product Documentation**

For large software projects, it is usually the case that documentation starts being generated well before the development process begins. A proposal to develop the system may be produced in response to a request for tenders by an external client or in response to other business strategy documents. For some types of system, a comprehensive requirements document may be produced which defines the features required and expected behavior of the system. During the development process itself, all sorts of different documents may be produced – project plans, design specifications, test plans etc.

It is not possible to define a specific document set that is required – this depends on the contract with the client for the system, the type of system being developed and its expected lifetime, the culture and size of the company developing the system and the development schedule that it expected.

However, we can generally say that the documentation produced falls into two classes:

1. ***Process documentation****:* These documents record the process of development and maintenance. Plans, schedules, process quality documents and organizational and project standards are process documentation.

2. ***Product documentation****:* This documentation describes the product that is being developed. System documentation describes the product from the point of view of the engineers developing and maintaining the system; user documentation provides a product description that is oriented towards system users.

Process documentation is produced so that the development of the system can be managed. Product documentation is used after the system is operational but is also essential for management of the system development. The creation of a document, such as a system specification, may represent an important milestone in the software development process.

**Process documentation**

Effective management requires the process being managed to be visible. Because software is intangible and the software process involves apparently similar cognitive tasks rather than obviously different physical tasks, the only way this visibility can be achieved is through the use of process documentation.

Process documentation falls into a number of categories:

1. ***Plans, estimates and schedules****:* These are documents produced by managers which are used to predict and to control the software process.

2. ***Reports****:* These are documents which report how resources were used during the process of development.

3. ***Standards****:* These are documents which set out how the process is to be implemented. These may be developed from organizational, national or international standards.

4. ***Working papers****:* These are often the principal technical communication documents in a project. They record the ideas and thoughts of the engineers working on the project are interim versions of product documentation, describe implementation strategies and set out problems which have been identified. They often, implicitly, record the rationale for design decisions.

5. ***Memos and electronic mail messages:*** These record the details of everyday communications between managers and development engineers.

The major characteristic of process documentation is that most of it becomes outdated. Plans may be drawn up on a weekly, fortnightly or monthly basis. Progress will normally be reported weekly. Memos record thoughts, ideas and intentions which change.

Although of interest to software historians, much of this process information is of little real use after it has gone out of date and there is not normally a need to preserve it after the system has been delivered. However, there are some process documents that can be useful as the software evolves in response to new requirements.

For example, test schedules are of value during software evolution as they act as a basis for re-planning the validation of system changes. Working papers which explain the reasons behind design decisions (design rationale) are also potentially valuable as they discuss design options and choices made. Access to this information helps avoid making changes which conflict with these original decisions. Ideally, of course, the design rationale should be extracted from the working papers and separately maintained. Unfortunately this hardly ever happens.

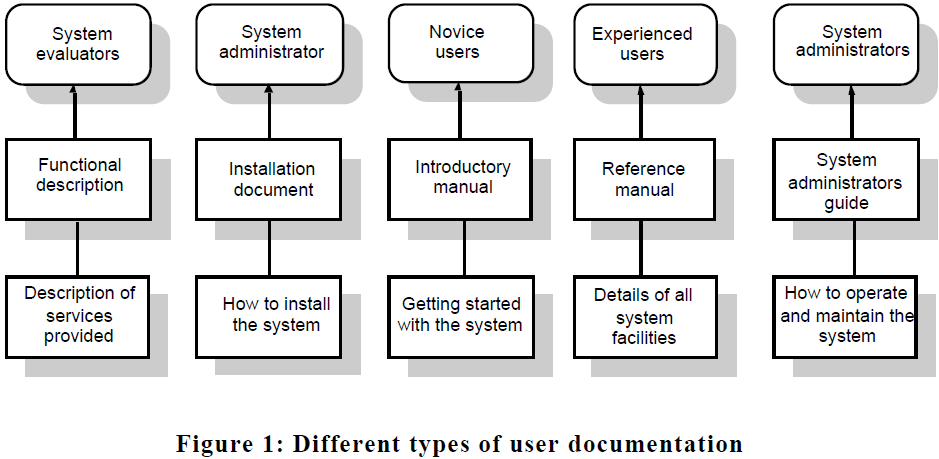
**Product documentation**

Product documentation is concerned with describing the delivered software product. Unlike most process documentation, it has a relatively long life. It must evolve in step with the product which it describes. Product documentation includes user documentation which tells users how to use the software product and system documentation which is principally intended for maintenance engineers.

**User Documentation**

Users of a system are not all the same. The producer of documentation must structure it to cater for different user tasks and different levels of expertise and experience. It is particularly important to distinguish between end-users and system administrators:

1. End-users use the software to assist with some task. This may be flying an aircraft, managing insurance policies, writing a book, etc. They want to know how the software can help them. They are not interested in computer or administration details.



2. System administrators are responsible for managing the software used by end-users. This may involve acting as an operator if the system is a large mainframe system, as a network manager is the system involves a network of workstations or as a technical guru who fixes end-users software problems and who liaises between users and the software supplier.

The *functional description* of the system outlines the system requirements and briefly describes the services provided. This document should provide an overview of the system. Users should be able to read this document with an introductory manual and decide if the system is what they need.

The *system installation document* is intended for system administrators. It should provide details of how to install the system in a particular environment. It should contain a description of the files making up the system and the minimal hardware configuration required. The permanent files which must be established, how to start the system and the configuration dependent files which must be changed to tailor the system to a particular host system should also be described. The use of automated installers for PC software has meant that some suppliers see this document as unnecessary. In fact, it is still required to help system managers discover and fix problems with the installation.

The *introductory manual* should present an informal introduction to the system, describing its ‘normal’ usage. It should describe how to get started and how end-users might make use of the common system facilities. It should be liberally illustrated with examples. Inevitably beginners, whatever their background and experience, will make mistakes. Easily discovered information on how to recover from these mistakes and restart useful work should be an integral part of this document.

The *system reference manual* should describe the system facilities and their usage, should provide a complete listing of error messages and should describe how to recover from detected errors. It should be complete. Formal descriptive techniques may be used. The style of the reference manual should not be unnecessarily pedantic and turgid, but completeness is more important than readability.

A more general *system administrator’s guide* should be provided for some types of system such as command and control systems. This should describe the messages generated when the system interacts with other systems and how to react to these messages. If system hardware is involved, it might also explain the operator’s task in maintaining that hardware. For example, it might describe how to clear faults in the system console, how to connect new peripherals, etc.

As well as manuals, other, easy-to-use documentation might be provided. A quick reference card listing available system facilities and how to use them is particularly convenient for experienced system users. On-line help systems, which contain brief information about the system, can save the user spending time in consultation of manuals although should not be seen as a replacement for more comprehensive documentation. I briefly discuss on-line documentation in a later section in this paper.

**System Documentation**

System documentation includes all of the documents describing the system itself from the requirements specification to the final acceptance test plan.

Documents describing the design, implementation and testing of a system are essential if the program is to be understood and maintained. Like user documentation, it is important that system documentation is structured, with overviews leading the reader into more formal and detailed descriptions of each aspect of the system.

For large systems that are developed to a customer’s specification, the system documentation should include:

1. The requirements document and an associated rationale.

2. A document describing the system architecture.

3. For each program in the system, a description of the architecture of that program.

4. For each component in the system, a description of its functionality and interfaces.

5. Program source code listings. These should be commented where the comments should explain complex sections of code and provide a rationale for the coding method used. If meaningful names are used and a good, structured programming style is used, much of the code should be self-documenting without the need for additional comments. This information is now normally maintained electronically rather than on paper with selected information printed on demand from readers.

6. Validation documents describing how each program is validated and how the validation information relates to the requirements.

7. A system maintenance guide which describes known problems with the system, describes which parts of the system are hardware and software dependent and which describes how evolution of the system has been taken into account in its design.

A common system maintenance problem is ensuring that all representations are kept in step when the system is changed. To help with this, the relationships and dependencies between documents and parts of documents should be recorded in a document management system as discussed in the final part of this paper.

For smaller systems and systems that are developed as software products, system documentation is usually less comprehensive. This is not necessarily a good thing but schedule pressures on developers mean that documents are simply never written or, if written, are not kept up to date. These pressures are sometimes inevitable but, in my view, at the very least you should always try to maintain a specification of the system, an architectural design document and the program source code.

Unfortunately, documentation maintenance is often neglected. Documentation may become out of step with its associated software, causing problems for both users and maintainers of the system. The natural tendency is to meet a deadline by modifying code with the intention of modifying other documents later.

Often, pressure of work means that this modification is continually set aside until finding what is to be changed becomes very difficult indeed. The best solution to this problem is to support document maintenance with software tools which record document relationships, remind software engineers when changes to one document affect another and record possible inconsistencies in the documentation.

**Document Quality**

Unfortunately, much computer system documentation is badly written, difficult to understand, out-of-date or incomplete. Although the situation is improving, many organizations still do not pay enough attention to producing system documents which are well-written pieces of technical prose.

Document quality is as important as program quality. Without information on how to use a system or how to understand it, the utility of that system is degraded. Achieving document quality requires management commitment to document design, standards, and quality assurance processes. Producing good documents is neither easy nor cheap and many software engineers find it more difficult that producing good quality programs.

**The IEEE standard for user documentation**

The first IEEE standard for user documentation (IEEE, 1987) was produced in 1987 and, at the time of writing, a new draft of this standard is being prepared for publication (IEEE, 2001).

Like all standards, this standard encapsulates wisdom and experience about software documentation and proposes a structure for user documentation. Using this structure as a basis, the standard discusses the content of software user documentation and proposes formatting standards for these documents.

I have already covered the documentation structure proposed by the latest version of the standard. To illustrate the formatting advice in the standard, here are some quotations from the current draft standard of good practice:

*The documentation should be provided in media and formats that allow its use by those with vision, hearing or other physical limitation.*

*A description of how to print the electronic documentation should be included in both the electronic and the printed documentation.*

*Because some users cannot distinguish between colors, documentation should provide text cues rather than using colors such as red and green as the only way to convey meaning.*

*Warnings, cautions and notes shall be displayed in a consistent format that is readily distinguishable from ordinary text or instructional steps.*

*Documentation formats for user-entered commands or codes shall clearly distinguish between literals (to be input exactly as shown) and variables (to be selected by the user).*

*Illustrations that accompany text should appear adjacent to their first reference in the text so that the associated text and illustration can be viewed simultaneously.*

You can see from these that the standard is helpful without being proscriptive and therefore different conventions used by different companies and organizations can be accommodated.

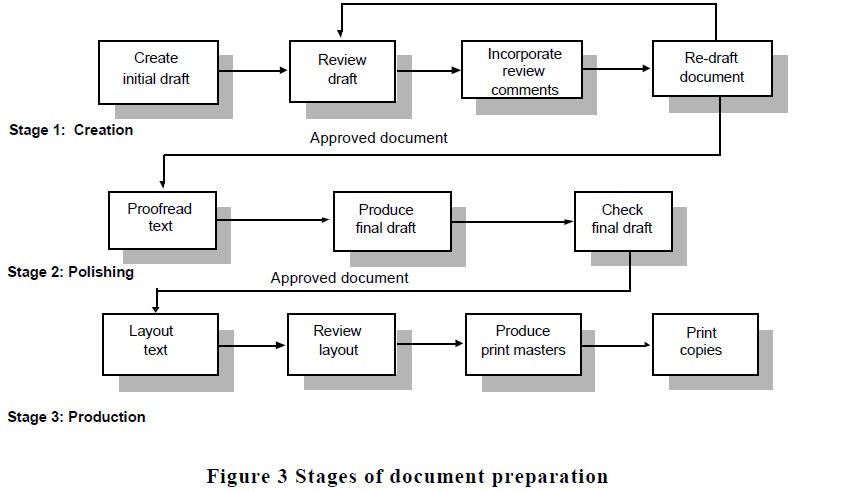
Like all standards, this documentation standard has to be adapted to the local situation where it is used. These should instantiate the advice in the standard to the local situation and define the specific structures and formats that should be used.

**Document Preparation**

Document preparation is the process of creating a document and formatting it for publication. Figure 3 shows the document preparation process as being split into 3 stages namely document creation, polishing and production. Modern word processing systems are now integrated packages of software tools that support all parts of this process. However, it is still the case that for the highest-quality documents, it is best to use separate tools for some preparation processes rather than the built-in word processor functions. The three phases of preparation and associated support facilities are:

1. ***Document creation:***The initial input of the information in the document. This is supported by word processors and text formatters, table and equation processors, drawing and art packages.

2. ***Document polishing:***This process involves improving the writing and presentation of the document to make to make it more understandable and readable. This involves finding and removing spelling, punctuation and grammatical errors, detecting clumsy phrases and removing redundancy in the text. The process may be supported by tools such as on-line dictionaries, spelling checkers, grammar and style checkers and style checkers.



3. ***Document production:***This is the process of preparing the document for professional printing. It is supported by desktop-publishing packages, artwork packages and type styling programs.

As well as these tools to support the document production process, configuration management systems, information retrieval systems and hypertext systems may also be used to support document maintenance, retrieval and management.

Modern word processing systems are screen based and combine text editing and formatting. The image of the document on the user’s terminal is, more or less, the same as the final form of the printed document. Finished layout is immediately obvious. Errors can be corrected and layout improved before printing the document. However, programmers who already use an editor for program preparation may sometimes prefer to use a separate editor and text formatting system.

Text formatting systems such as Latex interpret a layout program specified by the document writer. Layout commands (often chosen from a standard, definable command set) are interspersed with the text of the document. The text formatter processes these commands and the associated text and lays the document out according to the programmer’s instructions.

Text formatting systems can look ahead at the text to be laid out so can make better layout decisions than word processing systems whose working context is more restricted. Because the commands are really a programming language, programmers often prefer them to word processors but other, non-technical users usually find them more difficult to use.

The major disadvantage of text processors, once their programming has been mastered, is that they do not provide an immediate display of the output they produce. The user must process the text (this may take several minutes) then display the output using a preview package. If an error is discovered, it cannot be fixed immediately. The original source must be modified and the preview process repeated. Thus, although they can result in higher quality documents, most users find text formatters more inconvenient than word processors.

The final stage of document production is a skilled task which, for documents with large print runs, should be left to professional printers. However, desktop publishing (DTP) systems and graphics systems that support scanning and processing photographs and artwork are now widely available. These have revolutionized document production. DTP systems partially automate the layout of text and graphics. They allow very fine-grain control over the layout and look of a document and can be used by engineers to produce finished system documentation.

The advantage of using a desktop publishing system is that the cost of producing high-quality documents is reduced because some of the steps in the production process are eliminated. Even documents which are produced in small numbers can be produced to a high standard. The disadvantage of using desktop publishing systems is that they do not automate the skills of the graphic designer. Their seductive ease-of-use means that they are accessible to unskilled users who may produce unattractive and badly designed documents.

An enormous number of documents are produced in the course of a project and these need to be managed so that the right version of the document is available when required. If a project is distributed, copies of documents will be produced and stored at different locations and it is very important to maintain a ‘master file’ of documents which contains the definitive versions of each document.

This helps minimize a very common problem that arises when users of a document make mistakes because they are not working from the current version of a document.

Each document should have a unique record and this can be used as a key in a document database record. However, retrieval by other fields such as the title and author should also be supported.

The basic problem with managing documents using a file system to store the documents and a database management system to maintain document information is that users have to be disciplined in the way they use the system. They must ensure that they check out a copy of the document from the system each time they need it rather than use a local copy on their computer or the copy that they have printed. In practice, achieving this level of discipline is difficult and errors are always likely to arise.

In very large projects, specialized document management systems may be used that integrate the storage of the documents and the maintenance of document information. Document management software allows related documents to be linked, maintains records of who has checked out documents, may support the compression and de-compression of document text and provides indexing and information retrieval facilities so that documents can be found. Document management systems may also include version management facilities so that different document versions may be maintained.

REFERENCES

Dix, A., Finley, J., Abowd, G. and Beale, R. 1998. *Human-Computer*

*Interaction, 2nd ed.* London: Prentice-Hall.

Garg, P. K. and Scacchi, W. 1990. ‘A hypertext system to maintain software

life-cycle document’. *IEEE Software*, **7** (3), 90–8.

IEEE, 1987. *IEEE Standard for Software User Documentation*, IEEE-Std1063-

1987. New York: Institute of Electrical and Electronics Engineers.