

retrieval performance.

Finally, we wish to consider the best way to navigate the proposed document hierarchy. This hierarchy could also be used to fine-tune responses to user queries. While we may not want to prohibit user access to certain levels of detail, it could prove useful to allow a user to limit the areas browsed and the scope of query responses to exclude levels which are too general or too detailed for their purposes.

Future Work

In the future we plan to use this system to conduct user studies. We hope to determine its usefulness to documenters and to what degree this system can improve the quality of information located by users. We also hope to more fully integrate the selection of hypertext terms into the system. The current method is effective, but could become cumbersome. We also plan to investigate the system's usefulness as a method to track documents produced during different phases of the software life cycle. This methodology could be applied to organize and improve access to those documents. As mentioned earlier, we are also considering an index to be used with the hardcopy version of the documentation. A custom filter similar to those provided for hypertext links and cross-references should be provided to facilitate the comprehensive indexing of chosen terms.

Conclusion

We believe that software documentation can be improved by providing guidelines for document construction and an environment to facilitate this process. We have produced a prototype system to explore the features which we believe should be included in a documentation system. Our system consists of an authoring and viewing environment and provides both a mechanism to link a collection of documents using typed hypertext links and the facility to perform keyword searches on the text of the documentation. Much of this mechanism has been automated to enhance consistency and ease the load on the documenter.

We believe that these techniques can improve the usefulness of software documentation by addressing the needs of a wide variety of users and by providing these users with a number of ways to uncover the answers to their questions.

References

1. British Ministry of Defence, "Interim Defence Standard 00-55, The Procurement of Safety Critical Software in Defence Equipment," April 5, 1991.
2. Cybulski, J. L. and Reed, K., "A Hypertext Based Software Engineering Environment," *IEEE Software*, pp. 62-68, March 1992.
3. Delisle, N. M. and Schwartz, M. D., "Contexts--A Partitioning Concept for Hypertext," *ACM Transactions on Office Information Systems*, vol. 5, no. 2, pp. 168-186, April 1987.
4. Devanbu, P., Selfridge, P. G., Brachman, R. J. and Ballard, B. W., "LaSSIE: a Knowledge-based Software Information System," *IEEE Proceedings of 12th International Conference on Software Engineering*, pp. 249-261, 1990.
5. *Frame Developer's Kit for Specific Platforms - UNIX*, Frame Technology Corporation, October 1993.
6. *Frame Developer's Kit Programmer's Guide*, Frame Technology Corporation, October 1993.
7. Garg, P. K. and Scacchi, W., "A Hypertext System to Manage Software Life-Cycle Documents," *IEEE Software*, pp. 90-98, May 1990.
8. Horowitz, E. and Williamson, R. C., "SODOS: A Software Document Support Environment--Its Definition," *IEEE Transactions on Software Engineering*, vol. SE-12, no. 8, pp. 849-859, August 1986.
9. Horowitz, E. and Williamson, R. C., "SODOS: A Software Document Support Environment--Its Use," *IEEE Transactions on Software Engineering*, vol. SE-12, no. 11, pp. 1076-1087, November 1986.
10. Kahle, B. and Medlar, A., "An Information System for Corporate Users: Wide Area Information Servers," *Online Magazine*, pp. 56-60, September 1991.
11. Marchionini, G. and Shneiderman, B., "Finding Facts vs. Browsing Knowledge in Hypertext Systems," *IEEE Computer*, vol. 21, no. 1, pp. 70-80, 1988.
12. Parnas, D. L., van Schouwen, A. J. and Kw an, S. P., "Evaluation of Safety-Critical Software," *Communications of the ACM*, vol. 33, no. 6, pp. 636-648, June 1990.
13. Radio Technical Commission for Aeronautics, "Software Considerations in Airborne Systems and Equipment Certification," Document No. R TCA/DO-178A, March 1985.
14. Soloway, E., Pinto, J., Letovsky, S., Littman, D., Lampert, R., "Designing Documentation to Compensate for Delocalized Plans," *Communications of the ACM*, vol. 31, no. 11, pp. 1259-1267, November 1988.
15. United States of America Department of Defense, "DOD-STD-2167, Military Standard Defense System Software Development," June 4, 1985.
16. Wood, M. and Somerville, I., "An Information Retrieval System for Software Components," *SIGIR Forum*, pp. 11-25, Spring/Summer 1988.

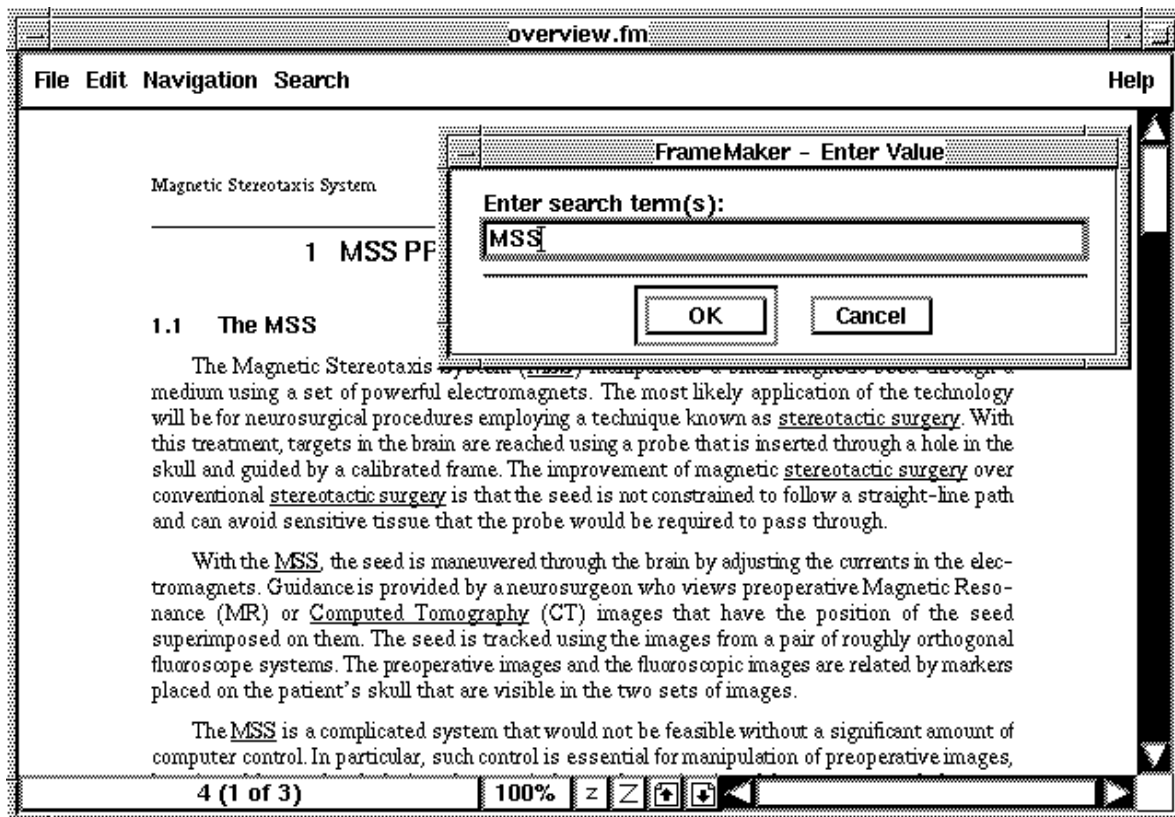


Fig. 5 Search interface as seen by the user.

Hardcopy Version of the Documentation

Much of the navigational capability of the on-line version of the documentation is maintained in the hardcopy version. Each typed hypertext link has an associated cross-reference which tells the destination page of the link. The typing is visible to the hardcopy user through cross reference type and color if the user has a color copy of the documents. The cross reference type is denoted by the numbering scheme employed. Cross references to appendices have the page number format 'A#'. Figure references have the format 'F#'. This numbering convention is useful for differentiating link types if the user has a black and white copy of the hardcopy version, a monochrome monitor or if the user is color-blind.

Obviously, the search engine cannot be used in the hardcopy version - we are considering adding an index to cover as much of this functionality as possible. In our discussion of previous work, we mentioned the problems with predefined keyword searching found in some other systems. This index approach would have many of the same problems: the author would have to attempt to anticipate the needs of the users and desired terms may not be included in the index. This scheme would not be sufficient for all purposes, but would be useful if a user were forced to work only with the hardcopy version of the documentation.

Current Issues

There are a number of problems that remain to be solved. We need to consider the best way to integrate and search source code and other formalisms. We would like to be able to perform more specific searches and searches which are directly applicable to source code, such as locating the point where a variable was declared. There are also CASE tools which can provide data about the source code. The information produced by these tools should also be considered for inclusion.

We also plan to investigate the best way in which to return query responses. Instead of simply returning pointers to documents containing query terms, we are considering using the sentences or paragraphs which contain the search term(s) to produce a composite answer document. Ideally, this document itself would answer the user's question. If not, links would be provided back to the complete documents from which the fragments were extracted. The information included in the fragments should allow the user to determine which returned documents would prove most useful, saving the user from the necessity of exploring all returned documents. It is still unknown which of these choices would prove most useful to the users.

We have the benefit of knowing that we are dealing with software documentation and have the option of specifying document organization and structure. We must determine if these factors can be exploited to improve information

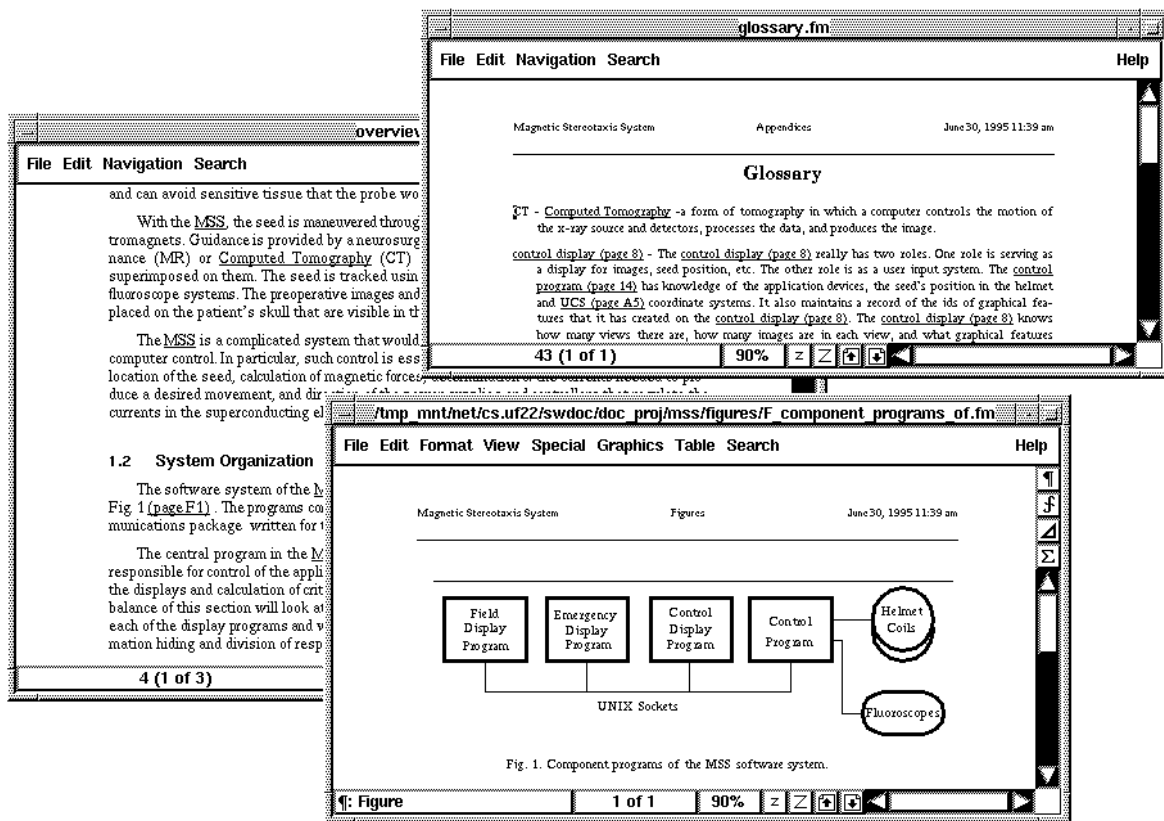


Fig. 4 Example browsing session.

at the same time, allowing the user to look up a definition and view a referenced figure while examining the document of interest. In this way, the user can have a set of relevant documents arranged on the screen to explore a subject area in detail.

In addition to the set of documents, users also have at their disposal an integrated WAIS search engine through which they can enter boolean keyword searches. The search engine was incorporated into the system using the FrameMaker Developer's Toolkit. This provides an intuitive and stable interface to the search engine. See Fig. 5 for an example search session. If desired, the WAIS search engine could be replaced without changing the user interface.

The user begins a query by choosing the 'Search' pull-down window. The user can begin a new search or continue the previous one. The search terms are entered into a pop-up window and then the search is performed. The response currently consists of a list of hypertext links which point to the documents and paragraphs in which the search terms are located. This list is returned in a generated FrameMaker

destination points of these links. The system currently has types defined to denote links to glossary entries, appendices, other sections, source code and figures. The filters provide convenient access to and utilize the standard FrameMaker hyper- and cross-reference utilities. They use the typing information specified in the document templates provided with the documentation system. For each specified term in any document, the filters will convert that term to the appropriate hypertext link or cross-reference. The filters use the information provided by the documenter in the filter configuration files to construct the specified links. Aside from the convenience of creating multiple instances of a link at one time, this process also assures that the links are consistent and exhaustive.

If new hypertext or cross-reference terms are desired, new filters can be generated for the documenter and run on the modified document collection to add the new links. If other changes are desired - if the documenter wishes to remove links, or change the destination of a set of links - the filter configuration files should simply be updated and the filters regenerated. The modified filters are then run on the original set of documents.

The standard editing environment provides the facility to insert hypertext links and cross references; the author is free to use this for instances where multiple links are not desired. The filters produce document transformations of the form shown in Fig. 2. In the segment that has been modified, underlined terms designate hypertext links with the typeface denoting link type. In instances where color is available, link types are denoted using different colors instead of typeface. The page numbers are cross references inserted to facilitate the use of the documents in hard copy form.

We are currently using the documents describing the MSS software to test our system. These documents are divided

into chapters which discuss major system components, appendices which discuss subtopics, collections of figures and tables, sets of documents describing code libraries and classes, and a glossary. The current implementation provides a set of hypertext link types to indicate links to each of these document types.

In the example shown in Fig. 2, italics represent links to the glossary page. No page numbers are necessary for this type of link. Block letters and the cross-reference format 'page F#' denote links to figures. Helvetica font and the cross-reference format 'page A#' denote links to appendices. The appendices contain supporting information which is important but not integral to understanding the system at large. Finally, boldface type and the cross-reference format 'page #' denote links to other main topic areas.

SLEUTH as Seen by Users

SLEUTH as seen by users (see Fig. 3) consists of the modified set of documents, the directory-structured source code index and the search engine, all integrated using a FrameMaker viewing environment which has been augmented to include an interface to the search engine. Users can navigate the documentation using tables of contents and the typed hypertext links, or they can use the search engine to investigate phrases of interest. Users can also use the source code index to navigate the source code.

The Viewing Interface

The on-line interface presented to users is the standard FrameMaker viewing environment that has been enhanced using the FrameMaker Developer's Toolkit [5, 6]. This version can be navigated using the typed hypertext links in the traditional point and click fashion. An example viewing session is shown in Fig. 4. Multiple windows can be viewed

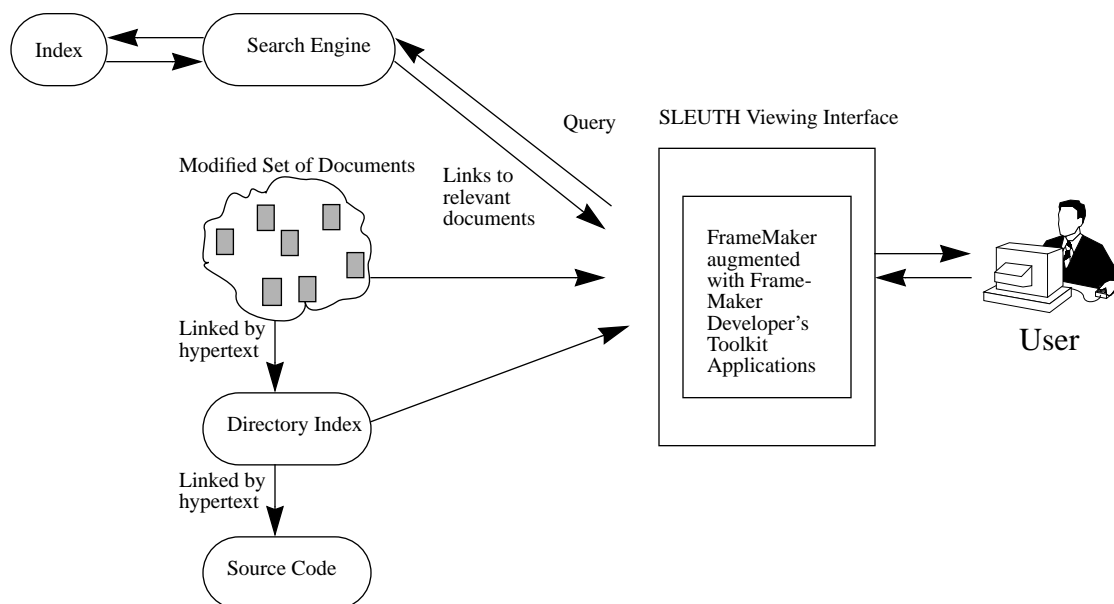


Fig. 3 SLEUTH as seen by users.

In addition to the basis provided by FrameMaker, the major components of the SLEUTH system are:

- a search engine which allows full text searching on the documents in the collection. The search engine allows a user to locate information on a specific topic if it is included in the set of documents.
- the interface to that search engine.
- configurable hypertext and cross-reference filter generators,
- and the facility to produce a directory-structured index for the system and library source code.

The SLEUTH system can be viewed from two angles, from the perspective of the documenter and from that of the documentation user.

SLEUTH as Seen by the Documenter

SLEUTH as seen by the documenter (see Fig. 1) consists of the FrameMaker editing environment, configurable filter generators, a search engine and the facility to produce a directory-structured index for the source code. When the initial set of documents has been completed and the configuration files have been updated, the custom hypertext and cross-reference filters are generated for the user. The initial set of documents is then passed through these filters to produce the modified set of documents. The text contained in the initial set of documents is also indexed for use by the search engine. The indexing process does not modify the documents. Finally, a directory-structured index is produced for the source code. This index provides links into the source code, but the indexing process does not modify the source code. The modified set of documents, directory

index and source code will be visible to the users.

The Authoring Interface

The interface presented to the documenter is the standard FrameMaker WYSIWYG editing environment. The documenter uses FrameMaker to compose the initial set of documents and any associated figures and tables. The SLEUTH system provides a document template which defines paragraph types, the formats to differentiate the typed hypertext links and other document formatting information.

The Search Engine

The SLEUTH system currently utilizes a WAIS (Wide Area Information Server) [10] search engine. WAIS is intended for distributed information retrieval and based on a client-server model of computation.

The indexer is the only portion of the search engine which concerns the author. The indexer creates a table of the document and paragraph locations of terms in the documentation text. Terms such as "a", "an", "the", etc. are not indexed. The documentation should be re-indexed whenever changes are released to the users.

The Filters

While composing these documents, the author uses a text editor to configure the filter generators for the hypertext and cross reference filters. The author provides a list of terms that will become hypertext links and a list that will become cross-references. The author must also record the types and

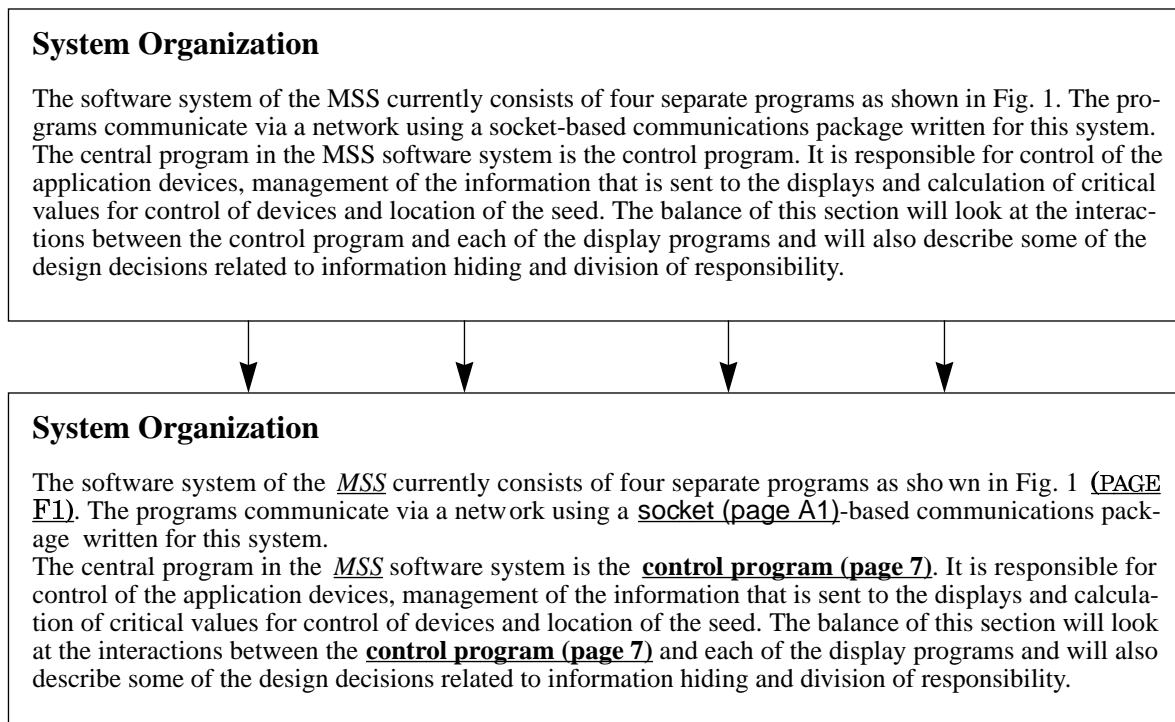


Fig. 2 Document fragment before and after modification by hypertext and cross-reference filters

Basic Features

We propose that an effective documentation system should contain a number of basic features:

1. The system should provide both an effective authoring environment to facilitate the construction of the document collection and an integrated viewing environment. The viewing environment should be integrated so that the author can be assured that structure and format choices will be consistently conveyed to the user.
2. Given that these documents are useless if they cannot be effectively accessed, the system should provide the user with effective tools for extracting information from the document collection.
3. The documents should be structured and organized to enhance the performance of the information retrieval component of the system. Typically information retrieval systems must be prepared to deal with a variety of document types. We have the option to specify document structure and organization to improve retrieval performance. The authoring environment should facilitate document organization.
4. The system should provide typed hypertext links to enhance access to the information contained in the documents. These links will provide visual cues for the type of information provided in addition to being used for navigation. As much as possible, the links should be generated synthetically so that all instances of a link are present, accurate and easily updatable.
5. Documents should be written and organized in a hierarchical fashion. Documents on progressively lower levels should elaborate on information presented in the next highest level. The range of information could extend from general system overview to source code.
6. Different users will likely find different levels of detail

most useful. Therefore, a facility should be provided to navigate this hierarchy in both a vertical and horizontal fashion. Navigating up and down the hierarchy would provide more or less detailed information on a topic as desired by the user. Navigating left to right would provide more information about other topic areas at the current level of detail.

7. Both on-line and hardcopy versions of material should be available. The hardcopy version should be a first-class component of the system and maintain as much of the functionality of the on-line version as possible.

We have endeavored to incorporate these features and to build documents and a document access system that provide truly useful information to the user.

The SLEUTH System

SLEUTH utilizes FrameMaker as the basis for both the authoring and viewing environments. FrameMaker provides a WYSIWYG editing environment and supports both hypertext and cross referencing. It also provides basic navigational features and provides a toolkit for customization. Because it was designed to be a document preparation system, FrameMaker provides an effective starting point for our prototype. Many of the basic features necessary for document creation and editing are provided, allowing effort to be concentrated on more specialized features. FrameMaker can be used to produce effective hardcopy versions of the documents because it is a document preparation system. Using FrameMaker to support authoring, viewing and printing eliminates any potential inconsistencies that might have occurred if we had chosen to utilize different representations for these three purposes.

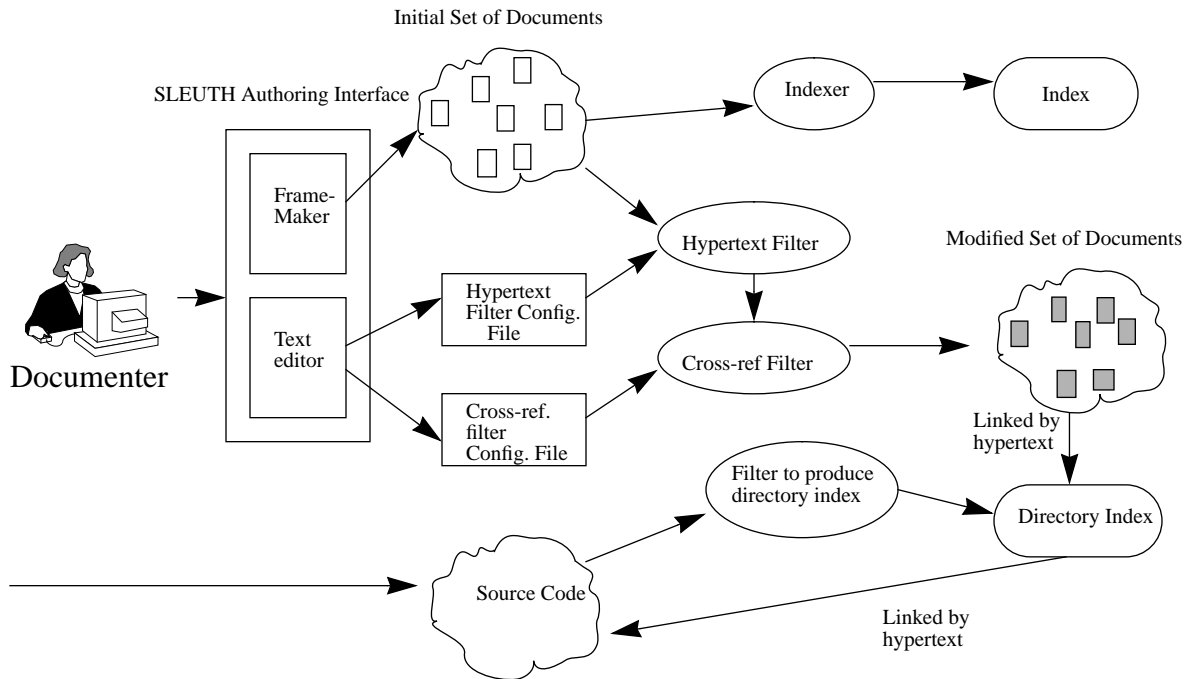


Fig. 1 SLEUTH as seen by the documenter.

tions of the new component with the existing ones.

- A project manager preparing a status report. This user needs general information on the areas of the project covered by the status report.
- A regulator who must determine if a requirement has been implemented. The regulator may need information at varying degrees of detail to complete this task.

An effective documentation system should address the needs of all of these users. Standards and document publishing systems alone are not an adequate solution to the problem. Documents produced using current documentation practices may not contain the information needed by users or the information may exist but be difficult to access. Both an effective supporting environment and guidelines for document construction are needed.

Existing Approaches

Document Publishing Systems

There are a number of commercial document publishing systems on the market which facilitate writing, viewing, and editing large documents and sets of documents. Some examples include FrameMaker, Interleaf, etc. These systems allow documents to be collected into sets and automate the creation of indices and tables of contents. Most provide a WYSIWYG editing environment and tools for creating diagrams and mathematical formulae. They provide useful tools but do not offer guidance on what should be written.

Other approaches to managing software documentation have been considered.

DIF

The Document Integration Facility is an environment to develop, maintain and browse the documentation associated with a large-scale software system. It was introduced by Garg and Scacchi in [7]. The documents produced in DIF are the documents associated with each phase in the software life cycle. Each segment of a document is viewed as an object and hypertext links between documents are relationships between objects. These relationships are stored in a relational database. Searching is allowed only on predefined keywords.

SODOS

SODOS is the Software Documentation Support Environment proposed by Horowitz and Williamson [8,9]. This environment integrates the ideas of an object-based modeling of the software life cycle with a database management system. The information gathered at each stage of the software life cycle is put into structured documents. The database consists of all documents associated with a project. SODOS provides a document interface which allows users to modify and query documents. Possible query terms are defined by the author.

HyperCASE

HyperCASE was proposed by Cybulski and Reed in [2]. It

integrates the two concepts of hypertext and CASE tools. HyperCASE uses hypertext to link related information in documents associated with the software life cycle. HyperCASE provides an application for creating, editing and presenting documents, a repository and a data dictionary. HyperCASE provides a number of browsing capabilities. Both DIF and SODOS integrate database management systems with the software documentation associated with the software life cycle. Queries are allowed on predefined keywords. This can be moderately helpful to the user; however, it places an additional burden on the documenter. When predefining keywords, the documenter is forced to anticipate the needs of the end user. If the documentation is used in a way that the documenter did not anticipate, desired terms might not be available for searching. This is similar to the problem often encountered with book indices. An individual may be certain that a key piece of information is included in the book, having read it once before, but be unable to locate the topic again because it was not included in the index.

Approach

Our main goal has been to apply information retrieval techniques to software documentation in order to provide a mechanism that allows a variety of users to find the answers to questions about software documentation quickly and consistently. Specifically, given a collection of software documents, a user should be able to find the answer to a specific question or a broad array of information on a general topic with equal ease.

In our treatment of this problem, we have considered three fundamental points:

1. The organization of on-line software documents.
2. The presentation of these documents when browsing.
3. Searching the document collection.

We have chosen to attack this problem by focusing on authoring, document structure and retrieval. We want to author and organize the documentation with retrieval in mind, understanding that documentation should be written so that it will be most useful for users. At the same time, documents should be kept current and the cost to the documenter of maintaining the documents should be as low as possible.

Different User Needs

We realize that different users of this system will have different needs. While a newcomer to the project would benefit from a general system overview, this is unnecessary for a longtime team member interested in a specific implementation detail. We hope to be able to effectively serve the needs of both of these users as well as those of managers, inspectors and other users whose needs cannot be anticipated. An effective system will provide both overviews of topic areas and answers to specific questions for these users.

We first studied current methods and the state of the art, listing features and shortcomings of existing systems. We then developed several prototype systems to determine the usefulness and feasibility of a subset of the features being considered.

A Systematic Approach to Creating and Maintaining Software Documentation

Allison L. Powell, James C. French, John C. Knight
(alp4g | french | knight)@virginia.edu

Department of Computer Science, Thornton Hall, University of Virginia, Charlottesville, VA 22903

Keywords. information retrieval, software engineering, documentation.

Abstract Current software documentation is difficult to write and seldom meets the varying needs of its users. We propose that by considering different users and applying information retrieval techniques to the information included in software documentation, we can provide effective access to that information. We submit a set of features for inclusion in documentation database systems and describe a prototype designed to determine the usefulness of these features.

Introduction

As software engineering projects grow in size, software documentation becomes increasingly important. It is essential that knowledge about the project be recorded frequently and consistently. However, while software engineers spend great time and effort preparing software documents, they often work without concrete guidelines. They have little assurance that the usefulness of these documents to themselves, other software engineers, or other readers is commensurate with the time spent. The result is inadequate or inappropriate documentation, frustrated software engineers, and complaints from users.

It is not enough to simply write software documentation. It must be maintained regularly and be accessible to users. Current documentation is often hard to use and internal documentation can be inconsistent between project revisions. This limits the usefulness of software systems in general. System documentation is often seen as such a daunting task that it is not done at all. There are many software systems for which the only available documentation is the source code.

We propose that software documentation is a source of information and should be treated as such. Our key area of concentration has been the application of information retrieval techniques to provide effective access to the information included in software documents. Another key con-

sideration has been meeting the different needs of different documentation users. Different users will naturally find different subsets of the information most useful, depending on the purpose for which they use the documentation.

We have considered both the documentation authors and the end users in producing a prototype system to facilitate the construction of more effective software documentation. Although our application area is safety-critical systems, these techniques are more widely applicable.

We believe that these techniques can improve the usefulness of software documentation by providing users with a number of ways to uncover the answers to their questions. We have produced a system which consists of both an authoring and a viewing environment. In the SLEUTH (Software Literacy Enhancing Usefulness To Humans) system, we provide a mechanism to link a collection of documents using typed hypertext links. We also provide the facility to perform keyword searches on the text of the documentation. We have endeavored to build documents and a document access system that provide relevant information. In our experiments, we are focusing our attention on two large safety-critical systems: the Magnetic Stereotaxis System and the University of Virginia Nuclear Reactor.

Problem

In general, software documentation is not designed to provide high-quality responses to questions from a variety of areas. While there are numerous standards which prescribe documents to be produced and their formats, they often provide no justification for the inclusion of particular documents. Such standards include DOD-STD-2167 [15], DOI78A [13], OO-55 [1] and others. There are also systems which automatically generate documents given a collection of source code. While there are some useful features in both of these approaches, the documents produced often contain little useful information.

Typical documentation is static and therefore does not address the different needs of various documentation users. Typical users might include:

- A maintenance programmer charged with installing a patch. This user needs detailed information on a specific system area, along with interactions with other components.
- A software engineer who must design a new system component. The software engineer will need detailed information on all components to determine the interac-

"Permission to make digital/hard copy of all or part of this material without fee is granted provided that copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery, Inc. (ACM). To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee." Copyright 1996 ACM.