INTRODUCTION TO SOFTWARE DOCUMENTATION

Alberto Aimar
IPT Group, IT Division
CERN, Geneva, Switzerland

Abstract

Documentation and information systems are needed in all phases of the development of a software product: from the user requirements phase, through design and implementation, until delivery and maintenance of the product. For this reason documentation should be designed, managed, produced, published and maintained.

These activities are often seen as second priority or a burden, but are extremely useful. They become easier when performed in a coherent environment that allows production of adequate artifacts and their publication in paper and electronic forms.

This summary provides some practical guidelines and references to resources that can help in the production of simple but effective documentation.

1. INTRODUCTION

During the development of a software project, documentation is undoubtedly the activity that we all like to do the least. On the other hand, often programs are simply thrown away and rewritten because they are no longer under control and there is not enough information to understand them. This summary is an attempt to provide recipes that reduce the pain of producing that minimal amount of documentation that will help other people to work on the same project (internal documentation) and that will allow somebody outside the project to use the product (user documentation).

2. DOCUMENTATION AND THE SOFTWARE LIFECYCLE

The process of producing software is made up of several activities (project management, configuration management, etc.) and phases (requirements, design, coding, etc.). Such activities and phases are always present in the lifecycle, independently on the development model chosen for the project. Figure 1 shows the “waterfall” and the “V” models, but there are also other models such as the “iterative” and the “rapid prototyping” models.

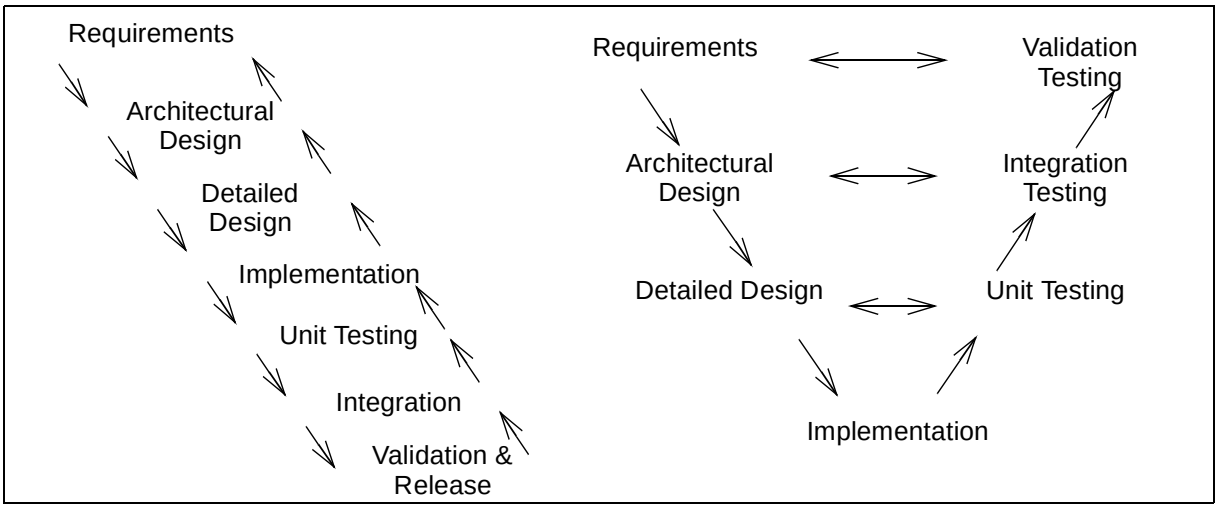

Fig. 1 The waterfall and the V software development models.

The purpose of this summary is not to discuss which model is the best: different models suit better different organizations or projects; what all these models have in common, and I intend to talk about this here, is that any software product without solid documentation will not last long or will cost much more than expected to maintain. Without information on the development process, implementation choices, test criterias, etc. nothing is clear and the software project always ends up in an unmanageable mess. What I call "solid documentation" is not the production of tons of paper but the updated and structured amount of information needed to keep the project under control.

Some projects will focus more on some phases and skip others; but documents must be produced during the development of a project. As shown in Fig. 2, all phases are done by different people (sometimes the same people, if they cover more than one role in the project) using different tools to produce different kinds of documents. In fact the arrows in Fig. 1 are the documents that are flowing in the software development process.

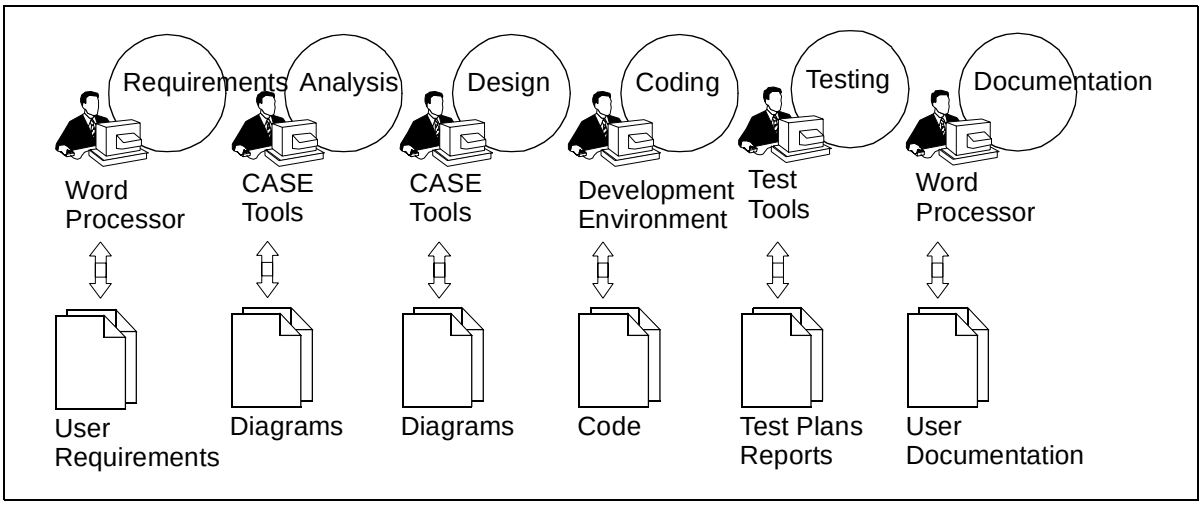

Fig. 2 Some of the documents produced in the software development lifecycle.

3. WHY DO WE NEED BETTER DOCUMENTATION?

There is clear fact: writing documentation is not fun: it is a need. As Fig. 3 shows, in a software project 60% of the time is spent to understand and redo (27%+33%) existing programs. This is often due to the fact that the documents supporting the project are inadequate, not detailed enough or obsolete.

Component	Percentage	
Program understanding	27%	Work 40%
Requirements analysis	24%	
Design	18%	
Implementation	18%	Rework 33%
Testing	13%	

B.Grady, An investment model for software process improvement.

Technical report

Hewlett-Packard Corporate Engineering

Fig. 3 The Grady experience: cost distribution of software development.

Poor documentation increases the time not spent in producing new features, poorly documented products have difficulty in finding users and devolve slowly until they become "untouchable" and need to be partially rewritten or simply thrown away. Good documentation, on the other hand, helps people

to quickly integrate in the project, allow proper project management because problems are highlighted early and the development can be controlled adequately.

Inside a project the documentation is the glue between phases and among the people working together, at different artifacts, but with the same project goal. The quality of a product is something that must be built, it is not something that comes for free and without effort.

The documentation of the early phases of the project (requirements and analysis) is often neglected but actually it is in these early phases where correcting wrong choices is much cheaper. As Fig. 4 shows, a wrong choice or a forgotten detail found in the implementation phase costs up to hundred times more than if it had been found in the analysis phase.

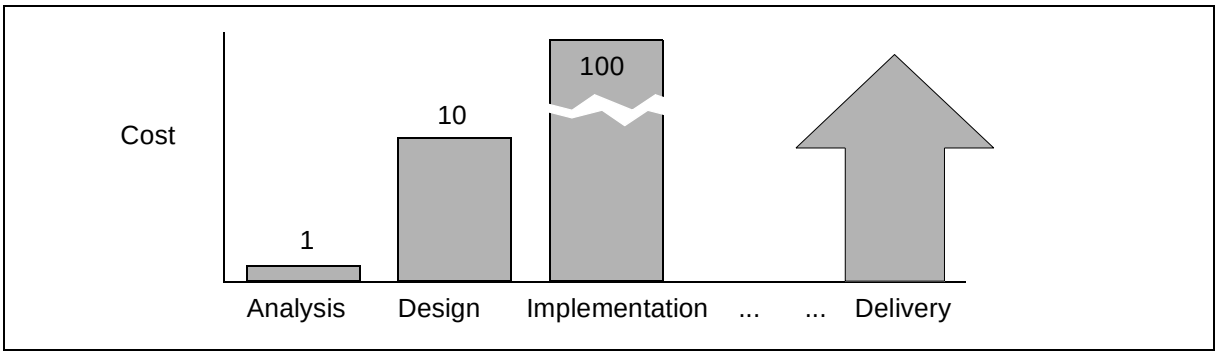

Fig. 4 Cost of a change in the different phases of the software lifecycle.

4. SOFTWARE ENGINEERING DOCUMENTATION

Software engineering is still a very young field compared to others, such as civil engineering or mechanical engineering. Would it be thinkable to build a bridge or a new car model without properly describing its components? And to proceed immediately to build bits and pieces thinking to put them together afterwards, without a real design and assembly plans? No! So, why do we do it in software?

There are several standards defined by the main software engineering institutes (for more information see Section 8.3) but the purpose here is to highlight very briefly the characteristics of the different documents (the grey spots Fig. 5) and not to present all the details.

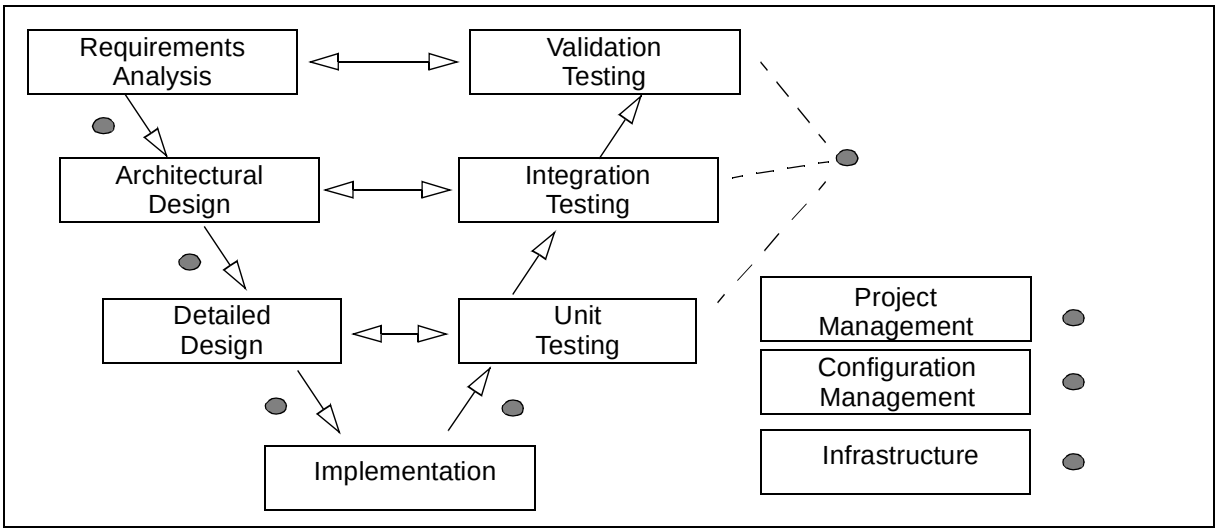

Fig. 5 Basic documents in the software development lifecycle.

4.1 Requirements documents

The requirements documents must describe the product, the functionalities and the quality expected. Wrong requirements will give the wrong product, fuzzy requirements will cause constant instability and changes.

In order to capture all aspects of the product, the requirements are usually separated in:

- functional requirements (features, deliverables, logic, documents, etc.);
- design requirements (quality, portability, performance, etc.);
- operational requirements (maintenance, security, support, etc.).

Each user requirement is formed by:

- a unique identifier;
- a description that must be unambiguous and clearly stated;
- attributes that complete the description, such as priority, examples, source, how to test the requirement, its acceptance criteria, etc.

See also the "User Requirements" paper in these proceedings.

4.2 Architecture documents

The architecture of a software artifact is its description in terms of all its components and the interfaces among the components. Usually the overview is done graphically (example in Fig. 6) but then the details must be specified in detailed text, in an architecture document.

Each component must have a unique identifier, the description of all its functionalities and its decomposition, if necessary, in sub-components. The interfaces are extremely important for obtaining modular applications, an efficient partition of the work in the team and parallel development. Each interface must be specified exactly in terms of data exchanged, direction of the data, which component triggers the data exchange, the detailed description of the data structures involved.

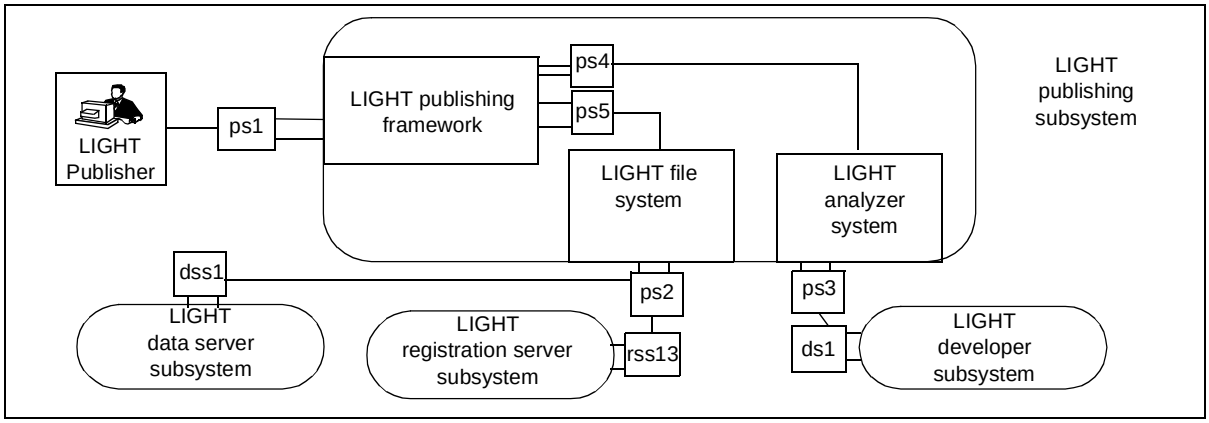

Fig. 6 Graphic overview of a component and its context.

4.3 Detailed Design

Once the components are defined, it is necessary to describe all the details of the component; usually for this purpose an analysis and design methodology (for instance OMT, Booch, UML, ROOM for object-oriented modeling) is used and its choice depends on the organization, the project infrastructure and the language platform chosen. Depending on the type of component these documents usually consist of use cases or scenarios, object or data models, dynamic or logic behavior models and so on.

4.4 Implementation documents

These are the result of the favorite activities of every developer: coding, debugging, etc. But code is not all, it must be the implementation of the design choices, must be adequately commented (I know these things are obvious but often they are not done!). For instance, comments should focus on the difficult

parts of the code, not simply parrot the easy bits of code; they should also explain the logic and why such logic is chosen.

4.5 Test documents

The product must be tested in order to make sure that it is "the" product we wanted to obtain; usually the test documents are done in parallel with coding and they use as input the requirement and the design documents. Tests, just like the application, must be studied, designed and implemented.

Usually the minimal set of tests are:

- a test plan that describes the global strategy, features and test modules;
- test cases each verifying a feature of a component, with all the commands needed to run the test, and how to setup the test environment;
- test reports which present the results of the execution of the individual tests.

4.6 Project management documents

All the tasks of the project are planned and tracked by the project management activities (see "Project Management and Tracking" in these proceedings). These documents should at least contain:

- a project plan, with the tasks description, resource allocations and dependencies;
- project scheduling and tracking documents, with progress status, Gantt and Milestone Trend charts (see [4], pg. 328);
- regular project reports, both for the team members and for the project customers.

4.7 Configuration management documents

The configuration management documents describe all the configuration items of the project; every artifact (document) of the project phases and activities (requirements, design, code, tests, etc.). They also describe the repository where all these documents are stored, how they can be modified, how they are versioned and released. See more details in "Configuration Management" in these proceedings.

4.8 Project infrastructure document

The project infrastructure documents are a set of documents that are different for different roles in the project (development infrastructure, management infrastructure, documentation infrastructure and so on). They basically describe how to participate in the project in a given role; therefore they contain information such as:

- all the conventions of the projects;
- which tools to use in the project and how they are used;
- templates and methodologies to follow;
- recommended reading for the project members;
- how to use the configuration system;
- how to report progress to the project manager;
- any other practical needs within the project.

5. USER SOFTWARE DOCUMENTATION

User documentation is not necessarily made of paper, it is all the information that is presented to the user: manuals, system or error messages, on-line help, Web sites, comments in the code examples, multi-media tutorials, etc. The appropriate tools should be used, such as word processors, Web builders, screen capture-replay tools and tutorial builders.

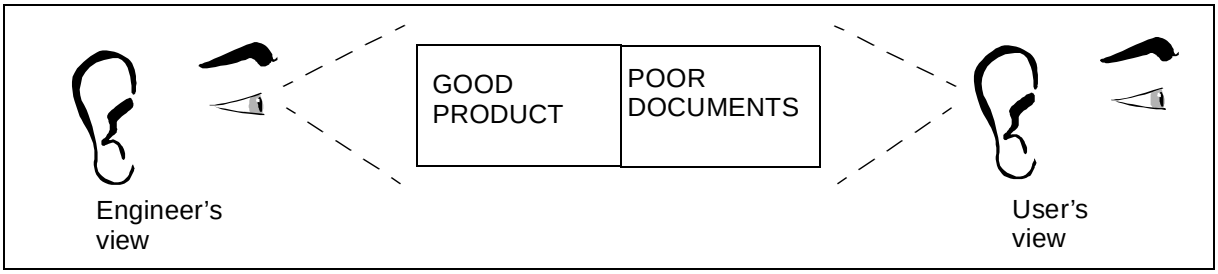

Fig. 7 User's and engineer's views of a product.

When producing user documentation you must always think of the user's point of view (Fig. 7). It is not enough to build a good product, it is also important to have an adequate documentation: your product will never get a second chance to make the first impression!
User documentation must not become a prose exercise, it should have the maximum amount of information with the minimum amount to read; the documentation is an important deliverable and it is developed in the same way as the application, with phases of analysis, design, editing, testing, etc. Documentation must be reviewed and tested with a few users, suggestions collected and new versions of the documents produced (and the changes should be highlighted).

There are a few immediate simple criteria to judge the quality of user documentation:

- bad documentation: engineer-centred, inappropriate layout and fonts, too wordy and verbose, too many cross-references and an inadequate physical format;
- good documentation: task oriented and user-centred, different formats and fonts for different purposes and user levels, allowing random access via indices and table of contents, complete textual information with a lot of graphs, pictures, etc.

6. SUGGESTIONS ON SOFTWARE DOCUMENTATION

6.1 Elements of good documentation

Try to convey information in a form other than text and prose use elements like:

- structured writing (with heading, small blocks, lists, sequences, etc.);
- flowcharts to show graphically the structure and the behavior of the product;
- trees and graphs to show the structure of the data
- lots of examples and playscripts that the user can use as such and understand how to proceed in practice;
- plenty of pictures and screen dumps to show, for instance, how to use the graphic interfaces.

6.2 The STOP method

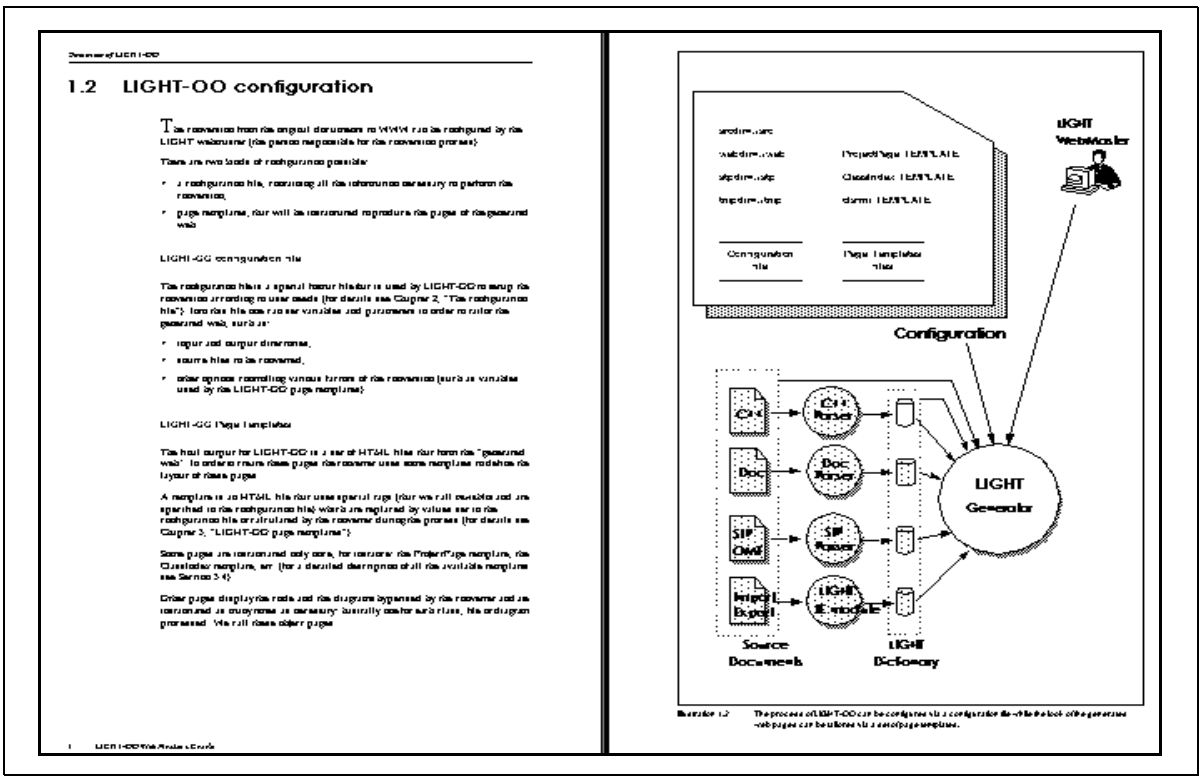

Fig. 8 The STOP method.

In the IPT group at CERN we have written many documents using the STOP method and have experienced how positively the user welcomes them. This method was invented at Hughes Aircraft Company (1960) and it is widely used in several companies both for internal and public documentation (NCR, Texas Instruments, IBM and others).

This method has several advantages for the reader and for the writer. The principles of the method are simple:

- break the information you want to convey into two-page components;
- one concept per component;
- text on the left (500 words) and graphics on the right, both explaining the same concept.

In fact every time the reader turns the page he disconnects and reconnects to the book; "normal" books very often interrupt the reader's attention even in the middle of an explanation. With this method, every time a page is turned a concept is completed and a new one arrives. Text and graphics are using different parts of our brain and some readers prefer one of the two types of presentation; by providing both forms we satisfy all kinds of audience.

For the editor it is also much easier to produce and reuse the material because the document is extremely modular, the graphic pages can be used as viewgraphs for presentations and the text is the track for the verbal part of the presentation.

7. CONCLUSIONS

I hope that in this summary I managed to convey the most important message I wanted to pass to you: if you document your work, you show that you are a mature engineer and a person who cares for colleagues and for the project. Believe me: there is really no point in working like mad and being able to explain and document only a fraction of the work you do.

Sometimes it is not our fault if we do not write documentation, but of the organization for whom we work: it is really a symptom of an organization's immaturity to look at the amount of work its staff does without considering the quality of the documentation that is a part of that work!

So write documentation while you work, build (or look for) your document templates and use them consistently; in this way you will build a set of coherent documents; they are part of the product you are working on. Yes, they are a pain in the neck, but prevent greater troubles to you and to your colleagues.

8. RECOMMENDED READING LIST

8.1 Documentation in general

[1] R.John Brockmann. *Writing better computer user documentation: from paper to hypertext.* John Wiley & Sons, ISBN 0-471-62260-5

[2] R.Low, H.Ford at alt., *Writing User Documentation: A Practical Guide for Those Who Want to Be Read.* Prentice Hall, ISBN 0-13-336835-1

8.2 Documentation for software development

[3] C.Mazza et alt. *Software Engineering Standards.* Prentice Hall, ISBN 0-13-106568-8

[4] C.Mazza et alt. *Software Engineering Guides.* Prentice Hall, ISBN 0-13-449281-1

8.3 Software documentation standards

[5] IEEE 1063: *Standard for Software User Documentation.*

[6] ISO 6592: *Guidelines for the documentation of computer-based application systems.*

[7] ISO 9127: *User documentation and cover information for consumer software packages.*

[8] ESA PSS-05: *Software Engineering Standards.*