Software maintenance: redocumentation of existing Cobol systems using hypertext technology

Freeman, Robert Marriott

Software Maintenance: Redocumentation of Existing Cobol Systems using Hypertext Technology

Robert Marriott Freeman

Master of Science Thesis

University of Durham

England

April 1992

Abstract

One of the major problems associated with the maintenance of existing software systems is their lack of documentation. This can make very large, poorly structured programs very difficult to maintain. Nearly all traditional documentation tools are either designed for use in the development stage of the software lifecycle or are report generators such as cross reference generators. The problems of lack of documentation are compounded when applied to third party software maintenance as the staff are often initially unfamiliar with the code they are maintaining. This thesis describes these problems in detail and evaluates the feasibility of a tool to help with redocumentation based on current hypertext technology.

Acknowledgements

The author would like to thank Mr J.Kane, Mr M.Walters and Mrs J.Hawkins of AGS Information Services Ltd. for their help in supplying data and financial support which has enabled this thesis to be written. Thanks are also due to Mr S.Connolly and Mr B.Eliasson of RANK XEROX UK Ltd. for their help and the donation of a 6085 documenter system on which this thesis has been written.

I also wish to thank my friends especially Malcolm Munro, Simon Cooper, Barry Cornelius, Rick Morgan and Richard Turver who have contributed to the research and given up many hours of their time.

Finally I would like to thank my parents for their support during my continued education.

This thesis is dedicated to my wife Judith, who has provided a great deal of support and motivation during the past year.

Software Maintenance:

Why does it happen?

Because it happens.

Roll the bones.

Neil Peart 1991

Contents

Chapter 1

Introduction

Since the initial survey by Lientz and Swanson[41] showing the vast amount of money being spent in software maintenance activities, a considerable amount of research has been done to try to reduce this expenditure. The topic of documenting software has also received a great deal of attention. However little research work has been undertaken combining the topics of maintenance and documentation.

In recent years some companies have been subcontracting the maintenance of their systems to specialist companies. For the subcontracted company, special problems exist, especially with the initial analysis of the systems.

The research described in this thesis has been concerned with these three aspects. There are three main aims of this research :

- Investigate the use of documentation in a software maintenance environment, with special reference to systems written in Cobol.
- Investigate the appropriateness of hypertext for software redocumentation.
- Build a prototype tool to assist the maintenance programmer with the analysis of code and the production of documentation.

Chapter 2 of this thesis gives the reader an overview of the software maintenance process, and then describes the idea of third party software maintenance. The last part of this chapter gives an overview of the four main reverse engineering disciplines with special emphasis on preventative maintenance and research done in this field. Software documentation and the need for redocumentation is described in Chapter 3, while Chapter 4 gives the requirements for a complete redocumentation tool. An evaluation of Hypertext for use in software redocumentation is presented in Chapter 5. The next chapter describes a documentation model for use in third party software maintenance, and Chapter 7 discusses the implementation of a Redocumentation Aid for the Maintenance of Software developed from the documentation model. Finally, Chapter 8 gives the conclusions of this research, and describes areas for future research.

Chapter 2

Software Maintenance

2.1 Introduction

The first part of this chapter gives an overview of software maintenance from a management perspective showing the economic importance of research in this area. A description of third party software maintenance is then given, highlighting AGS Information Services' operation. Finally the chapter describes the activities that are currently used in preventative maintenance, and the research being carried out in this area.

2.2 The Maintenance Problem

"Software Maintenance" is a widely used term within the IT industry, and it is now a well accepted fact, both within academia and industry, that software maintenance does constitute between 50 - 80% of the IT budget[40,46]. The cost of maintenance therefore can pose a serious problem to IT management, and can cause the development of new systems to be dramatically curtailed. At the same time, reducing these costs presents them with an opportunity for reducing their budget or increasing output substantially.

Different organisations use the term "software maintenance" to refer to a whole series of tasks, for example bug fixing, enhancements etc, and thus a clear definition is required. In this paper we shall assume the definition given by the IEEE [35]

> Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a changed environment.

In keeping with the above definition, Swanson[59] divided the maintenance activities up into four distinct areas:

- **Perfective Maintenance**
  The addition of extra functionality to a software product.

- **Adaptive Maintenance**
  A change in the environment in which the software product is used.

- **Corrective Maintenance**
  The removal of errors from a software product.

- **Preventative Maintenance**
  The change of a software product to improve its maintainability.

Turner[62] gives figures of 60%, 18%, 17% and 5% respectively for the relative maintenance effort between the various activities. Swanson, in later work with Lientz[41], expanded the definitions of the maintenance activities, with perfective maintenance changing the requirements, design and source code, adaptive maintenance changing the design and source code, corrective maintenance changing the source code only, while preventative maintenance changes the source code but can also change the design. From the above we can conclude that the most common maintenance activity, perfective maintenance, affects the greatest number of stages of the software life cycle.

Lientz and Swanson's results on the effects of the maintenance activities also have implications as far as documentation is concerned, and the extent of documentation that has to be changed with the different maintenance activities. This will be dealt with further in Chapter 3.

Software maintenance is still often approached in a fairly ad-hoc manner with no formal procedures in place to deal with enhancements or bug fixing. This position is the responsibility of management to correct, but as Cooper [20] points out, there is a lack of effective communication between the managers of the maintenance teams and senior managers running the business. If more information were available for senior managers, then perhaps a more structured and efficient use of maintenance staff would take place. Here is an opportunity for documentation to help with the feedback of information to managers showing the magnitude of the maintenance problem.

The cost of software is becoming more expensive with respect to the cost of hardware, a phenomena first predicted by Boehm[6]. The cost of Software maintenance is reaching serious proportions and the greater the maintenance problem the greater the problems throughout the IT industry.

Recently it has been reported that staff turn-over is getting so high that the average employment time is 6 months[33]. This is partly due to the unreasonable maintenance loads put upon some new staff. Inexperienced staff are not always the best people to do the maintenance. They do not have a deep understanding or familiarity with the system unlike some of the more experienced programmers. It is also often discouraging for a young programmer to be given some other person's code and given instructions to maintain it.

Often maintenance staff have little or no appropriate tools, unlike the design staff who find it easier to acquire new equipment and tools. It is important to design good and useful tools to make the job of maintenance more attractive, and decrease the staff turn-over.

2.3 Third Party Software Maintenance

Third party software maintenance is at present prevalent in two distinct guises. The first, of greater interest to this research, is where a company with an internal data processing section contracts out to a third party the maintenance of software, which has often been specifically written for the company to control jobs such as their payroll system[49]. The second type of third party software maintenance is described by Tang[60]. This is where a third party takes over the maintenance of a commercially available product. This takes place because, within the example company, Hewlett Packard, new products are released two and a half times faster than old software becomes obsolete leading to an accumulation of software which needs support. Therefore, after a product has become stable, the maintenance of the product is often given to a subcontractor to reduce the number of products which have to be maintained in house, thus freeing staff for future software development.

This thesis concentrates on the third party software maintenance carried out by AGS Information Services. An example of their operation will now be given.

When a proposal for maintenance has been agreed between a client and AGS, a team leader will produce a document describing the site of the client. This initial proposal is paid for by AGS, but all work after this is budgeted and paid for by the client. This document is then made available to the members of the team who will move to the site of the client.

At present this document is paper based. Because this document has to be maintained for the lifetime of the client, an on-line form of documentation would be more suitable. This document, as well as being read by the staff at the initialisation of the project, will also be read by new staff members recruited to work on the site of the client during later years. It also forms a basis for the contract between AGS and the client, and so its correctness, and continued correctness, is imperative.

The third party software maintenance approach has been shown to reduce the staff turnover in a company[1] as well as reduce the capital expenditure on maintenance. This is partly done by motivating staff using a software house environment with a full career path and fringe benefits. This is at least due to the exclusive use of experienced maintenance staff by the subcontractors, with trainees never being sent out to a client site. The system of using a separate company to carry out the maintenance overcomes the problem of junior programmers having maintenance imposed upon them.

There are several serious problems associated with third party software maintenance. Firstly, the maintenance staff entering the site of a client follow the client's guidelines for employees, and thus considerable readjustment has to take place. The site profile document can be very useful at this point. Secondly, and more significantly, because the chances of the maintenance staff ever having worked on any of the client's systems before are remote, a familiarisation period is required. Again the site profile document can be of significant use. This underlies the necessity for the maintenance of this document.

Technical documents giving low level details of the systems are also found invaluable. The use of the other forms of documentation required by maintenance staff and especially third party maintenance staff are found in Chapter 3. The problem of taking over the maintenance of the client's software systems is partly handled by taking over a client site in stages. This stops the maintenance productivity level falling too low, though there are invariably teething problems when the substitution of staff takes place.

2.4 Preventative Maintenance

Preventative maintenance can be categorized into four different types :

- **Inverse Engineering**

  The process of discovering the requirements specification for a software system or component, and then re-implementing using modern software engineering methods.

- **Design Extraction**

  The process of redesigning parts or all of a software system, to improve its quality, and then re-implementing the design.

- **Code Renovation**

  The modification of source code items based on modern software engineering principles to improve the maintainability of the software.

- **Redocumentation**

  The process of extracting information from the source code or associated materials to assist in the analysis of a system.

The term Preventative Maintenance is being used in this thesis to describe activities which are sometimes referred to as reverse engineering because the definitions of reverse engineering are varied and often confusing. One definition by Bennett et al[5] describes reverse engineering as:

> The process of redesigning parts or all of a software system, to improve quality, and then re-implementing the design.

This definition of reverse engineering implies a forward engineering process which is also implied by Bush's[11] definition of re-engineering . In contrast to this Chikofsky and Cross[18] state that:

> Reverse engineering is the process of analysing a subject system to identify the system's components and their inter-relationships, and to create representations of the system in another form or at higher levels of abstraction.

This definition is similar to Bush's[11] definition of reverse engineering except that it does not contain any forward engineering element but is just concerned with obtaining greater information of the system by recreating the specification, design or documentation. This definition of reverse engineering has elements which also appear in the Bennett et al[5] definition of preventative maintenance (i.e. inverse engineering, reverse engineering and re-engineering) but without the forward engineering content.

In short the terms are confused, and used interchangeably by different authors. Thus in this thesis the term preventative maintenance is being used to describe processes which are often grouped under the reverse engineering banner. The term reverse engineering used by Bennett et al in the definition of preventative maintenance can be replaced with the more descriptive term design extraction, while code renovation can replace re-engineering. This definition leaves one major omission - Redocumentation. Redocumentation should certainly appear in this list as the definition of preventative maintenance, described earlier in this thesis, is to increase the maintainability of a system. By recreating documentation or by verifying the correctness of available documentation, the maintainability of a system will be improved, and therefore must be included in a definition of preventative maintenance.

The four preventative maintenance processes are in order of automation, and also in order of the changes to the source code. For instance, a complete inverse engineering process, having regenerated the specification from the code, will then reconstruct the code using modern software engineering techniques. This, however, is a very expensive process. At the other end of the scale, redocumentation has no effect on the source code, but also can prove to be far more costly than code renovation as often documentation has to be updated from the specification stage through to the test and maintenance manuals. It is this process of redocumentation that is detailed in Chapter 3, while a brief introduction to the other three research techniques appears below.

2.4.1 Code Renovation

This is the simplest of the three techniques described here, and involves only making simple changes to the source code while leaving in place any errors that may be present from the design stage or any previous stage of the software lifecycle[48]. This technique is therefore simply present to make the code more readable. Code analysis is often suggested to be the most expensive phase of the software lifecycle[56] and as much as 40% of the software maintenance effort is spent trying to comprehend the code.

As Calliss[13] points out, the use of code renovation tools can also make the documentation within the code in the form of comments redundant or even misleading. This is a serious problem and can lead to the code analysis being far more difficult than with the original code. Therefore great care has to be exercised when code renovation is carried out. In particular strict guidelines have to be enforced to ensure that any code added is written to the original standards, and that any documentation accompanying the system is updated. In general this takes considerable manual supervision as opposed to the desirable totally automated process. However the REDO[38] project hopes to achieve this by the production of a tool kit to help with code renovation. These tools will also help form the basis of a system to help with design extraction and inverse engineering.

2.4.2 Design Extraction

Design extraction is the first of the techniques to refer to a higher level of abstraction of the lifecycle, as opposed to the implementation section. The technique is similar to code renovation except that instead of simply unravelling what could be regarded as confusing code, the system is often totally restructured using modern design and modularization techniques. An approach for design extraction and then re-implementation has been developed by Sneed[57]. With the tests Sneed has carried out the design extraction and re-implementation in a language not necessarily the same as the first (this is sometimes referred to as reverse engineering or recycling) has lead to an average of more than 33% saving of time and effort due to the drop in complexity and increase in maintainability. These results are also supported by a study conducted by Gibson and Senn[28] showing reduced error rates and maintenance efforts due to re-engineering.

Sneed's approach to carrying out the design extraction and re-implementation were detailed by him and Jandrasics[55], and briefly consists of the following:

- Static analysis of the existing program and creation of the necessary data and data command tables.

- Modularization of the program using the data and command tables together with some modularization criteria.

- Internal restructuring of the new modules in the design language.

- Manual optimization and adjustment of the new modules in the design language.

- Generation of structured target language modules from the design language modules.

As Sneed points all but one of the steps are now fully automated.

There are three main problems with this technique. One of these also occurs with code renovation, the loss of documentation within the code when the process is carried out. In addition, when the design of the system is being altered, the design documentation can be rendered useless. Procedures must be in place to make sure that the documentation is updated with this process - in short the documentation has to have design recovery implemented on it as well as the code. One final problem is proving that the program produced is equivalent to the original system. Design extraction methods to date do not show whether the code produced is logically the same as the previous system, and thus it is conceivable that any system produced from these tools could be different from the original.

2.4.3 Inverse Engineering

A great deal of software, which is still in regular use today, was originally designed before the development of modern structured design techniques such as SSADM[21]. The source code of a lot of this software therefore has little or no structure. As the original specification is probably not present, and if it is present the credibility of it is likely to be low, it is sometimes desirable to recreate the specifications from the code. When the specification has been fully obtained, then a forward engineering process can take place to bring the product into line with modern design standards. As with code renovation and design extraction, care has to be taken that no internal documentation is deleted or invalidated through the change.

As stated earlier in this thesis the major activity of software maintenance is perfective maintenance: the enhancing of a system by adding additional functionality. When an addition has been made, inevitably the specification of the system will have changed. Again obtaining a specification of this additional code is desirable so that the system is fully documented.

Yang[70] proposes the following system to implement the recovery of the specification:

- Understand the software.
- Identify the modification objective and the modification approach.
- Implement the modification.
- Verify the modification.
- Abstract the specification.

To perform the abstraction to a specification, a formal method has been developed using the transformations of Ward[66]. A major advantage of using formal transformations is that it overcomes any problems with program equivalence, as the transformations have been mathematically proven. This technique therefore provides an immediate advantage over the technique of code renovation described earlier.

The transformations are now being automated to be used in a tool known as the Maintainers Assistant[67]. Problems still exist with full specification recovery, the major one being the difficulty in regaining information lost in the forward engineering process which occurred at the initialisation of the project. To recreate the intended specification which may have been lost or corrupted during the design and implementation processes is near impossible. This project however brings us closer to retrieving the actual specification. At present, a considerable amount of manual input is still required, though work is in progress to increase the automation of the tool.

2.5 Summary

Software maintenance is partly responsible for the current high cost of software. The use of specialist teams to maintain the software can help reduce costs, but still there is a large expenditure involved in keeping the software operational. There have been claims in advertisements that modern fourth generation languages will not need maintaining. To date this has not been the case. Even if a language that did not need maintaining was developed, there would still be a vast number of systems written in other languages which would continue to need maintaining for many decades to come. Therefore the need for continued research into preventative maintenance that helps to make the present code more reliable and maintainable is essential. Although research into inverse engineering which can help greatly with systems presenting very serious maintainability problems, there is still comparatively little work being done in the area of redocumentation. This subject is discussed in detail in the next chapter.

Chapter 3

Software Documentation

3.1 Introduction

Documentation is essential for the communication of information which is not contained in the code alone. The area documentation can cover is great and in the first part of this chapter the various types of documentation that are typically produced during a system's lifecycle are detailed. The particular uses of documentation with respect to maintenance are then described. Finally techniques for measuring the quality of the documentation are discussed along with an overview of tools available for documentation.

3.2 Categories of Documentation

In an ideal situation, documentation should be produced throughout a system's life-cycle, though unfortunately this is rarely the case. Even though documents may well be produced in all the required phases, the completeness and usefulness of these documents is often called into question. The reason for the lack of good documentation is mainly due to two factors. Firstly, most programmers regard documentation as mundane, and as they can understand their code, so should any other programmer. The second reason, as stated by Boehm[7], is the huge financial cost of preparing documentation compared with the costs of the other activities of the software lifecycle. This is illustrated in the Figure 1 .

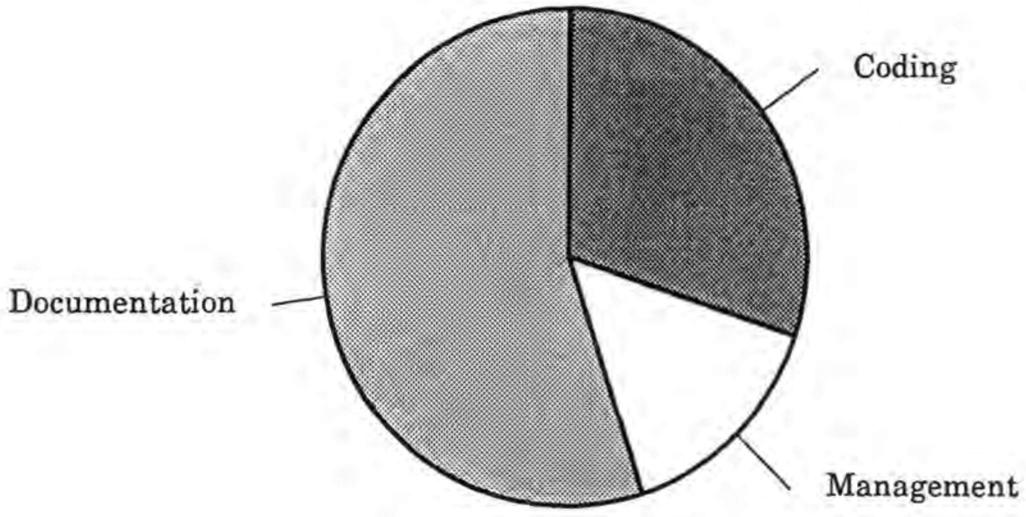


Figure 1
Percentage Costs During the Software Lifecycle

Because of this enormous proportion of the total expenditure attributed to documentation, and the fact that documentation is often perceived not to add to the functionality of a product, it is often one of the first areas of software production to be cut.

Nevertheless some good and relatively complete documentation guidelines exist. One example is the U.S. Department of Commerce's standards for documentation[63]. These standards require the production of the following documents:

- **Functional Requirements Document**

  To provide a basis for the mutual understanding between users and designers of the initial definition of the software, including the requirements, operating environment, and development plan.

- **Data Requirements Document**
  To provide, during the definition stage of software development, a data description and technical information about data collection requirements.

- **System/Subsystem Specification**
  To specify for analysts and programmers the requirements, operating environment, design characteristics, and program specification (if desired) for a system or subsystem.

- **Program Specification**
  To specify, for programmers, the requirements, operating environment, and design characteristics of a computer program.

- **Data Base Specification**
  To specify the identification, logical characteristics, and physical characteristics of a particular data base.

- **Users Manual**
  To describe the functions performed by the software in non-ADP terminology, so that the user organization can determine its applicability and when and how to use it. The users manual should serve as a reference document for preparation of input data and parameters, and for interpretation of results.

- **Operations Manual**

  To provide computer operation personnel with a description of the software and of the operational environment so that the software can be run.

- **Program Maintenance Manual**

  To provide the maintenance programmer with the information necessary to understand the programs, their operating environment, and their maintenance procedures.

- **Test Plan**

  To provide a plan for the testing of software; detailed specifications, descriptions, and procedures for all tests; and test data reduction and evaluation criteria.

- **Test Analysis Report**

  To document the test analysis results and findings, present the demonstrated capabilities and deficiencies for review, and provide a basis for preparing a statement of the software's readiness for implementation.

The U.S. Department of Commerce's standards also relate the preparation of the documents to various points of a system's lifecycle, as shown in Table 1.

Each section of the Department of Commerce's standards is complemented by other publications, for instance the IEEE 830 Software Requirements Document[34] which is a good example for a Functional and Data Requirements document and could be used to replace the Functional Requirements and Data Requirements documents in the Department of Commerce's guide.

Initiation Phase	Development Phase				Operation Phase
	Definition Phase	Design Stage	Programming Stage	Test Stage	
	Functional Requirements Documents	System/Sub system Specification	Users Manual		
		Program Specification	Operations Manual		
	Data Requirements Documents	Data Base Specification	Program Maintenance Manual		
		Test Plan	Test Plan	Test Analysis Report	

Table 1

Though the Department of Commerce lists maintenance documentation, it only appears in the programming stage. Within their structure there is no facility for any documents to be produced detailing updates or changes to the system, and no facility for updating any of the previously produced documents to make sure they are still valid after any changes have been made during the operational phase. This is a serious omission in an otherwise well-structured approach to documentation. To partly overcome this omission, a System Maintenance Journal as described by Martin and McClure[43] could be used in the operation phase. The contents are described below:

System Maintenance Journal:

- Change philosophy

- Quality preservation / improvement strategies

- Problems

- Trouble spots

- Change / Error history

Using a journal system still does not address the problem of updating or maintaining the documentation.

As has been stated earlier, the costs of producing the documentation to these standards is immense. This is not the only problem. As the size of programs has increased exponentially with the plummeting hardware costs, so has the size of the documents produced. As stated by Singleton[53], NASA scientists measure the test output of one run on the Space Shuttles software by the ton. Not only does this give enormous problems trying to store the documentation, but it also produces major access problems. The documentation becomes so large that finding the relevant piece of information within it without an exceptional automated index is very time consuming. The problem is compounded by trying to maintain consistency between all sections of the documentation.

3.3 Documentation and Maintenance

A survey has shown that the two major problems in understanding a system[16] are the lack of documentation and incorrect documentation. This is disturbing as one of the main activities of maintenance staff is that of understanding a system and its associated code. Documentation has always been given a low priority status compared to other activities in software development. Documentation is nearly always left to the end of a project, and as projects invariably run late, the documentation produced is often reduced from the amount that was originally intended, or even completely cancelled. It is not unusual for the only documentation concerning the design of the system to be contained in comments within the code itself. If restructuring takes place, this information can be lost or can become inappropriate.

Some information is useful as comments within the code itself, especially when it describes a complex piece of coding, however, design information should not appear within the code itself, but should be within attached documents. It is the lack of this design information that can make the maintainer's job of understanding the system more difficult. This information can be even more vital when third party maintenance is being carried out on a system, where maintenance staff will not have been previously exposed to the code.

A possibly greater problem in maintenance than the simple lack of documentation is the presence of incorrect or unreliable documentation which is often found to be the case in older systems[39]. Finding a small section of the documentation to be misleading can result in the whole set of documentation being discarded as useless. One of the more recent ideas is that of self documenting languages, as claimed by some 4GLs. Though this may help the maintainer understand the code itself, it will not give them an insight into why a certain procedure was written in a particular way, or why a design decision was made. It is often the information contained within these last two classes which is most valuable to the maintenance staff.

Even if a truly self documenting programming language were developed, then it would still be of no use for the maintenance of old code, the majority of which is written in COBOL or Fortran. Neither of these languages can be described as self-documenting . Unfortunately the momentum of these languages will ensure their continued use for many years to come[71]. The language C, along with the UNIX operating system, is now becoming more popular especially within financial institutions. It is envisaged that the maintenance of this software will cause great problems within the next decade due to the features of the language which can make the code very difficult to understand if the programmer so desires.

Even within maintenance, documentation is often not produced despite the maintainer's view that it is useful. Thus the documentation produced after a maintenance task has been completed is often inadequate in relation to the magnitude of the system change[4]. A reason that the documentation is incomplete is that it is often regarded as laborious, and there are very few good tools to assist with it. A documentation tool allowing the user to record any ideas about the system, and for someone else to be able to retrieve this knowledge would certainly be useful.

It is not always possible or cost-effective to completely reproduce the design documentation or specification from the source code, and it has not yet been established whether this is the most suitable form of documentation for software maintenance. To attempt to reproduce complete documentation is expensive. It is therefore important to try and identify a method that will allow redocumentation to take place gradually during the maintenance activity while the maintenance programmer is extracting information. A system for incrementally redocumenting old systems in this way has already been described by Foster and Munro[25] and by Fletton and Munro[24]. Adopting this approach should also allow for the maintenance staff to enter what details they feel are important for the maintenance of a particular system.

3.4 Measuring the Quality of Documentation

As described earlier in this thesis, the quality of documentation can be crucial to its use by maintenance staff. The problem comes when trying to define what is meant by the "quality of documentation". It is often more suitable to try and define what is meant by "adequate documentation", i.e. documentation that is satisfactory in quality.

Arther and Todd[4] have formulated a general taxonomy for the evaluation of computer documentation represented in a tree model, see Figure 2. The tree's leaf

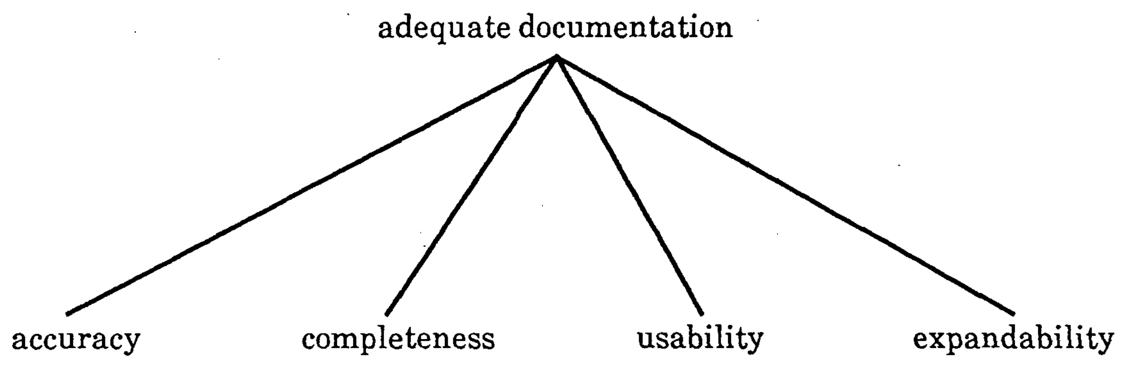


Figure 2. Decomposition into Qualities

nodes display the various *qualities* of the documents. This tree representation of document adequacy therefore fits in with our definition of adequacy - the satisfaction of quality. Below, a description of each of the *quality* indicators is given.

3.4.1 Accuracy

Accuracy in a document is probably one of the most important attributes. As stated earlier in this thesis, if one part of the documentation is found to be in error, then this calls into question the rest of the documentation associated with the system. As well as not containing straight factual errors, the documentation, to be accurate, must reflect the current state of the system, and not that of a previous release. In this case the documentation is strictly correct, but inconsistent with the code. Other inconsistencies, such as differences between requirements and design documents are also frequently present.

Presence of the latter form of inconsistencies can often explain why a system has not met user expectations and also has serious implications for the system in the form of the ripple effect. An inconsistency at the junction of the specification and the design can lead to non-intended features in the code, test manuals, and maintenance manuals, in short throughout the system's life. An inconsistency between the implementation manual and test manual is therefore less serious, but still a problem.

Along with inconsistency, traceability is often a problem, with the route between the requirements and the implementation either non existent or obscured. The lack of traceability within the system can make the analysis of code with respect to other areas of the system lifecycle difficult. This lack of traceability implies either inaccuracy in the design of the system, or at least in the design of associated documentation.

3.4.2 Completeness

Completeness is the percentage coverage of the documentation with respect to the whole system.

Maintenance work is normally exhausted on a small area of the code. As Fletton[23] states, 80% of the maintenance effort is expended on 20% of the code. Therefore do you need 100% completeness of documentation? However it is difficult to establish beforehand which sections of a system will require maintenance. If the implementers knew this then these areas would probably be corrected before release.

We must conclude, not knowing which areas will produce trouble in the future, that 100% completeness is ideal at release. If the system documentation, when reaching the maintenance phase, does not have 100% completeness, and problem areas are beginning to be identified, then the documentation can be updated incrementally. This is discussed in Chapter 4.

3.4.3 Usability

Usability, in documentation terms, is the ease with which the reader can obtain the necessary information from the documents. This therefore covers several areas: readability, suitability and accessibility.

The areas of readability and suitability are covered in great depth by Guillemette[29], and these attributes have a great effect on the reader. If the reader is not appropriately addressed by the author, then the documentation can be rendered useless. For instance it is imperative that user documentation is not too technical, and technical information should not describe features of the language the system is written in but the system itself. These may sound obvious, but such errors are present in a large amount of documentation.

Within usability comes accessibility. This term covers two areas. Firstly documentation must be obtainable by the user and must be in a form that they can readily access. If the user cannot get the documentation when it is needed because the form of storage was not suitable, then the vast cost in producing the documentation has been wasted. Accessibility also identifies the problem of accessing information within a document itself. As James states[36], a vital article buried in a stack of irrelevant papers is almost as unavailable as if it had never been written. As discussed earlier, the size of documentation can be enormous, and therefore it is essential that an adequate indexing and retrieval system is available to the reader.

3.4.4 Expandability

Expandability is defined as increasing either the volume, extent or scope of a document. This attribute is vital for the maintenance of documentation. If facilities for updating the documentation along with the code are not present, then the documentation can be rendered useless by a new second release. Great care has to be exercised when updating documentation so that useful information is not discarded or deleted. Clear procedures for making these updates have to be in place to stop useful documents being ruined by incorrect or inconsistent information.

3.5 Existing Documentation Tools

Documentation tools are available, but are less common than many other tools to help with software production.

There are at present three main types of documentation tool, automatic documentation tools, documentation environments, and specialist documentation tools. The tools listed here are mainly those for use with COBOL, but others with particular merit are also listed.

3.5.1 Automatic Documentation Tools

Automatic documentation tools are mainly available for use in COBOL environments. Most take the form of static analysis tools producing a series of reports.

Cross referencers

Nearly all standard COBOL compilers can produce a cross reference listing. Such listings give information on the declaration and usage of working storage and functions. More complex cross referencers can be purchased, some of which give additional information on the type of statement involved at each reference. Typical examples of these cross reference generators are Autoref, CICS-OLFU and SELLA, though the latter comes as part of a reformatter and complete documentation package. The use of these tools can help in navigating around unfamiliar code, though the form in which the output is produced is invariably a report. This information makes more sense when viewed in parallel with the source code. The online viewing of source code on a windowed display would certainly be an improvement, as it would be the automation of a process typically done by maintenance staff.

SOFTDOC

SOFTDOC[37] is a typical documentation tool based on the static analysis of COBOL or PL/1 code. It produces module tree graphs, HIPO diagrams, internal and external interfaces, control flow graphs, data reference tables, test paths, and symbolic constants to help the maintainer with the analysis of a system. The information produced is usually in the form of paper based listings, possibly not the ideal form for code analysis.

C Information Extractor

As with SOFTDOC, the C Information Extractor[17] obtains information on a system, written in C, but provides the information within a relational database as opposed to a listing. This allows the maintainer to ask questions as to the calling and use of variables within the system. This is a major step forward, as it implements information hiding, allowing the user to see only the data which is of immediate concern and not allowing irrelevant data to confuse their train of thought.

SIRE

SIRE is a full text retrieval system to help locate and manage a huge amount of text. As source code and associated documentation are all in the form of text, it seems natural to consider such a system for use in software maintenance[15]. Full-text retrieval works by taking every word in a file and putting them into an inverted table to enable the quick retrieval of the file's contents. Often within software maintenance it is necessary to find the definition, setting and use of variables, and here the use of an inverted table is extremely effective. SIRE works only on a single file present on an IBM PC or DEC VAX using a negative dictionary, either a default system version, or one provided by the user.

ZyINDEX

Like SIRE, ZyINDEX is a full-text retrieval system, although it differs in the fact that it can use multiple files present on an IBM PC and can use phrase query keywords based on Boolean and adjacency operators.

3.5.2 Documentation Environments

Over the past few years a number of documentation environments have been written enabling the user to produce and manage textual and graphical documents for all phases of the software lifecycle. Most of these environments support traceability, central storage of all documentation relating to a project, and the enforcement of a standard layout to the documents produced. A selection of these tools is discussed in this section.

Modular documentation

Though not strictly an environment, modular documentation proposes a method of having a family of documents based on the decomposition of any system into specific levels of abstraction[2], namely:

- **System**

  One or more major components - subsystems that fulfill the user requirements.

- **Subsystem**

  One or more programs that represent a major system component.

- **Program**

  A collection of modules that fulfill a logical function.

- **Module**

  One or more logically related procedures that may be compiled and controlled separately as a unit.

- **Procedure**

  An identified portion of a module consisting of language statements that can be activated on demand.

- **Statement**

  A collection of data elements declarations and language statements.

This top down approach is then split into documents to be produced during the design stage (system, subsystem, program and module level) and then the implementation stage (module, procedure and statement level). This approach presents a method for completely documenting a system during the development stage, though unfortunately lacks any information with respect to the maintenance phase, though it could easily be extended. This could be done by splitting the documents to be produced into three, rather than two, sections: design, implementation and maintenance. The amount of maintenance carried out on the source code would determine which sections of the documents would have to be changed.

FORTUNE

The FORTUNE[44] documentation system helps to produce and manage documentation for a system, based on the software lifecycle. Documents from requirements, design implementation and testing stages are stored. It is then possible for hypertext style links to be added so that traceability between the documents can be seen. This is of great use to the maintenance staff if the documentation has been prepared using this method. It is also easier to know which sections of the documentation will need to be changed if a section of the source is changed. It can also give details on other areas of the code that may be affected by a change. If a complete redocumentation is required, then this system would be of use, but for incremental documentation it would be of limited use.

SODOS

SODOS (Software Documentation Support)[32] is a tool based on data abstraction mechanisms to help with the definition and manipulation of software during the definition stage. SODOS is used on top of a data base management system and an object based model of the software life cycle consisting of requirements definition, functional requirements, architectural design, implementation, integration and test, maintenance, and configuration management. In the construction of documents, all information is put into the structural database so that it can be referred to at a later point in the development or maintenance. By use of a structured model within SODOS, the traceability of information should be apparent. This has great use in the maintenance phase, as requirements can be traced through to the implementation and testing.

DIF

The Documentation Integration Facility (DIF)[27] is a hypertext based system to manage and integrate the documents produced throughout the system lifecycle. Not only does it integrate documents from within one project, but it also integrates the documents from several projects. The eight point lifecycle used is traditional, only relating to the design, implementation, and testing stages, and not the maintenance stage of the software though it does have facilities for a maintenance manual. DIF provides facilities for the parallel development of documents, and the reuse of documents already constructed.

Also of greater interest are the facilities for browsing the documents with the use of hypertext links. During the development, information on the software is stored in textual objects or hypertext nodes. The user may navigate between the various nodes of information by way of hypertext links. As Fletton[23] points out, the effectiveness of this system is dependent on the skills of the author in splitting the documentation into nodes, which in the case of software documentation is not always obvious.

SOFTLIB

SOFTLIB[58] is not a tool to help with the preparation of documentation, but a documentation library based under UNIX. Documentation is split into sections relating to software components, defined as an area of code that has a specification. This is a very different approach to the lifecycle approach used by DIF, Fortune and SODOS. The authors of the system claim that increased reuse of the software is obtained if this method is used. However this approach makes gaining an overview of the system difficult and can also make it difficult to see how the overall requirements of the system relate to the implementation. It does however provide useful facilities to help with the checking of consistency between terms in a set of documents produced for any one system. Completeness of the documentation is also checked as each part of the software has a pre-defined document that is required to be completed.

Concordia

Concordia[65] is a development environment for technical writers, acting as an extension to Genera, the software development environment provided on

Symbolics computers. The central component of the system is a document database of independently accessible records maintained and written using Concordia. Retrieval and reading of the documents from the database may be made using the Document Examiner[64], a window based utility. Concordia itself provides the framework, organizing the tasks and activities in the document lifecycle using the three sub-activities of text editing, graph editing and previewing, of which the former is the most used. With a typical WYSIWYG editor, the appearance of the final document is presented as the issue of greatest importance, leaving the user to concentrate on the structure and content to a lesser extent. Concordia moves the emphasis to categorise the information for display - text, headings, tables, and lists, using a generic markup language so as to make the author concentrate on content rather than form. This is an important issue, and a system such as Concordia capable of maintaining the configuration of a large number of documents could prove useful in a maintenance situation ensuring that documents of quality information rather than quality presentation are produced.

Documents written using Concordia are stored in a hypertext database. This documentation is then viewed using the Document Examiner. This a hypertext based system allowing the user to move around the documentation stored within the database by going from one hypertext node to another. The user can also be presented with a graphical view of their current position in relation to other documents present and the part of the system they are currently examining.

3.5.3 Other Documentation Tools

Several tools also exist which don't fit into the previous two categories. These are discussed in this section.

DOCMAN

The Documentation Based on Cross Referencing Tool (DOCMAN) has been developed by BT in collaboration with the University of Durham. The tool uses three main components[25]:

- **UXREF**

  A mixed language, multiple file cross-referencer.

- **TEXTREF**

  A tool for adding documentation information to the cross reference.

- **IXREF**

  Interactive interface to UXREF and TEXTREF.

Thus the user may display textual information on selected variables and enhance the present documentation. The authors suggest the use of three distinct types of documentation:

- **Encyclopædia Documentation**

  Detailed information containing information to help with the analysis of the code, providing information on the exact nature and use of variables, function return results, etc.

- **Glossary Documentation**

  Reference information on the definition of terms and phrases with special reference to a specific system. For instance, the term *quality assurance* can refer to a general statement when used within certain projects, and to a specific set of goals which have to be achieved in other instances. In the case of the latter the glossary should contain information on the exact goals to be achieved leaving the reader in no doubt to the standards that have to be met.

- **Overview Documentation**

  General information on the system giving information that would be particularly useful to the new maintainer of the system. This documentation is designed to give a general overview of the whole system, describing, for instance, how the system fits together and a high level view of how the various programs interface with each other.

Since the original work by Foster and Munro described above, Fletton[23,24] has developed the ideas by adding a hypertext based front end onto DOCMAN. This interface allows the user to view the code, and documentation in parallel. The system takes the information from TEXTREF and sets up the hypertext links between the code and documentation. The user can then highlight one of the referenced items and automatically find other occurrences of the variable in the code, or any reference to it in the documentation. This therefore allows for rapid viewing and analysis of the source code enabling far quicker understanding of the system.

SEELA

As part of a full reverse engineering tool, SEELA[31] provides a tool to help with the maintenance of structured programs written in a variety of third generation languages including COBOL. This is done with the use of four productivity tools: a structure editor, a program browser, a program formatter, and a program documenter. The system analyzes the source code and displays it so that it is represented in a program design language (PDL). The reader is then made to focus their attention in the form of program blocks, each block appearing in a window. As well as providing an environment for documentation, module cross reference listings and other similar reports could also be produced.

One of the disadvantages of the system is the possible loss of code documentation as code is displayed in the PDL. Also by allowing users to move program blocks easily around using cut and paste techniques, care has to be observed that code is not "hacked" by the user and that proper design methods are maintained. Though a great deal of facilities exist for automatically generated documentation, no facilities exist for maintenance or production of any traditional textual information.

PAID

The Partially Automated In-line Documentation tool[33] uses software metrics to determine where comments should be present within the source code of a program. The guidelines for where comments should be inserted are as followed:

- At the beginning of the program to give the name of the author, title of the program, object of the program and methods used by the program.

- In the declaration section to explain variables, data types, record structures, etc. used in the program.

- At the beginning and end of each block.

- Within the program and/or block body as deemed necessary (particularly in sections of code that are notoriously difficult to maintain, such as recursive routines).

Although the first three guidelines are implemented easily, problems occur trying to implement the fourth guideline. PAID uses a textual complexity measure to determine the need for documentation. This fourth guideline is going to be different for different languages, for instance, pointers in C can cause particular problems. Having determined where comments are required, and if no such comments exist, the user is asked to add appropriate documentation. As stated previously, only a relatively small area of the code needs maintenance, and therefore the need to go through the whole program and comment it is questionable. Also this tool only makes sure documentation is present at critical points, but it does not concentrate on the quality of the information provided. This could be a serious omission.

3.6 Summary

At present the documentation tools that exist are mainly for use in the development stage of the software lifecycle, where the documentation initially is formed from user requirements and refined through the design process to eventually produce code. This is especially true of documentation environment tools. In software maintenance the emphasis is more on analysis of the code, and producing documentation from this analysis. The tools for producing documentation in the maintenance phase are, in the main, cross reference or structure chart generators, though there are now tools to help with the production of traditional text for use in the maintenance phase, though these are aimed at code level documentation.

All the documentation tools reviewed are, however, incomplete for use in third party software maintenance, for generally they have no facilities for providing an overview of a system or site on which the systems reside. This overview is essential during the initial take-over of the maintenance of a system described in section 2.3. Also none of the documentation tools produce documentation to help with the management of maintenance. Finally the documentation tools in production are also incapable of dealing with previously produced paper-based text. A solution to overcome these problems is discussed in the remainder of the thesis.

Chapter 4

Requirements for a Redocumentation Tool

4.1 Introduction

This chapter describes the requirements for a redocumentation tool. First the facilities which have been identified as necessary for use in a third party situation are discussed. Secondly the environment in which the author anticipates that the tool will be used is described.

The information collected is based on that supplied by AGS Information Services, one of the largest international third party software maintainers, and for whom the research has been conducted. The requirements for a redocumentation tool, outlined below, are derived from the requirements identified from the analysis of their operation at AGS's head office, in London, and the Rank Xerox UK Ltd. site in Uxbridge, Middlesex.

4.2 Required Facilities for a Redocumentation Tool

In an ideal situation a complete documentation tool should allow the user to write a full set of documents for any given system. When dealing with redocumentation it is important to consider any documentation that might already be in existence. Within a third party maintenance environment, these documents may be of vastly different formats. As stated in Chapter 3, in many cases up to 80% of the maintenance effort is expended on only 20% of the code and thus it may not always be necessary to produce a complete set of documentation. A tool therefore, to be used in redocumentation, should not force the maintainer into completely redocumenting a system, but should allow for the documentation of only the part of the system causing the greatest maintenance effort.

At this point it should be noted that the author does not condone partial documentation of a software system at the time of production as this negligent attitude has contributed to the maintenance problem. As far as is possible, documentation should be 100% complete on delivery to the users. It is also essential to remember that a document may have several different versions for different versions of the software, all of which have to be catered for. With reference to these points, an eight point overview of facilities was constructed by Freeman and Munro[26]. This has since been refined and expanded and now forms the basis of the requirements for a redocumentation tool. These ten requirements will now be described.

4.2.1 Support of Four Documentation Activities

Documents which are useful must undergo changes in parallel with the system to prevent them loosing their applicability. Changes will come under four headings:

- Corrections to the documentation.

- Updates to the documentation so that it remains relevant to the updated system.
- Additions to the documentation to give the reader added information.
- Changes to the documentation to increase its maintainability.

The first requires replacement to parts of the existing documents to increase the accuracy of the text. The second requires changes to part or all of the documentation so that the additional functionality of the system, or the new environment is represented in the documentation, thus making sure consistency is maintained between the code and text. The third requires that an additional set of documentation is prepared so that the documentation covers all parts of the system, while the fourth may improve the readability or layout of the text, but not the contents. A documentation system must be able to cope with these four maintenance documentation activities, which are directly equivalent to the four maintenance activities listed in Chapter 2.

4.2.2 Configuration Control of Documents

During a system's life time code is frequently changed, and as stated above, the documentation should also be altered. Often several versions of the code will exist running on different systems, or with different functionality. It is essential for a documentation tool to be able to cater for these different versions, making sure consistency is maintained between the code and documentation.

4.2.3 QA Procedures

It is often necessary for the quality of a document to be assessed so as to assertain whether it is being updated properly. By applying built-in procedures which show details of how the document is being updated, a manager can easily acquire the information to determine the success of the documentation policy, in use during a particular project. The quality assurance aspect can also apply to the code. By analysing the documentation being produced, it is possible to see which areas of the code are producing most problems. As discussed earlier, only a small part of the code seems to cause most problems, thus, if the documentation can show which parts of the code are causing the greatest maintenance effort, procedures to carry out preventative maintenance on these areas can be set in motion. This is only possible if documentation is being produced by the maintenance staff.

4.2.4 Enforceable Standardisation of Document Layouts

Often, documents are written in a very ad-hoc manner, to the author's own design. This can lead to frequent omissions and make it difficult for other users to read the documentation. If documentation is written to a high quality standard initially, the need for redocumentation would be vastly reduced. It would not, however, be completely removed as document maintenance is required, in parallel with software maintenance.

Having an enforceable standard partly overcomes the problem of omissions in the text. As stated earlier in this thesis, a third party software maintenance environment can cause problems with trying to gain uniform standards for documentation across all the client sites. This is due to different sites having their own in-house standards, and in this case it would not be appropriate for a contract company to enforce new standards, unless requested. There is however a possibility for documentation to refer to existing standards of code and documentation. The third party maintainer should attempt to bring the previously produced documentation up to the standard set by the parent company, and therefore any tool should help with this. It is however acknowledged that the maintainer's priorities lie in improving the documentation relating to the code taking the greatest maintenance effort, and that this is brought up to the pre-defined standard.

4.2.5 Facilities to Incorporate Any Existing Documentation

It is essential to be able to incorporate any existing documents, in the suite of documents for a system, without having to retype them. This is due to several factors. Firstly the cost of retyping the documentation for a complete system, so that it is in machine readable form, would be immense, both in manpower and processor time. Also, as stated earlier, a lot of the documentation is out of date and/or of little use to the maintenance programmer. If this is the case it would be pointless retyping documents whose use will be, at best, minimal. It is therefore desirable for a documentation tool to be able to incorporate details of this *external* documentation so that it can be traced from within the tool. This should help to alleviate needless expenditure.

If, at a later date, a piece of information is found to be in regular use, and found to be accurate, then a procedure should exist for making the information available, on line, through the documentation tool. At this point the documentation should be brought up to the latest desired standards resident within the company. It is for this reason that the comparatively simple use of a scanner to read in the text is not advised. Instead a rigorous procedure making sure that any inaccuracies or inconsistencies within the currently existing data should be enforced.

4.2.6 Automated Parallel Viewing of Documents

As stated in Chapter 3, most COBOL compilers are capable of producing a cross reference listing. It would be ideal for the maintenance personnel to be able to view the code at the same time as the cross reference information, which has frequently been found to be beneficial when analysing how the code uses areas of working storage or functions. By using a windows based system, this is possible. It is also desirable to be able to present the documentation on screen at the same time, so as to increase the speed of analysis even more by increasing the parallelism of the viewing.

Having the three types of textual information on screen as described above, though making systems analysis more efficient, does not have as higher a productivity gain as can be achieved when the documents are linked. This gain is due to all the information present on the area of immediate interest being presented to the reader; this removes the need for the searching of documents to retrieve required information. Being able to scan the code, and at the touch of a button, being able to read any relevant documentation and cross reference listing relating to the chosen area, decreases the effort in code analysis by reducing the effort taken to retrieve information.

4.2.7 Creation of a Hierarchical Document Structure

Documentation should be at various levels of detail, ranging from a broad overview of the client site which would help to act as an introduction to new members of the maintenance team, to very detailed descriptions of certain areas of the code. The documentation could be envisaged as a tree-like structure, the user starting at the root node, and then working through the appropriate number of levels by traversing links, and possibly to a leaf node if that level of information is required. By incorporating such a method, which also embeds a high level form of information hiding, or more strictly a method where the user only receives information on a restricted area, the new maintainer is not swamped by information, and the seasoned maintainer not distracted by introductory information which is of no concern to him.

4.2.8 Casual Updating of the Documentation

Documentation, as well as being viewed casually, should also be capable of being updated casually; thus a maintenance programmer can add information on the system at various points through the system's life. This allows for a more complete set of documentation in the long run. The problems which can occur with this method are making sure that the accuracy and completeness of the documentation is maintained. Regular checks have to be made to make sure that the authors of any documentation are, firstly, making any necessary corrections to documents that are found to be in error or lacking information, and secondly that what they write is correct and of a pre-defined quality.

4.2.9 Easy And Interesting To Use

One of the major problems with documentation is that it has not been updated, or not completed in the first place. A good tool will encourage the users to produce documentation for their system eliminating this problem. If the system is too complicated to use then time, and therefore money, will be wasted by the operators learning to use it. A very complicated system will also discourage the casual user. A system based upon the mouse paradigm of point and click, providing a simple user interface, such as Windows 3.0, is therefore appropriate.

4.2.10 Selected Viewing of Areas of the Code

When viewing a large program, it is often easy to become distracted by code which is of no immediate relevance. For instance when working on the analysis of the use of a variable, the analyst often only requires details of the setting, use, and change of that variable without having other areas of code to confuse the issue. It would therefore be desirable to be able to view selected areas of the code, for instance only viewing the areas concerning a single variable. By using a windowing system as described in Section 4.2.5, it is possible to display a restricted view of the program listing in parallel with code and documentation to help with code analysis, and make the understanding of certain areas of the code easier. Again this is implementing the principle of information hiding.

4.3 Operational Environment

For a redocumentation system to be of any use it will have to be tailored for the environment in which the documentation will be needed. In a general maintenance environment, documentation, whether existing or newly written, is only needed on the site where the system is resident. With third party software maintenance the situation is not as clear. Details on a site and the working environment present at the site have to be accessible at the head office of the company doing the software maintenance. This is so that any new staff of a maintenance team can become familiar with their working environment. Some of this information may also be required at the site, though in an abridged form so that information which could be regarded as sensitive to the client is not available.

Therefore there are two distinct environments for the use of the documentation, the maintenance site, and the contracted company's head office. We shall deal with these two issues separately.

4.3.1 Site Environment

The major data processing facilities owned by the vast majority of AGS's customers use IBM mainframes running systems written in various dialects of COBOL. Other languages which are used are in the main PL/1 and Fortran. We are initially only going to concentrate on the redocumentation of COBOL programs as they are by far the most common.

The redocumentation system has to be able to communicate with these IBM mainframes, though for future implementations it must also be flexible enough to be able to cope with the now increasingly diverse hardware set-ups.

In an ideal world it would be desirable to have the redocumentation tool resident on the mainframes. There are, however, several arguments against this. Most of the computers do not have the graphical support necessary to display windows based WYSIWYG editors already identified as useful. Secondly processing time is at a premium on most of these machines, and is charged. Most windows and hypertext based programs are extremely memory intensive, and thus, having a documentation tool running could seriously damage the performance of other jobs. Finally, space is often extremely limited on these machines and also charged for. Therefore the tool will have to be resident on a host which has connections to the mainframe so that it is not resident in chargeable space.

Most sites now have a selection of IBM Personal Computers (or clones) which are often connected to the mainframe using an X25 link or similar. It is therefore possible for one to download software or text from the mainframe to the P.C. Several machines, including the P.C., Apple Macintosh, and Xerox 6085 system, were considered for the role of running the documentation tool. Even though the latter two are probably better machines in terms of the facilities for text processing, the P.C. was thought to be the optimum environment due to their presence within most of the large data processing sites. COBOL listing files and any documentation can be down-loaded to the P.C. from the mainframe and the user does then not have to be concerned with incurring large bills or the degradation of other users' jobs because of the excessive use of the mainframe.

Having established the hardware for such an environment, the issues of software must be considered. At the requirements stage this cannot, and should not, be finalised as this is more of a design and implementation priority. It is, however, reasonable to assume that the software will reside on a windows based environment with facilities to enable the implementation of links enabling the automated parallel viewing of documents detailed earlier in this chapter.

4.3.2 Head Office

Documentation will also need to be stored at AGS's headquarters where documents from the top of a hierarchical structure are of most use. Current documentation giving a high level overview of the client sites already exists on a Xerox 6085 machine running the Viewpoint operating system. Though there are obvious problems associated with having documents in different environments, the documents that exist and are required at head office are different from those used on site. The problems of moving the documentation from one environment to another therefore outweigh the difficulties in retyping the documentation on a P.C. based system. Therefore the parts of the system to be used at the head office will be required to run on the Xerox 6085 running the Viewpoint operating system.

4.4 Summary

The redocumentation aid will be resident in two separate environments, P.C. based for the information to be used on site, and Xerox based for that residing at the head office. Ten major requirements regarding the facilities that have to be present for a redocumentation system for use in third party software maintenance have been identified. Two of the requirements listed specify the need to generate links between or within documents. It is this area that is discussed in Chapter 5. The remaining requirements form the basis of the documentation tool described in subsequent chapters of this thesis.

Chapter 5

Hypertext

5.1 Introduction

This chapter describes the basic principles of hypertext, a system based on nodes of information with traversable links connecting each node. An overview of hypertext origins from non-computerized card storage systems to present day systems is given. The facilities and typical uses of hypertext system is then discussed. The use of links within a redocumentation and third party software maintenance tool has already been identified. The use of hypertext for setting up links used in software maintenance is evaluated and is discussed in this chapter.

5.2 The Origins of Hypertext

Hypertext is an approach to information management in which data is stored in a network of nodes connected by links, with each of these nodes containing, for example, text, graphics, audio or video information, as well as source code[54]. The idea of a hypertext system was first conceived by Bush[12] in 1945 with his description of "Memex". This system enabled the storing of books, records, and communications while allowing increased efficiency in the viewing of the documents. This was done by modelling the storage system so that it was similar in structure to that of the human memory, operating by association. When the mind focuses on an area, it can snap instantaneously to the next area suggested by the association of thoughts.

This concept of a complex web of associations being implemented using software was first put forward by one of the pioneers of hypertext, Nelson[47], and was first demonstrated in 1968 by Engelbart[22]. The basic concept applied to computers, as described by Conklin[19], is quite simple:

> Windows on the screen are associated with objects in a database, and links are provided between these objects, both graphically (as labelled tokens) and in the database (as pointers).

An example of the correspondence between the two is shown in Figure 3. In window A, a word b, also a link, has been highlighted. The pointer in the database corresponding to link word b causes window B to be opened with the text from node B in the database, displayed.

In a standard Hypertext application, hundreds of these links can be expected. The generation of individual links is often relatively simple with the use of a graphical window based environments allowing the user to create the links by specifying the connections between two nodes using a mouse[69]. However it requires skill to structure a document to make the greatest use of hypertext's flexibility and structure.

5.3 Uses and Facilities of Hypertext Systems

One of the most basic forms of hypertext system is the set of 3" $\times$ 5" cards that are often used for note taking. The cards typically have two fields, the first a keyword

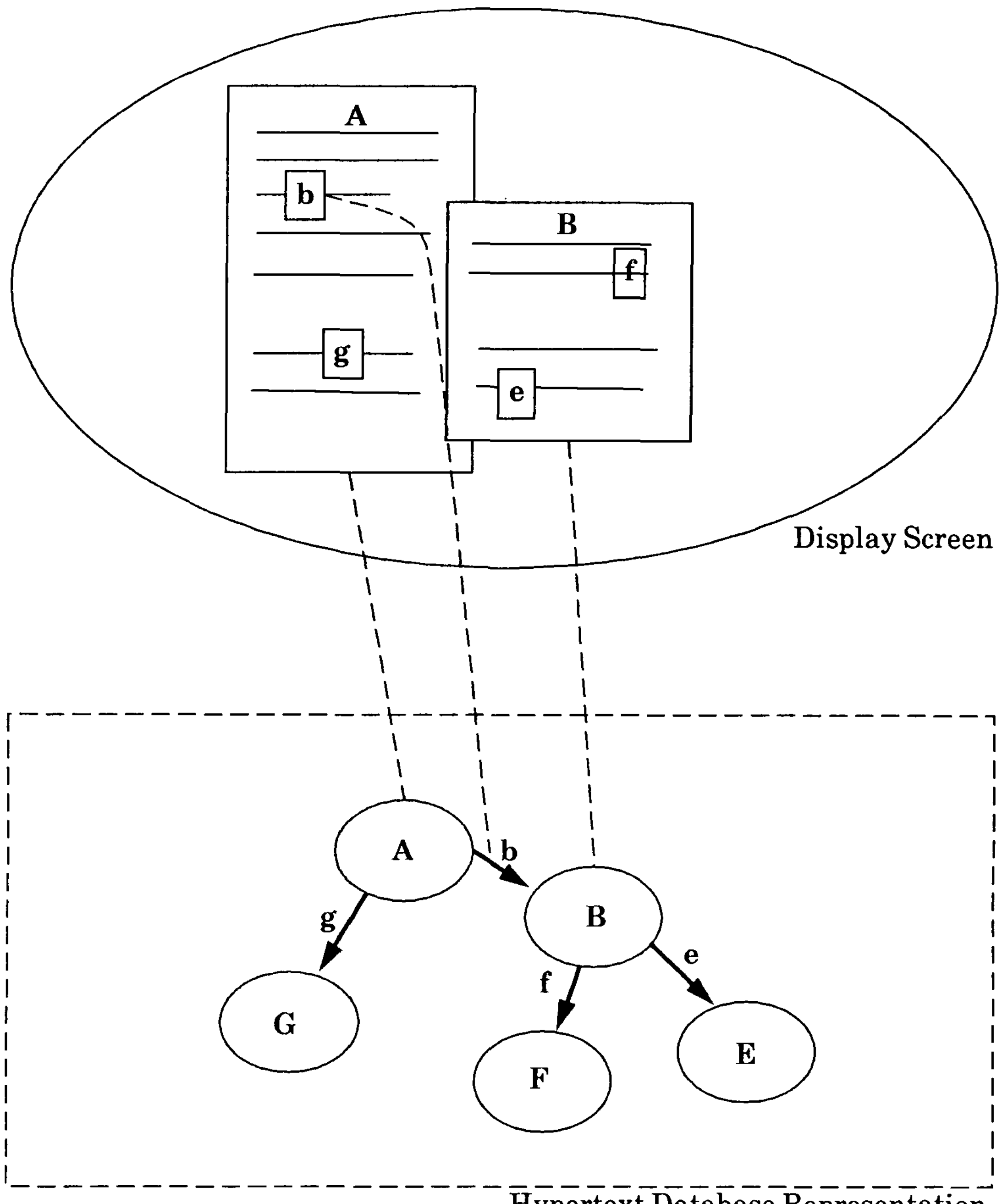


**Figure 3
Hypertext Representation**

on which the card will be indexed, and the second the detailed text relating to the first field. These cards are then stored in a box either in alphabetical order, or to some predefined sorting scheme. Dictionaries with cross references to other sections, and Bibles with references to other books, chapters and verses, are examples of other forms of hypertext system, as to some extent is this thesis with cross references to other papers. These basic forms of hypertext have been in existence long before Bush formalised the ideas of a system working on links. The ideas behind hypertext have therefore shown its use and applicability due to its continued use over several centuries though in a somewhat primitive form.

More modern hypertext systems, and the type which are of interest to the computer scientist, are non-manual, computer-based systems. Such systems normally contain the following facilities:

- **Link and Node Database**
  There is a link node database that contains the information which the system uses to allow navigation through the hypertext system. The nodes can be textual, graphical, audio or even executable code which is run when the node is accessed.

- **Multiple Window Interface**
  Each window that is displayed only holds the information held in one node of the database. A hypertext system does not fulfill its potential if the user can only display one node, or window of information. By being able to view several pieces of related information at one time the speed of comprehension can be increased, as the reader can cross reference between the related documents without having to flick between nodes of information. It should be noted, however, that allowing too many pieces of information to be displayed at once can also lead to confusion because of the quantity of information present on the screen at any one time.

- **Multiple Links Within a Document**
  As can be seen in Figure 3 each node can have more than one link. If each node has only one link, then the nodes become ordered in the form of a list and therefore form a sequential document.

- **Multiple Browsing Options**

  As well as being able to scan forward from node to node, most hypertext systems allow the user to navigate by using an index of terms, and also by reviewing the route that has been taken to get to the present information. The latter is important as it is often easy to get lost within the complex web of a hypertext system. The ability to backtrack is also widely used and thus saves the reader from having to repeat the traversal of some or most of the links to reach a piece of information that had been viewed only a few steps previously.

The use of computer based hypertext systems has not been as great as could have been expected with the prevalence of cross referencing in manual text based systems, as described at the beginning of this section. Although with computerised documents, cross reference information often exists, it is still rare for this to be automated. This could be due to the difficulties and time required to set up accurate links. Also when a change is made that, either expands or decreases the amount of text, a considerable amount of effort has to be expended to update the links unless an automated process is present.

Despite these problems there are a great number of successful on-line hypertext implementations of which a detailed survey has been completed by Conklin[19]. The implementations can be divided into four main types: general hypertext systems, problem exploration systems, macro literary systems, and documentation browsing systems. Details about each of these will now be given.

5.3.1 General Hypertext Systems

General hypertext systems are mainly designed to allow a user to develop their own hypertext systems for a number of different applications rather than being applications in their own right. These were of particular interest to this research because of the possibilities of using one of the systems to help with the production of links within the documentation tool.

The most well known version of a full hypertext system is NoteCards[30, 61] produced by Xerox PARC, the original home of the present WIMP environments. The system runs on the Xerox LISP machine and is highly regarded. It is not always a simple process to construct a complete system, because NoteCards has a complex LISP based programmer interface. This interface is the reason that the system did not meet one of its explicit design goals: that minor changes to NoteCards should be achievable with a small amount of work by casual, non-programming users. Despite this, the system is probably the best of its generation with excellent use of text and graphics, and a highly usable browser option showing the links present in either a part or the whole system.

ViewCards[69] was developed from NoteCards to run on the Xerox 6085 Documenter System. It was designed to be used by the non-programmer, with no coding experience required. Unfortunately the setting up of links cannot be automated and only very basic commands are available in an attempt to make its use more widely spread. This has had the effect of making the system less desirable for the serious hypertext developer. It has however retained the excellent use of graphics and text along with the browser option.

One of the more recent, and very popular systems, is HyperCard[3] developed by Apple and shipped with all new Macintosh computers. The system is similar to

NoteCards, with a complete programming language HyperTalk, allowing the user to make a customised browser. Using the Apple's WIMP interface, complex systems can be constructed relatively easily. One problem with HyperCard exists, namely that link destinations are only allowed to be cards and not points within cards[3].

Guide[9] was initially a research project at the University of Kent and was first available running under UNIX, but has since been developed by Office Workstations Ltd. to run also on the IBM P.C. and the Apple Macintosh. As Brown[10] describes, Guide isolates the user from the underlying structure of the data by displaying the text as a single scroll, rather than in separate windows around the screen. The major mechanism within Guide is the Replacement Button. This button is replaced by material linked with that button, thus expanding an area of a document to give greater detail. A second method for expanding the information present is with the use of Note Buttons. These are present on the Macintosh and P.C. versions of Guide. A Note Button causes a pop up window to display the replacement text while the mouse button is held down. Glossary Buttons, present in the UNIX version of Guide, are similar to Note Buttons. However, they display the replacement text in the bottom section of the screen. The final linking mechanism found in the Macintosh and P.C. versions is the Reference Button. This link is a more typical hypertext link causing a jump to a different point in the document, or either to a different document. In section 4.2.6 the need for parallel viewing of documents was discussed. The Guide system would not be suitable for this due to the use of a single scroll.

5.3.2 Problem Exploration Systems

Problem exploration systems aim to help with the initial understanding of a tool or problem area where traditional sequential texts may not be as useful as a hypertext system. Their usefulness derives from the fact that at the early stages of the thought process on a certain topic, various unconnected ideas may form and therefore a less restrictive form of guide is often found useful. These systems also have the advantage of normally presenting the reader with small chunks of relevant information so that their thought process is enhanced, rather than swamping them with large chunks of information which can occur with traditional texts. One of the best known systems of this type is the research project IBIS (Issue Based Information System)[51] which is used to help solve problems that have no clear problem space. As well as custom designed hypertext developments to implement problem solving systems, Marshall[42] describes a system implemented using NoteCards to help with policy making decisions and for capturing the structure of a political organisation. The success of this project shows the wide range of uses that general hypertext systems can have in other specialized hypertext fields.

5.3.3 Macro Literary Systems

Typically macro literary systems help to support and create large on-line libraries where there may be connections or links between the various documents in the collection. One system called Xanadu[45] was created by Nelson, founder of the term Hypertext. This system saves the initial version of each document and any changes made to the document, and thus does not have to save a full copy of each version of the document.

5.3.4 Documentation Browsing Systems

Document browsing systems have a variety of uses to help with gaining the maximum amount of information from blocks of documentation within the smallest amount of time. Typical applications include teaching, public information, and on-line help facilities, for instance the HELP facility used within Microsoft Windows 3.0[50]. Such a system allows a user to arrive at the information they require quickly by traversing a few links rather than having to scan several pages of text to find the relevant information. As described earlier, hypertext allows non-linear thinking, and this is put to great use in the Storyspace system described by Bolter and Joyce[8]. The system allows the user to interact with fiction, and help make up a story as they choose by the way they navigate through the text.

The uses of hypertext and documentation can also be applied to the software lifecycle, and it is this, with special reference to the maintenance section, that is described in the next section.

5.4 Hypertext and Redocumentation for Software Maintenance

There have been a number of hypertext tools developed to help with the construction or displaying of documentation. As has been stressed before, hypertext is particularly apt at displaying text which has a great number of references either to other documents, or to other sections within a document. Also, and of particular interest, hypertext can be used to help with the documentation of areas of the software lifecycle. Because of the references between, for instance, the specification and design, that should exist in a well developed system, there will be a great number of cross references both between and within documents. Therefore implementation under hypertext can help to ensure consistency between the various stages of the lifecycle.

The arguments for using hypertext in the development stage extend equally to software maintenance. As in the development stage, documents created within the maintenance phase often refer to previously produced texts, and therefore there is a considerable argument for using a hypertext system in software maintenance. One very major problem that arises is that it is very rare for system documentation to be present in a hypertext form, and thus an uneconomic amount of work would be required to redocument the system using hypertext. It is for this reason that, unless a system has already been written using FORTUNE or SODOS, redocumentation using either of these tools would be uneconomic.

In contrast, the documentation system proposed by Fletton[24] and discussed with reference to redocumentation in Chapter 3 describes a wider use of hypertext within software maintenance, that of helping to redocument code and providing a hypertext based analysis and documentation system. As described in Chapter 4 the redocumenting of systems within third party software maintenance requires a document hierarchy. Fletton's system concentrates on documentation for the code at a high level of detail and is ideal for code analysis. For third party software maintenance, a more comprehensive documentation approach has to be adopted.

As detailed in Section 4.3, the environments in which the documentation will reside are different, and as no common hypertext system can be used across the range of hardware and operating systems, different development systems will have to be used. For development of the systems developed for use at the AGS head office, the Xerox NoteCard system running on the 6085 Documenter is the only package available. For the systems to be developed for use on a P.C., the choice is less clear. As one of the requirements for the redocumentation system is to enable full parallel viewing of the documents, a hypertext system running under Microsoft Windows would be ideal, as the programming environment available under the Windows package would allow for implementation of the other facilities listed in Chapter 4. At the time of investigation just after the release of Windows 3.0, no hypertext system had been made available and therefore the hypertext links were implemented using the Microsoft Software Development Kit. The author acknowledges that this situation has changed with the release of several products such as Asymetrix's Tool Book.

5.5 Summary

The applicability of the use of hypertext in software maintenance has already been investigated by Fletton[7], and found to be beneficial. The use of hypertext in documentation tools, albeit for use within the development stage, has also been described and found to be useful. However, no single hypertext based documentation system fulfils all the requirements detailed in Chapter 4. The design and implementation of a system to meet these requirements are described later in this thesis.

Chapter 6

Documentation Structure For Use in Redocumentation

6.1 Introduction

Having established a basic set of requirements for a redocumentation tool and facilities for accessing the documentation, a documentation structure has been drawn up for use in the tool. The tool is called the Redocumentation Aid for the Maintenance of Software (RAMS).

The documentation structure consists of a set of five documentation types, as follows:

- Site Information.
- Management Documentation.
- Overview Documentation.
- System Documentation.
- Existing Documentation.

A detailed description of the information that appears in each of these documentation types appears later in this chapter.

The reason for this choice of documentation types within the RAMS system grew out of the requirements for a documentation aid to be used by a third party maintenance team within a client organisation, detailed previously, as well as AGS's internal standards for the production of documentation for use in such an environment. The RAMS system, however, has been designed so that each documentation type can be tailored to a particular client company's requests.

This chapter gives a detailed description of the design of the documentation types and their structure.

6.2 Document Structure

The five types of documentation listed above can be represented in a hierarchical structure, with the documentation types in increasing levels of abstraction with respect to the code as progression is made up the hierarchy. The site information can be regarded as being at the highest level giving an overview of a complete site and the systems present in that site. The management documentation gives an overview of the performance of all the systems, with facilities to focus on particular systems. The Overview documentation is more detailed, giving information on a particular system. At the bottom of the hierarchy is the system documentation which will provide the documentation for a module of a system in great detail. With this will be incorporated the existing documentation. A graphical representation of the hierarchy is shown in Figure 4.

A detailed description of each of the levels will now be given.

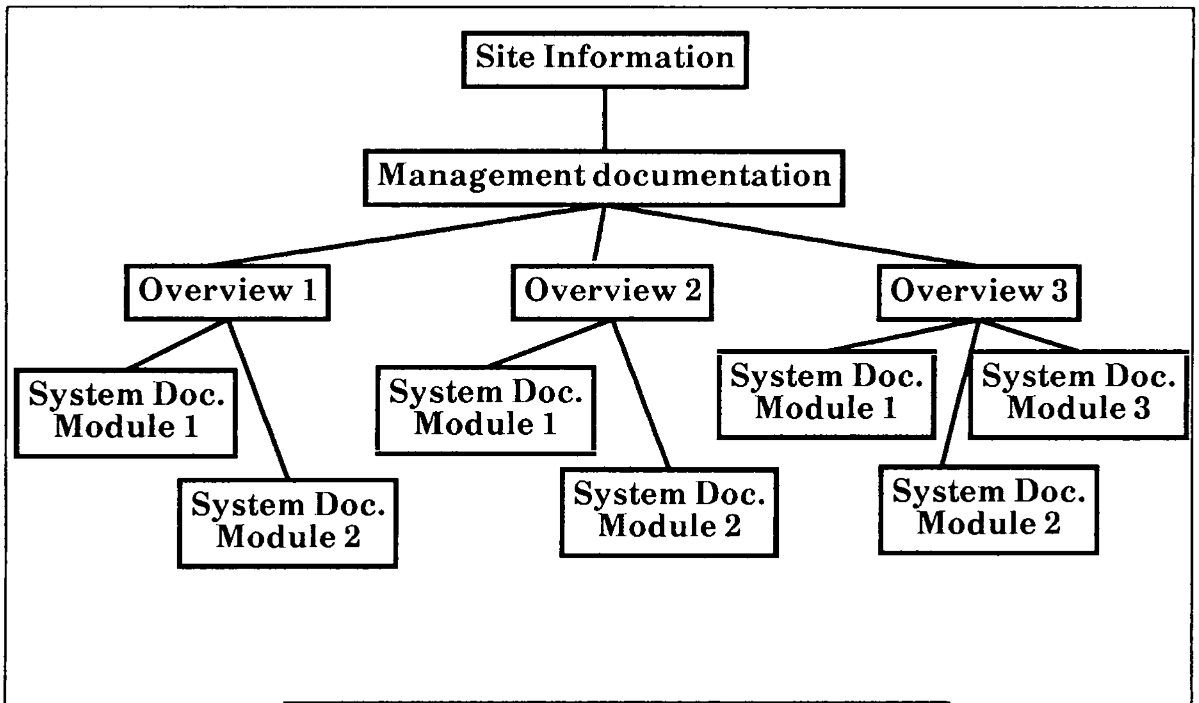


Figure 4
Documentation Hierarchy within RAMS

6.3 Site Information

Site information is at the top of the hierarchical documentation structure, providing a general introduction. It is compiled from information obtained in an initial third party maintenance proposal on the client site and information gathered after the acceptance of the proposal, such as office hours, commuting options, system hardware, operating systems present, languages supported, and systems to be maintained. This document fulfils a twofold purpose. Firstly to provide an introduction to new members of staff to a client site, and secondly to act as a point of reference. The information is already held by AGS on a Xerox 6085 documenter in the form of a standard report[1].

This document is divided into a number of distinct sections. Because the text is designed so that it does not have to read sequentially, it is ideal for implementation under a hypertext environment, with each section being displayed on an individual card, with links to related cards.

The contents of the document are being maintained without change as the document has already been frequently used and proved to be more than adequate in providing an overview.

Because the document is an overview, as well as acting as a point of reference, some details appear in more than one section. One typical example of this is some of the information that appears within the initial maintenance proposal section, also appears in later sections on site location. Again a hypertext solution to the structure can overcome this duplication which is some times necessary within a traditional text. Having a piece of data present in only one place helps to increase the maintainability of the documentation. This is because an update to a section can be made in a traditional document, while the person making the change is unaware of any further occurrences of the topic. This then leads to inconsistencies with the rest of the information contained within the document.

6.4 Management Documentation

As has been discussed, it is important to have as much information as possible to give back to the line managers, and in doing this it may help them understand some of the problems associated with maintenance. Information such as problem logs, containing details of problems and the time and action taken to fix them, would be kept along with change details, giving details of the extent of the change, the exact modifications made, and any other documents that were updated.

Documents such as procedures for change should be accessible in this section, thus setting standards for the maintenance of a system. This is valuable for third party maintainers, as a client company may have methods of change particular only to themselves, and it is vital that the maintenance staff have the information to follow these procedures. These documents often refer, not only to the change procedures for the code, but also to the in-house standards for changing the documentation. This information is also often found invaluable when gaining an overview of the site, and it is worth duplicating this information and having it present within the site information.

Because the documentation gives exact details on where maintenance is being carried out, it can show which particular areas of the system are causing problems. By highlighting these specific areas, it can show which areas of the system could be in need of restructuring. By being able to pinpoint problem areas in the system, and only having to restructure those causing an increase in maintenance effort, a great deal of expenditure can be saved by not restructuring or rewriting the whole system. This also has the advantage of leaving working code alone, and not replacing it with a, possibly, less reliable replacement. The author acknowledges the problems of interfacing the new code with this method of selective replacement of code which, in general, is not a trivial task. Ways of identifying links between areas of code have been the subject of a great deal of research, and are described at length by Calliss[14].

The management documentation will initially contain information generated from the production of lower level documentation. Information will also be obtained showing where the greatest amount of code analysis is being done. The data stored will give the system, program, and module name where the work is being carried out. The line and variable name which is being investigated and/or documented, and the amount of textual information produced will also be stored, to allow review at a later point. This information will be presented in the form of a table, allowing easy analysis of the data by management personnel to help in the definition of future maintenance directions for a system.

6.5 Overview Documentation

The overview documentation will provide a pictorial view of the structure of the system. This documentation will provide a tree-like representation of the structure of the system at various levels of refinement. For simple systems, it is easy to show the whole structure, but for systems of a typical size, the displaying of the complete system initially would only heighten the confusion of the analyst. By being able to traverse the structure gradually, expanding only the sections which they perceive to be of immediate use, the user becomes gradually familiar with the system.

For instance the top level may show separately compilable modules, the next level may show the sub modules of one particular module, and the lowest level may show the procedures within a module. Showing the actual links between the various modules and the way they interconnect as opposed to the procedures contained within a module, or procedures within a sub-module, is a far more difficult problem which is discussed at length by Calliss[14]. It is not expected to try to show these links in initial implementations.

Again an information hiding approach, or more strictly, an approach where the user will only be presented with information immediately relevant to them, has been maintained. The user will navigate hypertext links through a tree-like representation of the system. By choosing their route, and only seeing the part of the system they are requesting to see, they are not distracted by areas of the system with which they are not presently concerned. This set up is appropriate for two purposes, which are similar to the site information but on a system level. The first purpose is to give the beginner an introduction to a system without swamping them with information, and secondly the experienced user can select part of the diagrammatic representation of the system to refresh their memory.

6.6 Technical Documentation

The technical documentation could probably be described as the most useful form of documentation for code analysis, with the greatest potential for improving productivity. Within this section there are four types of documentation: source code, cross reference information, any related textual documentation, and a slice of the program. It will be possible to cross reference between all four documents, thus helping to increase the speed of code analysis and decrease the time taken to obtain a detailed understanding of the system.

One major feature of the system documentation tool is that of slicing the code, where lines of code will be displayed containing information relating to a search term given by the user. This approach to slicing has been simple with the slice based only on textual information as opposed to the cognitive psychological view proposed by Weiser[68]. Although a slice of code is not usually regarded as documentation, this form of interactive documentation is as useful as a lot of written text. This is partly due to the accuracy and directness of a slice, and therefore should pay a greater part in the analysis of a system.

Hypertext style links between all the documents will be set up to enable quicker viewing of the documents in parallel. Initially links between the variables listed within the cross reference information and the code will be enabled, and then links to the documents will be set up. The user, on selection of a variable will be able to view the code where the variable appears, and at the same time be able to view any relevant documentation in a separate window. The use of a window based mechanism with the hypertext links will incorporate the ideas of parallel viewing.

The textual documentation will be produced partly automatically, and partly by the maintainer. Information should be given for each variable and procedure within a piece of code. Information from the cross reference and any slices done on that variable will be able to be kept along with any comments which the maintainer feels appropriate.

6.7 Existing Documentation

As discussed previously, it is important to be able to incorporate existing documentation within the documentation tool. This saves a large effort in retyping existing papers, possibly with a great number of errors contained within them. Using a database with entries on the details of various sections, and their place in the already existing literature, or within the documentation written within RAMS, a maintenance query system is available. This will help the user track down information if they have a problem with maintaining any part of the system. The use of a knowledge base will help with the information search and give details on where to find the information, and contain a field to comment on the credibility of the documentation in question. If a piece of existing documentation is frequently used, the information should be typed into RAMS where it could be accessed more quickly and easily than if it were paper based.

There will always be problems maintaining this database because maintenance personnel are usually overworked, and in common with most software professionals, behind schedule. Thus the front end to such a database is of utmost importance, so that it provides hints to update the database at suitable points, but not to the extent that the user does not wish to use the system.

It is partly up to management to make sure this database is maintained, otherwise it will suffer the same lack of credibility that a great deal of paper based systems suffer. To help management keep a track on the use of this database, information on the extent of its use will be used within the management documention.

6.8 Summary

This chapter has presented five types of documentation that are required for use in third party software maintenance. Instances of these documentation types can be viewed as forming a hierarchy with the information ranging from a general overview of a client site to a detailed description of the source code for a particular component of a system. The implementation of a system containing all aspects of these documentation types is given in the next chapter.

Chapter 7

The Redocumentation Aid for the Maintenance of Software (RAMS)

7.1 Introduction

This section gives details of the design and, where appropriate, implementation of the four tools which implement the documentation structure described in Chapter 6. The four tools are called Site Information based on Hypertext (SIBOH), Overview Documenter, Management Documenter, and System Documentation Tool (SYSDOC). These tools form the Redocumentation Aid for the Maintenance of Software (RAMS).

The first tool to be implemented was the Site Information Based on Hypertext (see Section 7.2). The tool was developed first because it has been adapted from the site documentation, and a text based system was already available. Currently the System Documentation tool (see Section 7.5), developed from the technical documentation and existing documentation, is being tested on site. It was implemented second as it was this part of the tool that there was the greatest need for at the time of implementation. The Overview Documenter (see Section 7.3), based on the overview documentation, has also been prototyped. The Management Documenter (see Section 7.4), developed from the management documentation, will be the last to be coded as it will take information from the

System Documentation tool which therefore needs to be fully tested and operational.

7.2 SIBOH - Site Information Based On Hypertext

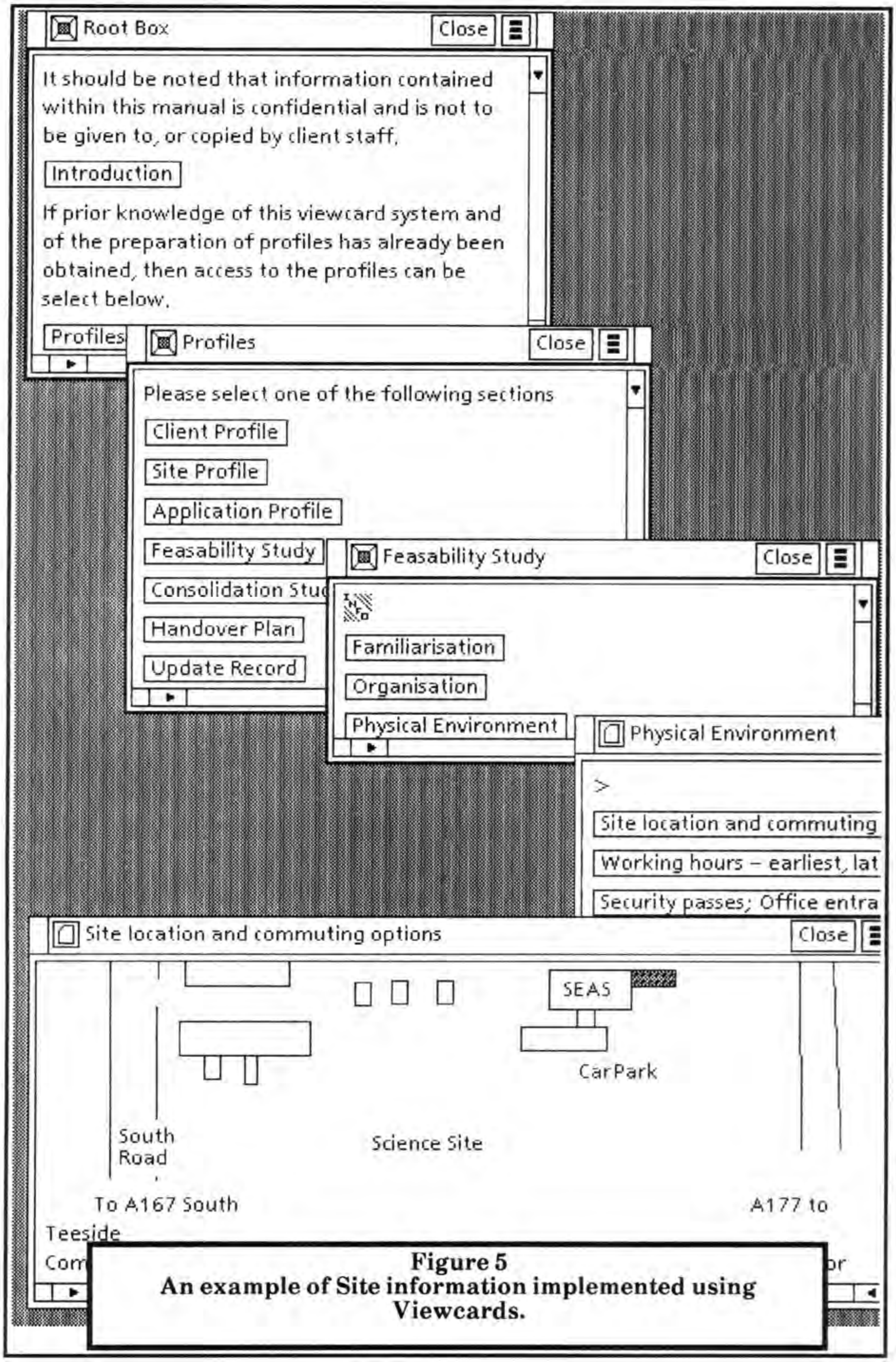
The SIBOH documenter was the first to be designed and implemented. This was due to the recent completion of the non-sequential site information document by AGS. The information had already been collected and was therefore available for incorporation into the new system; thus the transfer to hypertext was not going to be expensive in terms of man-hours. At the time of implementation, this form of document had only been used to store the information on one client site. It was therefore felt prudent to complete the implementation before further information had been entered into the traditional document.

As the information was already present on a Xerox 6085 documenter and, due to other relevant documents being stored on the system, it was felt that the hypertext system must also remain on this machine. As described in Chapter 5, the only hypertext system available for the 6085 documenter is Viewcards. This system is actually ideal for the simple implementation of a paper based document such as this, where links between sections of documents, rather than links between the contents of the documents are required. The document, which cannot be reproduced due to commercial confidentiality, has already been well structured, and therefore no major alterations to the basic form had to be made before implementation. There were several occasions where the same information was stored in different sub-sections, and by using hypertext this was eliminated, making updating of the information easier.

Viewcards has three types of node, card boxes, cards, and browser screens. The original document containing the site information was split into 7 distinct sections, and each of these has been implemented as a box. All seven were themselves contained within a box to represent the complete documentation for one site. Each of the sub-sections has been implemented as a card within the appropriate box, with links between related items in the cards. In a few cases, two levels of boxes have been used so as to maintain the structure of the original document. A browser option is available so that the total structure of the document, and the links between the cards can be seen.

Figure 5 shows an example of a SIBOH screen, where the user has opened the profiles box from the root box, and then in turn opened the feasibility study box and the physical environment card. Finally they have opened the link to display the site location and commuting options card. As can be seen, ViewCards allows excellent use of the full graphics environment present within the 6085 Documenter, and, as shown in this example, allows for links to other cards to be present within graphics.

The use of hypertext in this environment fulfils the principles of information hiding, described previously, with the user only seeing the details of the company or system which they require at a set point in time. It may be that some company details are confidential and that only certain cards are available for viewing by less senior company members. In this case, passwords can be added to cards or boxes allowing selective viewing of the information within a card, or the cards within a box. Also with the structure implemented using ViewCards, a hierarchy of documentation has been kept and extended, with the user obtaining more detailed information on a topic the deeper they explore the links between nodes.



Root Box

It should be noted that information contained within this manual is confidential and is not to be given to, or copied by client staff.

Introduction

If prior knowledge of this viewcard system and of the preparation of profiles has already been obtained, then access to the profiles can be select below.

Profiles

Please select one of the following sections

Client Profile
Site Profile
Application Profile
Feasability Study
Consolidation Stu
Handover Plan
Update Record

Feasability Study

Familiarisation
Organisation
Physical Environment

Physical Environment

Site location and commuting
Working hours – earliest, lat
Security passes; Office entra

Site location and commuting options

Figure 5
An example of Site information implemented using Viewcards.

7.3 Overview Documenter

The overview documenter is designed to help to introduce a new employee to a specific system, as opposed to a complete site as in the SIBOH documenter. Again the aim was to use information hiding so that the user would not be swamped with information on first entering the system. In general, most computer systems have some form of hierarchy, i.e. system, module, function, and instruction level, and it is the aim of the overview documenter to give a graphical representation of the top three of these levels.

The initial requirements were for the tool to be implemented on the Xerox 6085 and thus be present with the site information tool. This would allow connections between the two. A major problem apparent with this is that the ViewCards hypertext system has no low level programming language and thus an automated version of the viewer cannot be implemented on this system. In an ideal situation, the name of the system would be entered, and the tool would display the program structure without a user initially having to provide the input.

A prototype browser has been implemented using ViewCards. At the start of a session a system name has to be entered into the hypertext system so that the overview documenter will initially display the high level overview of the system in question, and then, when the user selects part of this overview, a more detailed view of that part of the system will be shown. An advantage of using ViewCards is the browser option. Instead of initially representing the system by physically placing boxes and cards within boxes and setting up links between cards, the user may use browser cards to show the graphical connections between the boxes, cards, and browsers. A browser is ideal for displaying levels of abstraction where little or no text appears, for instance showing system hierarchy.

In the example in Figure 6, a process has been displayed, and following that "Generate Report" and "Database Check" were selected, and thus a lower level of abstraction was shown relating to the aforementioned modules. If the user had then selected the "Main Heading" link in the "Generate Report" browser, a card with either the code for this operation or details on the action would be displayed. Alternatively another browser may have been displayed, if the operation was going to involve more than a few steps. This is a representation of a very simple system but shows the basic principles involved.

7.4 Management Documenter

It has already been stated that the primary aim of the management documentation is to report back information on the extent of the maintenance activity and give details about the tasks to be completed and those recently finished. The reporting procedure must neither make the users of the system feel that they are being spied upon, nor make the maintenance process slower due to the collection of results. Two approaches are available. The first is the collection of problem logs and lists of tasks to be completed, which is already carried out at most sites. The second is an automated system that is invisible to the maintainers. Information, such as the last update to the documentation or which data name or function was being examined, can be taken from the system documentation tool as this will give the most reliable record of the systems, and parts of the systems, being examined.

There are therefore two distinct parts to the management documenter. Firstly the collection of data, and the second the displaying of data in a form which will enable managers to easily see where the maintenance effort is being expanded. The information will be stored in an Omnis 5 database resident on a P.C. This will

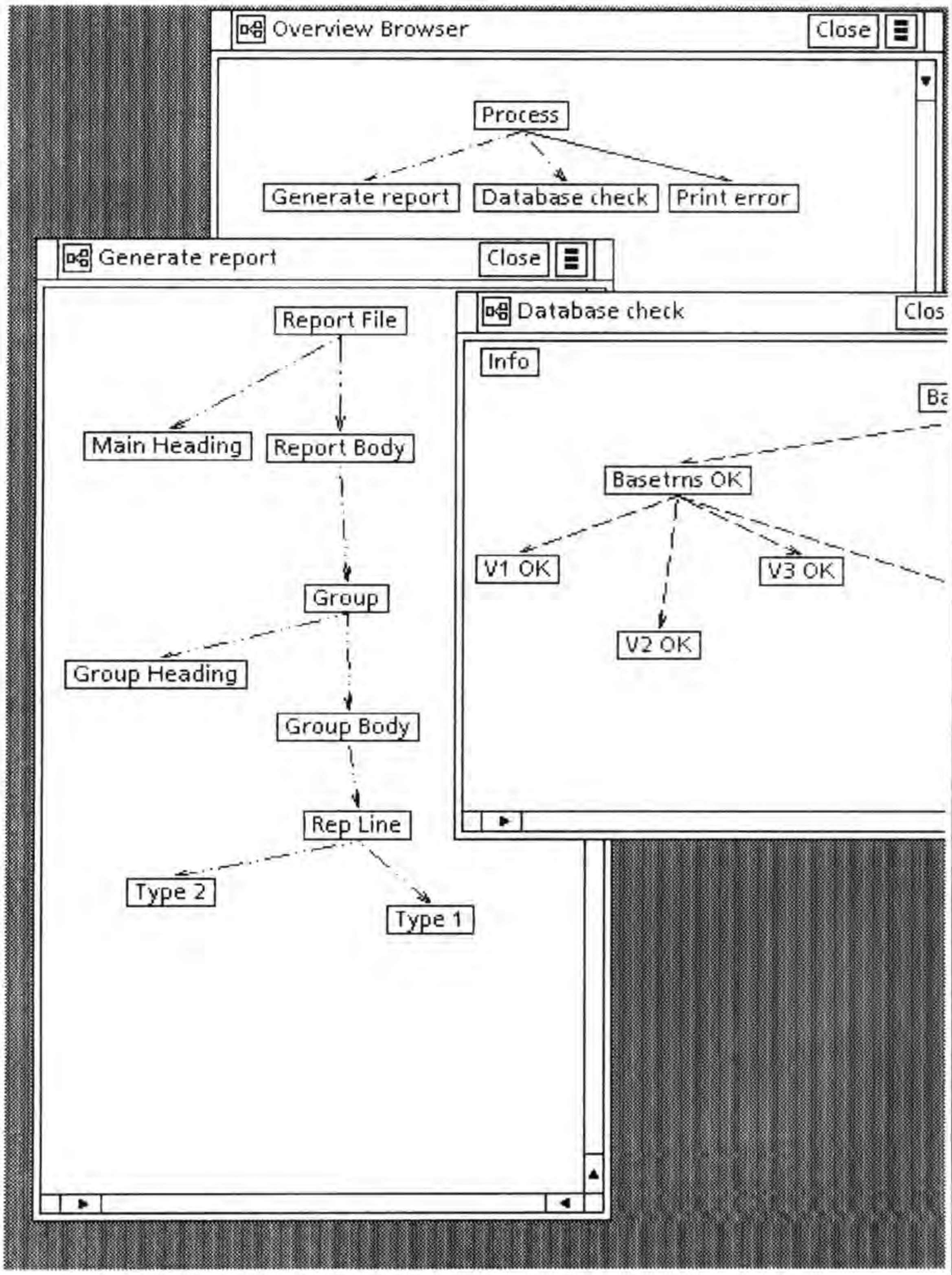


Figure 6
Site information example implemented using Viewcards

provide compatibility with the System Documentation tool discussed in section Section 7.5. At present the information is available in a variety of formats and there is no guarantee that the same type of information will be in a similar format at different sites. An example of this is that problem logs are hand written at one site and are available on line on an IBM mainframe on another site. The contents of the reports are also often different.

The requirements for the exact content of the documents have yet to be agreed with the users, and thus as yet implementation of either a method for collecting the data, or displaying the data, has been impossible. The data that will be collected included:

- Currently collected:
  - Current schedule of jobs for all systems
  - Jobs awaiting priority
  - Jobs awaiting priority great than three months
  - Fault reports
  - Change reports (including time for change and extent of change)

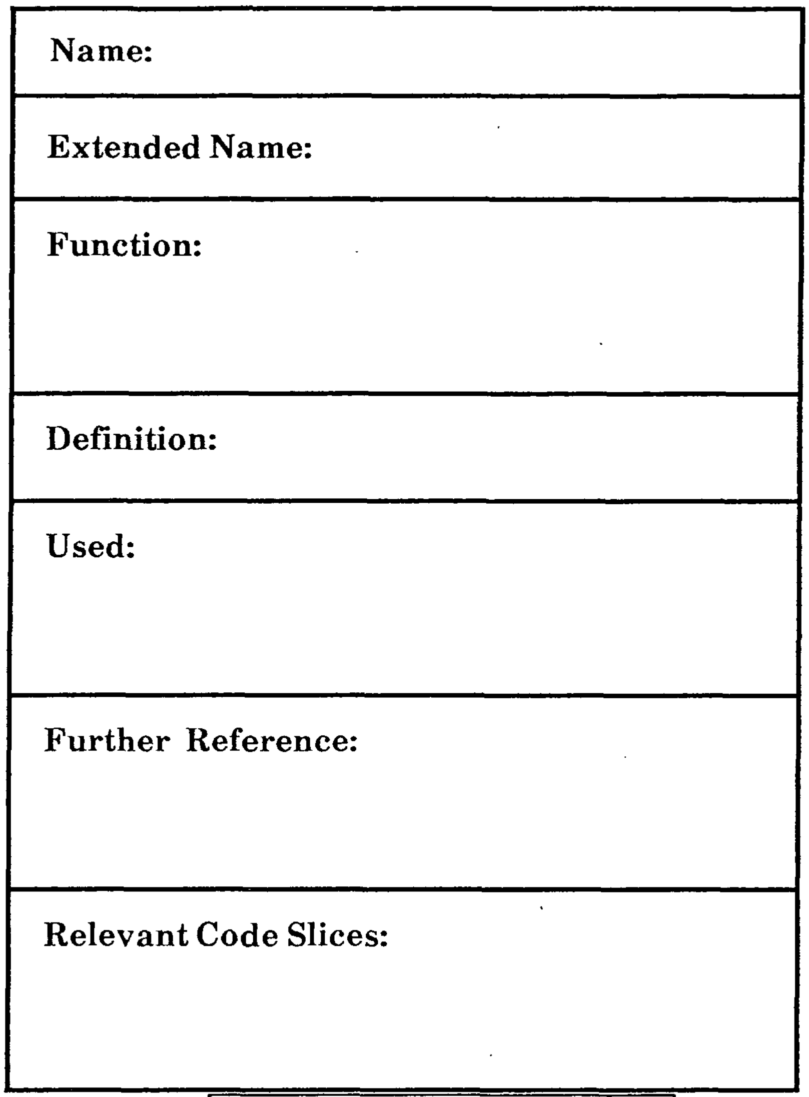
- To be automatically collected (from the use of SYSDOC):
  - Amount of analysis of the system
  - Amount of analysis of data names and functions within a system
  - Amount of documentation being produced for a system

As can be seen, data for all the applications being maintained on a site will be collected. The data base will hold data at two levels, site level, giving an overview of all the systems, and systems level, giving an overview of the work being carried out in a specific system. The currently collected data will be entered into the data base. Procedures must be in place to make sure that not too much time is spent on file for the redocumentation is the actual code and the cross reference listing, containing both the references for data names (or variables) and procedure names. The information is obtained by a simple search of the listing file to produce a file containing the source code and a second file containing the cross reference information.

7.5.2. Document generation

To convey information to the user, a standard form for each data name and function is created. An example is given in Figure 7. Using a Windows-based database (Omnis 5), information can be stored on each variable and function that is present within the cross reference listing. The database will be created when a COBOL file is viewed for the first time. The name of the variable (data name or function name) will be entered in the "name" field, the line of code in which the variable is declared is made will appear in the "definition" field, and the lines of code where the variable is used will appear in the "used" field. The user when viewing the file will be able to automatically access the relevant documentation by means of a hypertext link (see Section 7.5.3).

Information can then be entered if required in the remaining fields. The "extended name" field is present for the user to enter a more meaningful data or function name, the "function" field is for information relating to the use, or non obvious function, of the variable. The "further reference" field provides links to other information that may be on-line or in external text, while the "relevant code slices" field is an area in which the user can copy information obtained from slicing the code (see Section 7.5.4).



Name:
Extended Name:
Function:
Definition:
Used:
Further Reference:
Relevant Code Slices:

Figure 7
Blank Documentation Form

When the file is viewed for a second or latter time, the information previously entered is retained, though new variables may now be present, in which case an entry in the database will be made for these. The fields that are automatically filled in by the system will also be updated if the information has changed with relation to any of the information previously stored. Only the fields entered by the user will not be updated.

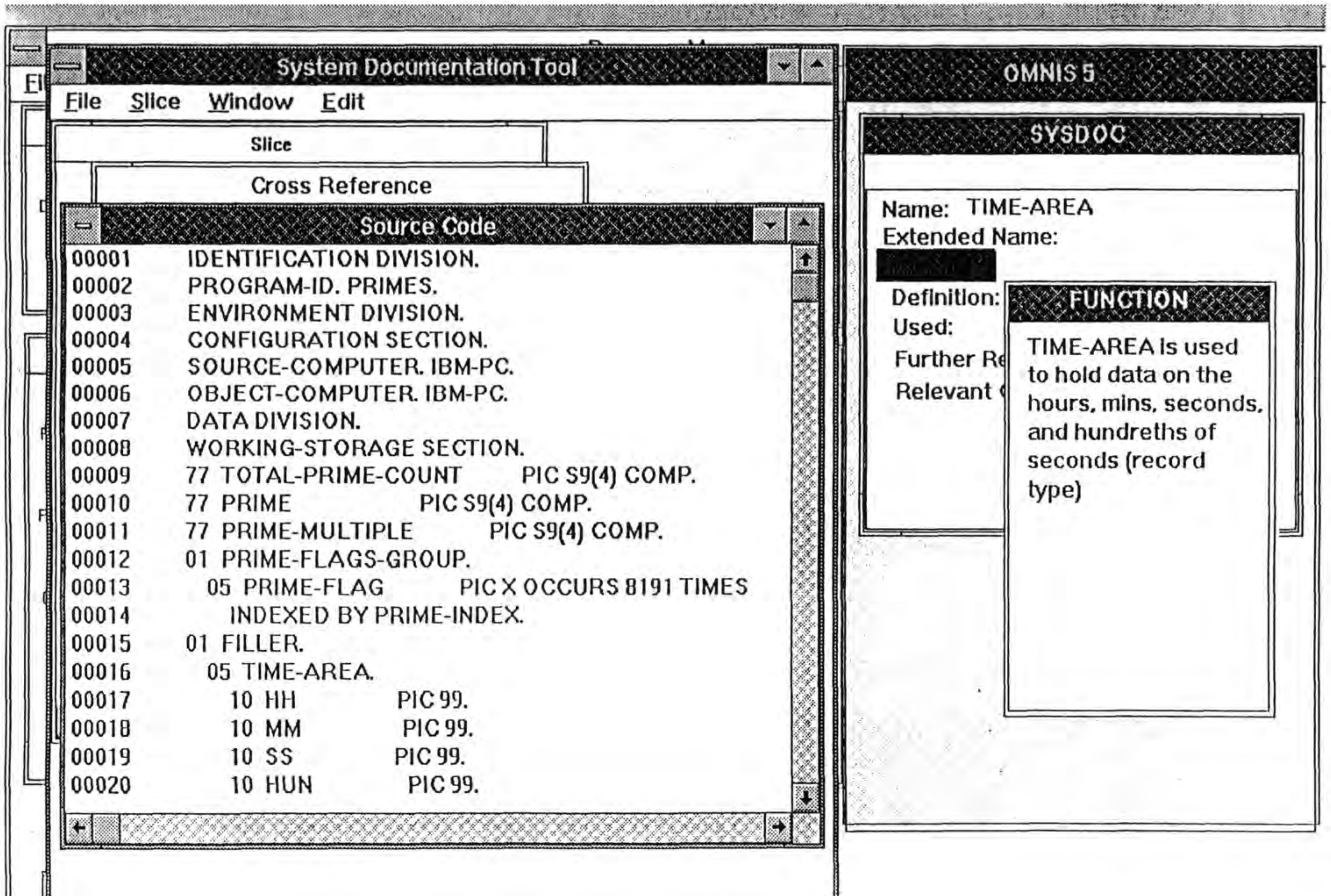
7.5.3. Hypertext Initialisation

Within the source code file, all occurrences of the data names and function names within the source code are marked as links to the cross reference database. The cross reference file already contains the information on the line number of further occurrences of all the data and function names. Nodes containing the documentation for each entry in the cross reference file already exist and are managed by the Omnis 5 database, and thus a specific link from all the references in the cross reference file to all the corresponding entries in the database is not required.

7.5.4. Sysdoc User Facilities

As stated earlier, the user interface has been developed using Microsoft Windows and thus most commands to the system are from menus using a mouse. The menus available are File, Slice, Window and Edit. The prototype version initially displays three windows containing, the COBOL source code, cross reference listing, and documentation. As stated earlier the documentation is kept in forms stored in an Omnis 5 database. To facilitate this, two applications are executing at once, one controlling the Omnis 5 database, and the second controlling the remaining windows and the user interface. This is possible by using the dynamic data exchange facilities within Windows 3.0 and Omnis 5.

An example of the user interface is shown in Figure 8 where the COBOL source code is in the front window. While viewing the code the user can activate the hypertext facilities either to scan to the next occurrence of a data or function name within the code, or to bring up the appropriate documentation form. This is stored



```
IDENTIFICATION DIVISION.
PROGRAM-ID. PRIMES.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-PC.
OBJECT-COMPUTER. IBM-PC.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  TOTAL-PRIME-COUNT        PIC S9(4) COMP.
77  PRIME                    PIC S9(4) COMP.
77  PRIME-MULTIPLE           PIC S9(4) COMP.
01  PRIME-FLAGS-GROUP.
    05  PRIME-FLAG           PIC X OCCURS 8191 TIMES
        INDEXED BY PRIME-INDEX.
01  FILLER.
    05  TIME-AREA.
        10  HH               PIC 99.
        10  MM               PIC 99.
        10  SS               PIC 99.
        10  HUN              PIC 99.
```

Name: TIME-AREA
Extended Name:
Definition:
Used:
Further Re
Relevant

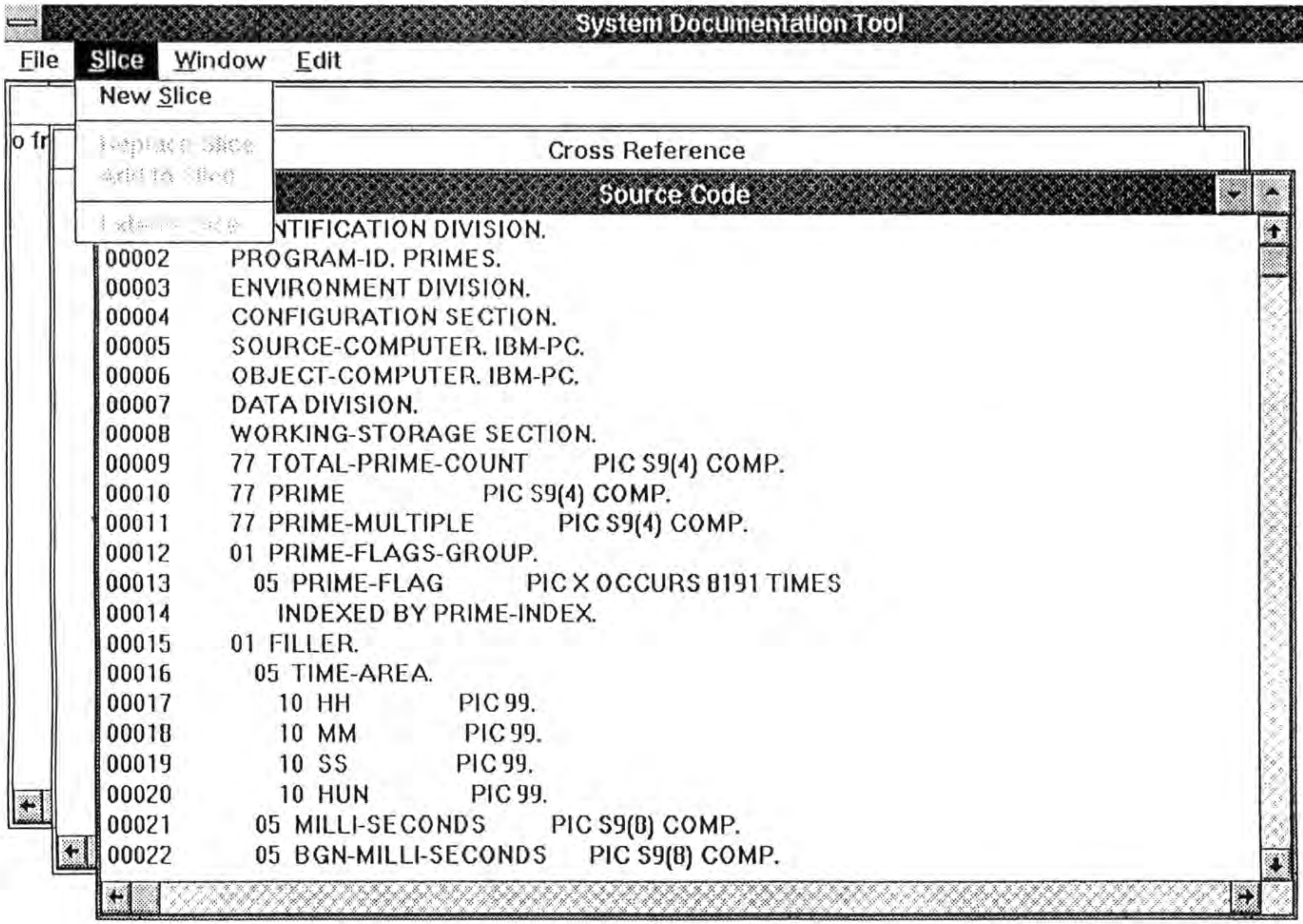
TIME-AREA is used to hold data on the hours, mins, seconds, and hundreths of seconds (record type)

Figure 8 - SYSDOC Windows

as a hypertext-style node within the database, to be displayed in the documentation window. It is then possible to write any relevant text in the appropriate fields of the documentation form, which will be saved at the end of the viewing session. Cut and paste options will be available to help speed up this documentation; thus the user will be able to copy text from any window into the documentation window with simple mouse actions. Full control over the sizing, and positioning of the windows are also available to the user.

The user has the option to slice the source code by choosing the slice menu (see Figure 9). *Slicing* involves selecting lines of the source code that fulfil particular criteria. For example these criteria may be all lines that contain a GOTO, or a particular data name. Once a slice has been performed, the system documentation tool will open a fourth window that shows only the lines of code selected by the previously entered search criteria. The search can be on either COBOL reserved words (PERFORM, IF, GOTO, CALL, or all of them), or on a typed variable. If the user types a string to be searched upon, the tool can look for occurrences of the string as a variable name, or as part of a variable name, thus increasing the system's flexibility.

Appendix A.1 shows the contents of a small COBOL program which can been loaded into the source code window. To carry out a slice on the data name PRIME, the user would choose the slice menu type PRIME in the "Variable Box" (see Figure 10) and press the Data Name button. The results output to the slice window are shown in Appendix A.2. The slice file contains the lines in which the variable is present and the number of lines between each occurrence. Appendix A.3 displays the output from the slice file if the user again picked the data name PRIME, but requested the system to search for the variable as part of a word as well as whole words by pressing the Inclusive button instead of the Data Name Button. Finally Appendix A.4 shows an example of the slice file after the COBOL



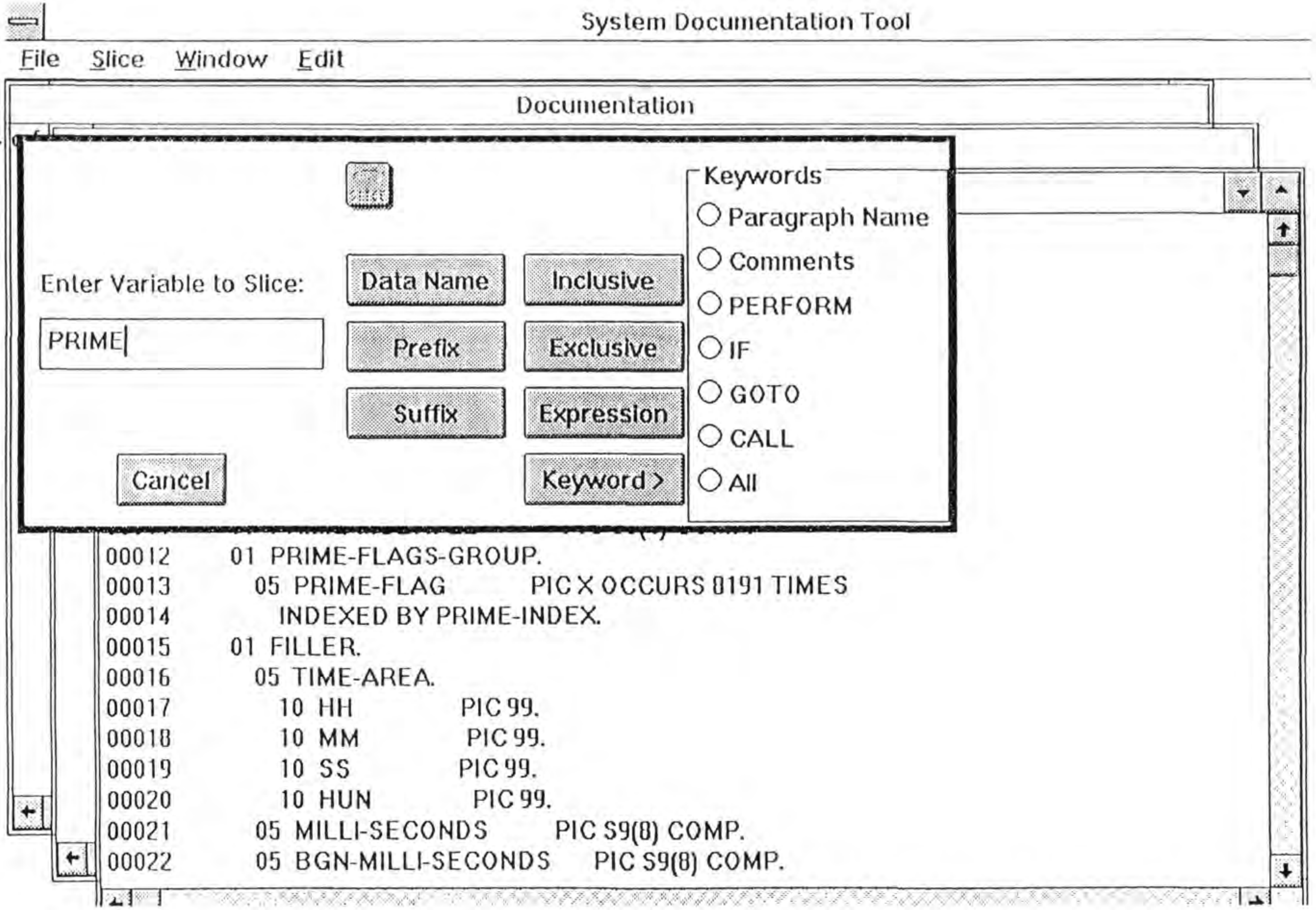
```
        NTIFICATION DIVISION.
00002   PROGRAM-ID. PRIMES.
00003   ENVIRONMENT DIVISION.
00004   CONFIGURATION SECTION.
00005   SOURCE-COMPUTER. IBM-PC.
00006   OBJECT-COMPUTER. IBM-PC.
00007   DATA DIVISION.
00008   WORKING-STORAGE SECTION.
00009   77 TOTAL-PRIME-COUNT      PIC S9(4) COMP.
00010   77 PRIME                  PIC S9(4) COMP.
00011   77 PRIME-MULTIPLE         PIC S9(4) COMP.
00012   01 PRIME-FLAGS-GROUP.
00013      05 PRIME-FLAG          PIC X OCCURS 8191 TIMES
00014         INDEXED BY PRIME-INDEX.
00015   01 FILLER.
00016      05 TIME-AREA.
00017         10 HH               PIC 99.
00018         10 MM               PIC 99.
00019         10 SS               PIC 99.
00020         10 HUN              PIC 99.
00021      05 MILLI-SECONDS       PIC S9(8) COMP.
00022      05 BGN-MILLI-SECONDS   PIC S9(8) COMP.
```

Figure 9 - A Windows Menu Showing Slice Options

reserved word PERFORM has been selected and the Perform button pressed followed by that of the Keyword button. The buttons Suffix, Prefix, and Exclusive work in a similar way to Inclusive, but only produce the lines containing the suffix, prefix etc. of the variable that has been input.

After a slice has been completed, the user may choose to replace the present slice, or increase it in one of two ways. The first is to select another variable to be used in the slice, in which case the lines in which the previously selected variable and the presently selected variable are displayed in the slice window. Appendix A.5 shows a slice where a new slice was created using the variable name PRIME, followed by the user choosing the "add to slice" option in the slice menu, typing in MILLI-SECONDS and pressing the Data Name button. Secondly the user can select to add additional lines to the slice whereupon the system adds the number of lines selected to the slice after the given line number. First the "extend slice" option is selected in the slice menu. Appendix A.6 shows an example of where PRIME has been entered as the variable, the Data Name button has been selected and finally the user has chosen the "extend slice" option and requested the slice to be extended by five lines forward from line 43.

By using the slicing option, the user can build up a picture of a certain area of the code, without being distracted by information which does not concern them. In short, this option provides information hiding for the analysis stage of software maintenance. The use of slicing in combination within hypertext-style links to a documentation database is considered to be one of the most useful facilities of the complete documentation system.



Enter Variable to Slice: PRIME

```
00012     01  PRIME-FLAGS-GROUP.
00013         05  PRIME-FLAG          PIC X OCCURS 8191 TIMES
00014             INDEXED BY PRIME-INDEX.
00015     01  FILLER.
00016         05  TIME-AREA.
00017             10  HH          PIC 99.
00018             10  MM          PIC 99.
00019             10  SS          PIC 99.
00020             10  HUN         PIC 99.
00021         05  MILLI-SECONDS       PIC S9(8) COMP.
00022         05  BGN-MILLI-SECONDS       PIC S9(8) COMP.
```

Figure 10 - A Window's Dialog Box Used for Entering Slice Details

7.6 Summary

This chapter described the design of four tools based on the five documentation types described in Chapter 6. Both the Site Information based on Hypertext tool, and the System Documentation tool have been implemented. A prototype for the overview documenter has also been implemented. The design of the prototype management documenter has also been described.

Chapter eight

Conclusions

8.1 Introduction

This chapter gives a brief evaluation of the Redocumentation Aid for the Maintenance of Software (RAMS). The benefits and drawbacks of using the system are discussed, along with work which would enhance future systems.

8.2 Evaluation and Fulfilment of Requirements

The research described in this thesis has met the initial objectives outlined in Chapter 1. Chapter 4 gives a detailed outline of the requirements for a documentation system to be used within a third party software maintenance environment. When evaluated against these requirements, the Redocumentation Aid for the Maintenance of Software has fulfilled the the aims detailed in this section. These include:

- It is possible to make the additions and corrections to support the four documentation activities.

- Quality assurance is maintained with the use of the management documenter.

- A standard document layout is enforced by the SYSDOC and SIBOH systems.

- Casual updating is maintained throughout the tools with the user able to document as required.

- A hierarchy of documents, as described in Section 6.2, has been created.

- The SYSDOC system allows the user to view the code and documentation in parallel by using the hypertext-style links.

- This system incorporates the facility to include existing documentation, and selected viewing of areas of the code through the use of the simple slicing facility.

- By the use of the Microsoft Windows environment, a common interface to the tools is provided for the user.

The RAMS system is being developed for use by AGS Information Services Ltd., and the requirements for the tool have been developed from their needs. This has the disadvantage of occasionally restricting the scope of the research. The use of certain technology was restricted. Also the research was more directed than would often be the case. However it enabled approval for the requirements, listed in Chapter 4, to be obtained at a relatively early stage of the research.

To date, one of the original site information documents has been transferred to the SIBOH tool. This conversion was achieved using cut and paste facilities on the Xerox 6085. The use of hypertext has removed the duplication in the document and it is now more maintainable. Initial response from users of the system find obtaining information on a subject less time consuming than with the previous traditional text. This is partly due to the use of links between related topics.

The System Documentation tool is currently undergoing preliminary on-site testing. The slice option has been frequently used and regarded as a considerable help in code analysis. This facility mimics the way AGS programmers work when trying to understand the code of a new system. Feedback from AGS has meant the addition of a number of pre-defined slicing options such as "slice exclusively" and "keyword slice".

The RAMS prototype has met most of its main requirements and is in the process of being completed before full on-site trials begin. Section 8.4 details those parts of the system that have not been implemented. The success of RAMS cannot be judged yet but the enthusiasm of the initial feedback has helped greatly in showing the viability of such a system.

8.3 Benefits and Drawback of the Approach

The major benefits of the approach adopted in the development of the RAMS system can be sumarized as follows:

- **Increased speed of code analysis**
  With the use of slicing and hypertext links between code and documentation, the speed of obtaining information on a particular area of a system is increased.

- **Document hierarchy for third party maintenance**
  By adopting a hierarchical approach the third party maintainer will not be

swamped or starved of information. A hierarchy is more essential in third party maintenance due to the greater range of information, from details on maintenance sites to complex information on code usage.

- **Incremental documentation**

  It is possible, and encouraged, for the user to only document the parts of the system they are currently working on and thus reduce expended documentation effort. The use of a standard database form, relating to a specific area, will also help target the documentation effort to the problem area.

- **Simple user interface**

  The interface used for the tools is based on the WIMP style of interface; thus complex sequences of commands are not required. The Microsoft Windows interface used in SYSDOC is rapidly becoming a standard user interface and, therefore, an ideal operational environment

- **Slicing**

  The slicing of the COBOL to show only small sections of the code helps the user to concentrate on a single area without distraction from other parts of the implementation. This has helped with code analysis by focusing the user's thoughts.

- **Saving of knowledge during analysis**

  By being able to save any slices of the documentation created during the analysis of the code, the information, which is often regarded as highly valuable, can be viewed by future analysts.

A number of drawbacks have also been identified. Most of these are regarded as minor in relation to the advantages gained using the RAMS system. The drawbacks are summarised below:

- **Downloading of code to P.C.**

  For the use of the SYSDOC system, the source code has to be downloaded to a P.C. This has had to be the case as a great number of mainframe systems do not provide the windowing capabilities required and the mainframe is unsuitable for such applications.

- **Multiple environments**

  The four systems available within RAMS run on two types of hardware: SYSDOC and the Management Documenter on an IBM P.C., and SIBOH and the Overview Documenter on the Xerox 6085. This is due to the user requirements. The data however can easily be transferred due to a P.C. interface present on the 6085.

- **Initial increase in workload**

  To use the SIBOH an increase in the time taken to enter the document initially has been noted. This is offset by the increase in readability and speed of analysis. A great deal of work has to be done to set up each overview for use in the Overview Documenter. Therefore, until this can be automated (see Section 8.4), only a system which is requiring a large amount of analysis should be entered into the Overview Documenter

8.4 Further Research and Development

The research carried out has developed a complete system for the redocumentation in a third party software maintenance environment. The prototype, already being preliminarily tested on-site, has opened up many other areas of research which would enhance the present system. These areas, which time has not permitted the exploration of, are described below.

The research has shown the requirements for the Overview Documenter and the Management Documenter. In the case of the former, a method for the automation of the setting up of an overview of a system to reduce the amount of time required to complete this action is needed. The author recognizes the work required to extract information from the code and represent it in this form. Such tools are already available, though not based on hypertext. These systems, such as Battle Map and EDSA, are reviewed by Simon[52] and are expensive. It would be advisable, instead of completely writing one of these tools, to take the output of one of the currently available systems and enter this into a hypertext system.

The Management Documenter needs to be implemented. Considerable work has already been done on the collection and display of data by Cooper[20], and it would be ideal to combine the system proposed by him into the RAMS system. Also a system has to be set up to allow for easy viewing of the maintenance standards, change control procedures, and other in-house standards detailed in Section 6.4.

One of the requirements of the redocumentation aid for the maintenance of software was to provide configuration control of the documents and of the source code, and this should appear in future implementations. Again complete configuration control is not trivial and has been the focus of a great deal of research by Kenning[39] . It would be ideal to use such a system in combination with the documentation aid as opposed to re-inventing the wheel.

At present the RAMS system has been developed for use on COBOL source code. Though within large institutions COBOL is still widely used, more systems are being developed with languages such as C. Therefore for maximum use of the RAMS system, it must become generic. This would only require the change of the module containing the code which initially formats the COBOL listing file, and the module which creates the slice information (both of these modules are present within the SYSDOC system). This enhancement would also affect any automation of the Overview Documenter.

Appendix A

Example Test Files and Output from SYSDOC Slice Operations

A.1 Primes COBOL Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PRIMES.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-PC.
OBJECT-COMPUTER. IBM-PC.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  TOTAL-PRIME-COUNT      PIC S9(4) COMP.
77  PRIME                  PIC S9(4) COMP.
77  PRIME-MULTIPLE         PIC S9(4) COMP.
01  PRIME-FLAGS-GROUP.
    05  PRIME-FLAG         PIC X OCCURS 8191 TIMES
        INDEXED BY PRIME-INDEX.
01  FILLER.
    05  TIME-AREA.
        10  HH             PIC 99.
        10  MM             PIC 99.
        10  SS             PIC 99.
        10  HUN            PIC 99.
```

```
00021        05  MILLI-SECONDS        PIC S9(8) COMP.
00022        05  BGN-MILLI-SECONDS      PIC S9(8) COMP.
00023        05  DISPLAY-MILLI-SECONDS  PIC Z(8).
00024      PROCEDURE DIVISION.
00025      PRIME-COUNT-ROUTINE.
00026        PERFORM CALC-MILLI-SECONDS
00027        MOVE MILLI-SECONDS TO BGN-MILLI-SECONDS.
00028        PERFORM CALC-PRIMES 10 TIMES
00029        DISPLAY 'COUNT:' TOTAL-PRIME-COUNT
00030        PERFORM CALC-MILLI-SECONDS.
00031        SUBTRACT BGN-MILLI-SECONDS FROM MILLI-SECONDS
00032          GIVING DISPLAY-MILLI-SECONDS.
00033        DISPLAY 'Elapsed time was' DISPLAY-MILLI-SECONDS
00034          ' milliseconds'
00035        STOP RUN.
00036      COUNT-PRIMES.
00037        SEARCH PRIME-FLAG VARYING PRIME-INDEX
00038          WHEN PRIME-FLAG (PRIME-INDEX) IS NOT EQUAL TO
ZERO
00039            ADD 1 TO TOTAL-PRIME-COUNT
00040            SET PRIME-MULTIPLE TO PRIME-INDEX
00041            ADD PRIME-MULTIPLE PRIME-MULTIPLE 1 GIVING
PRIME
00042            ADD PRIME TO PRIME-MULTIPLE
00043            SET PRIME-INDEX UP BY 1
00044            PERFORM I-C UNTIL PRIME-MULTIPLE IS GREATER
THAN 8191
00045            GO TO COUNT-PRIMES.
00046      CALC-MILLI-SECONDS.
```

```
00047         MULTIPLY HH BY 60 GIVING MILLI-SECONDS.
00048         ADD MM TO MILLI-SECONDS.
00049         MULTIPLY 60 BY MILLI-SECONDS.
00050         ADD SS TO MILLI-SECONDS.
00051         MULTIPLY 100 BY MILLI-SECONDS.
00052         ADD HUN TO MILLI-SECONDS.
00053         MULTIPLY 10 BY MILLI-SECONDS.
00054      CALC-PRIMES.
00055          MOVE ALL '1' TO PRIME-FLAGS-GROUP.
00056          MOVE ZERO TO TOTAL-PRIME-COUNT.
00057          SET PRIME-INDEX TO 1.
00058          PERFORM COUNT-PRIMES.
00059      I-C.
00060          MOVE ZERO TO PRIME-FLAG (PRIME-MULTIPLE).
00061          ADD PRIME TO PRIME-MULTIPLE.
```

A.2 Slice on PRIME (Data Name Option)

```
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = =  25 lines missing
00026        PERFORM CALC-MILLI-SECONDS
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = =  1 line  missing
00028        PERFORM CALC-PRIMES 10 TIMES
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = =  1 line  missing
00030        PERFORM CALC-MILLI-SECONDS.
```

```
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = =   13 lines missing
00044          PERFORM I-C UNTIL PRIME-MULTIPLE IS GREATER
THAN 8191
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = =   13 lines missing
00058       PERFORM COUNT-PRIMES.
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = =   3 lines missing
```

A.3 Slice on PRIME (Inclusive Option)

```
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = =   1 line  missing
00002     PROGRAM-ID. PRIMES.
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = =   6 lines missing
00009     77  TOTAL-PRIME-COUNT       PIC S9(4) COMP.
00010     77  PRIME               PIC S9(4) COMP.
00011     77  PRIME-MULTIPLE         PIC S9(4) COMP.
00012     01  PRIME-FLAGS-GROUP.
00013         05  PRIME-FLAG         PIC X OCCURS 8191 TIMES
00014             INDEXED BY PRIME-INDEX.
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = =   10 lines missing
00025     PRIME-COUNT-ROUTINE.
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = =   2 lines missing
```

```
00028       PERFORM CALC-PRIMES 10 TIMES
00029       DISPLAY 'COUNT:' TOTAL-PRIME-COUNT
=====================================
=====================  6 lines missing
00036     COUNT-PRIMES.
00037       SEARCH PRIME-FLAG VARYING PRIME-INDEX
00038         WHEN PRIME-FLAG (PRIME-INDEX) IS NOT EQUAL TO
ZERO
00039           ADD 1 TO TOTAL-PRIME-COUNT
00040           SET PRIME-MULTIPLE TO PRIME-INDEX
00041           ADD PRIME-MULTIPLE PRIME-MULTIPLE 1 GIVING
PRIME
00042           ADD PRIME TO PRIME-MULTIPLE
00043           SET PRIME-INDEX UP BY 1
00044           PERFORM I-C UNTIL PRIME-MULTIPLE IS GREATER
THAN 8191
00045           GO TO COUNT-PRIMES.
=====================================
=====================  8 lines missing
00054     CALC-PRIMES.
00055        MOVE ALL '1' TO PRIME-FLAGS-GROUP.
00056        MOVE ZERO TO TOTAL-PRIME-COUNT.
00057        SET PRIME-INDEX TO 1.
00058        PERFORM COUNT-PRIMES.
=====================================
=====================  1 line missing
00060        MOVE ZERO TO PRIME-FLAG (PRIME-MULTIPLE).
00061        ADD PRIME TO PRIME-MULTIPLE.
```

A.4 Slice on PERFORM

```
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = =  9 lines missing
00010    77  PRIME            PIC S9(4) COMP.
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = =  30 lines missing
00041            ADD PRIME-MULTIPLE PRIME-MULTIPLE 1 GIVING
PRIME
00042            ADD PRIME TO PRIME-MULTIPLE
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = =  18 lines missing
00061          ADD PRIME TO PRIME-MULTIPLE.
```

A.5 Additional Slice

```
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = =  9 lines missing
00010    77  PRIME            PIC S9(4) COMP.
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = =  10 lines missing
00021       05  MILLI-SECONDS      PIC S9(8) COMP.
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = =  5 lines missing
00027       MOVE MILLI-SECONDS TO BGN-MILLI-SECONDS.
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = = = = = = = = = = = = = = = = = = = = = =  3 lines missing
```

```
00031        SUBTRACT BGN-MILLI-SECONDS FROM MILLI-SECONDS
====================================
======================  9 lines missing
00041            ADD PRIME-MULTIPLE PRIME-MULTIPLE 1 GIVING
PRIME
00042            ADD PRIME TO PRIME-MULTIPLE
====================================
======================  4 lines missing
00047        MULTIPLY HH BY 60 GIVING MILLI-SECONDS.
00048        ADD MM TO MILLI-SECONDS.
00049        MULTIPLY 60 BY MILLI-SECONDS.
00050        ADD SS TO MILLI-SECONDS.
00051        MULTIPLY 100 BY MILLI-SECONDS.
00052        ADD HUN TO MILLI-SECONDS.
00053        MULTIPLY 10 BY MILLI-SECONDS.
====================================
======================  7 lines missing
00061          ADD PRIME TO PRIME-MULTIPLE.
```

A.6 Extended Slice

```
====================================
======================  9 lines missing
00010       77  PRIME              PIC S9(4) COMP.
====================================
======================  30 lines missing
00041            ADD PRIME-MULTIPLE PRIME-MULTIPLE 1 GIVING
PRIME
```

```
00042          ADD PRIME TO PRIME-MULTIPLE
00043          SET PRIME-INDEX UP BY 1
00044          PERFORM I-C UNTIL PRIME-MULTIPLE IS GREATER
THAN 8191
00045          GO TO COUNT-PRIMES.
00046      CALC-MILLI-SECONDS.
00047        MULTIPLY HH BY 60 GIVING MILLI-SECONDS.
00048        ADD MM TO MILLI-SECONDS.
=====================================
====================== 12 lines missing
00061         ADD PRIME TO PRIME-MULTIPLE.
```

Bibliography

[1] AGS Information Service Ltd., "AGS Team Maintenance Standards", Release 1.0, December 1988

[2] Anderson, R.E., "Modular Documentation: a Software Development Tool", Proceedings of the National Computer Conference, 1981, pp 401 - 405

[3] Apple Computers Inc, "HyperCard User Guide", 1989

[4] Arthur, J.D., and Stevens, K.T., "Assessing the adequacy of documentation through document quality indicators", Conference on Software Maintenance - 1989 proceedings, IEEE Comp. Sci Press, Washington, pp 40 - 49

[5] Bennett, K.H., Cornelius, B.J., Munro, M., and Robson, D.J., "Software Maintenance", Software Engineering Reference Book, Edited by McDermid, 1991, pp 20.1 - 20.18

[6] Boehm, B.W., "Software and its Impact: a Quantative Assessment", Datamation, May 1973, pp 48-59

[7] Boehm, B.W., "Software Engineering Economics", Prentice Hall, 1981

[8] Bolter, J.D., and Joyce, M., "Hypertext and Creative Writing", Proceedings of Hypertext 1987, pp 41 - 50

[9] Brown, P.J., "Interactive Documentation", Software: Practice and Experience, March 1986, pp 291 - 299

[10] Brown, P.J., "Turning Ideas into Products: The Guide System", Proceedings of Hypertext 1987, pp 33 - 40

[11] Bush, E., "Reverse Engineering: What and Why", Proceedings of the Forth European Software Maintenance Workshop, Centre for Software Maintenance, Durham - September 1990

[12] Bush, V., "As We May Think", Atlantic Monthly, 176, 1, July 1945, pp 101 - 108

[13] Calliss, F.W., "Problems with Automatic Restucturers", ACM SIGPLAN Notices, 23, 1987, pp 13 - 21

[14] Calliss, F.W, "Inter-Module Code Analysis Techniques for Software Maintenance", PhD. Thesis, University of Durham, 1989

[15] Chang, S., and McGowan, C., "Full-text Retrival in Software Maintenance", Proceedings of COMSAC 87, 1987, pp 53 - 57

[16] Chapin, N., "Software maintenance: a different view", AFIPS Conf. Proc. 54 National Computer Conference, 1985, pp 509 - 513

[17] Chen, Y., and Ramamoorthy, C.V., "The C Information Abstractor", COMP86, October 1986, pp 291 - 298

[18] Chikofsky, E.J., and Cross, J.H., "Reverse Engineering and Design Recovery: A Taxonomy", IEEE Software, vol 7, no1, Jan 1990, pp 13 - 18

[19] Conklin, J., "Hypertext: An Introduction and Survey", IEEE Computer, September 1987, pp 17 - 41

[20] Cooper, S.D., and Munro, M., "Software change information for maintenance management", Conference on Software Maintenance - 1989 proceedings, IEEE Comp. Sci Press, Washington, pp 279 - 287

[21] Downs, E., Clare, P., and Coe, I., "Structured Systems Analysis and Design Method", Prentice Hall, 1988

[22] Englebert, D.C., and English, W.K., "A Research Centre for Augmenting Human Interlect", Proceddings of the 1968 Fall Computer Conference, Montvale, N.J., 1968, AFIPS Press, pp 395 - 410

[23] Fletton, N.T., "Documentation for software maintenance and the documentation of existing systems", M.Sc. Thesis, University of Durham, 1988

[24] Fletton, N.T., and Munro, M., "Redocumenting software systems using Hypertext technology", Conference on Software Maintenance - 1988 proceedings, IEEE Comp. Sci Press, Washington, pp 54 - 59

[25] Foster, J., and Munro, M., "A documentation method based on cross-referencing", Proceedings of the Conference on Software Maintenance, Austin, Texas - September 1987, IEEE Comp. Sci. Press, Washington

[26] Freeman, R.M., and Munro, M., "Xebra - Xerox Based Redocumentation Aid", Proceedings of the Software Maintenance Association Conference. 1990, Vancouver, pp 4.35 - 4.47

[27] Garg, P.K., and Scacchi, W., "A Hypertext System to Manage Software Life Cycle Documents", Proceedings of 21st Annual Hawaii International Conference on System Sciences, 1988, pp 337 - 346

[28] Gibson, V.R., and Senn, J.A., "System Structure and Software Maintenance Performance", Communications of the ACM, Vol. 32, 3, March1989, pp 347 - 358

[29] Guillemette, R.A., "Application Software Documentation: a Reader Measure", PhD. Thesis, University of Houston, May 1986

[30] Halasz, F.G., "Reflections on NoteCards, Seven Issues for the Next Generation of Hypermedia Systems", Communications of the ACM, Volume 31, Number 7, July 1988, pp 836 - 852

[31] Harband, J., "SEELA: Maintenance and Documentation by Reverse Engineering", Conference on Software Maintenance - 1990 proceedings, IEEE Comp. Sci Press, Washington, pp 466 - 466

[32] Horowitz, E., and Williamson, R.C., "SODOS: A Software Documentation Support Environment - Its Definition", IEEE Transactions on Software Engineering, Vol SE-12, No. 8, August 1986, pp 849 - 859

[33] Huffman, J.E., and Burgess, C.G., "Partially automated in-line documentation (PAID): Design and implementation of a software maintenance tool", Conference on Software Maintenance - 1988 proceedings, IEEE Comp. Sci Press, Washington, pp 60 - 65

[34] IEEE 830 Software Requirement Document

[35] IEEE Software engineering standards, 1984, pp 31-32

[36] James, G., "Document Databases", Van Nostrand Reinhold, New York, 1985

[37] Jandrasics, G., "Static Analysis of Commercial Programs with the SOFTDOC system", Technical Reports, SES Software Engineering Services, Pappelstr. 6, D-8014 Munich, Germany, 1981

[38] Katsoulakos, T., "Overview of the ESPRIT project REDO", Proceedings of the Third European Software Maintenance Workshop, Centre for Software Maintenance, Durham - September 1989

[39] Kenning, R.J., and Munro, M., "Towards Configuring Operational Systems", Software Tools Notes, Software Tools Conference, Wembly, June 1990

[40] Lientz, B.P., and Swanson, E.B., "Software maintenance: A user/management tug-of-war", Data Management, April 1979

[41] Lientz, B.P., and Swanson,E.B., "Software Maintenance Management", 1980, Addison Wesley

[42] Marshall, C.C., "Exploring Representation Problems using Hypertext", Proceedings of Hypertext 1987, pp 253 -268

[43] Martin, J., and McClue, C., "Software Maintenance: The Problem and its Solutions", Prentice Hall, 1983

[44] McGowan, S., "FORTUNE - an ipse Documentation Tool", Technical Report, CAP (UK) Ltd., 1987

[45] Miller, J.C., Strauss, B.M., "Implementations of Automatic Restructuring of COBOL", ACM SIGPLAN Notices, 22, 1987, pp 41 - 49

[46] Morissey, J.H. and Wu, L.S.Y., "Software engineering: An economic perspective", Proceedings of the 4th international conference on software engineering, Munich, September 1979, pp 17-19

[47] Nelson, T.D., "Getting it Out of Our System", Information Retrieval: A Critical Review, G. Schechter, ed. Thompson Books, Washington, 1967, pp 191 - 210

[48] Nelson, T.H., "Replacing the Printed Word: A Complete Literary System", IFIP Proceedings, October 1980, pp 1013 - 1023

[49] Pawson, S., "Maintenance in a commercial D.P. environment", Proceedings of the First Software Maintenance Workshop, Centre for Software Maintenance, Durham - September 1987

[50] Petzold, C., "Programming Windows : the Microsoft Guide to Writing Applications for Windows 3", Microsoft Press, Redmond, Washington, 1990

[51] Rittel, H., and Webber, M., "Dilemmas in a General Theory of Planning", Policy Sciences, Vol 4, 1973

[52] Simon, A., "Requirements for a Software Maintenance Support Environment", MSc Thesis, University of Durham, 1991, pp 147 - 149

[53] Singleton, M.E., "Automating Code and Documentation Management", Prentice Hall, 1987

[54] Smith, J.B., and Weiss, S.F., "Hypertext", Communications of the ACM, Vol 31, No. 7, July 1988, pp 816 - 819

[55] Sneed, H., and Jandrasics, G., "Software Recycling", Conference on Software Maintenance - 1987 proceedings, IEEE Comp. Sci Press, Washington, pp 82 - 90

[56] Sneed, H., "Software Renewal: a Case Study", IEEE Software 1(3), 1988, pp 56 - 63

[57] Sneed, H., "Ecconomics of Software Re-engineering", Proceedings of the Forth European Software Maintenance Workshop, Centre for Software Maintenance, Durham - September 1990

[58] Sommerville, I., Welland, R., Bennett, I., and Thomson, R., "SOFTLIB - a Documentation Management System", Software - Practice and Experience, Vol 16, No. 2, Febrary 1986, pp 131 - 143

[59] Swanson, E.B., "The dimension of Maintenance", Proceedings of the 2nd International Conference on Software Maintenance, October 1976, IEEE/ACM, pp 492 - 497

[60] Tang, R., "Third Party Software Maintenance", Proceedings of the Third European Software Maintenance Workshop, Centre for Software Maintenance, Durham - September 1989

[61] Trigg, R.H., and Irish, P.M., "Hypertext Habitats: Experience of Writers in NoteCards", Proceedings of Hypertext 1987, pp 89 - 108

[62] Turner, R.J., "Software Maintenance: Generating Front Ends for Cross Referencer Tools", M.Sc. Thesis, University of Durham, 1989

[63] U.S. Department of Commerce, "Guidlines for Documentation of Computer Programs and Automated Data Systems", Federal Information Processing Standards Publication 30, June 1984

[64] Walker, J.H., "Documnet Examiner: Delivery Interface for Hypertext Documents", Proceedings of Hypertext '87, University of North Carolina at Chapel Hill, 307 - 324

[65] Walker, J.H., "Supporting Document Development with Concordia", Proceedings of 21st Annual Hawaii International Conference on System Sciences, 1988, pp 355 - 364

[66] Ward, M., "Proving Program Refinements and Transformations", PhD. Thesis, University of Oxford, 1989

[67] Ward, M., Calliss, F.W., and Munro, M., "The Use of Transformations in the Maintainers Assistant", Conference on Software Maintenance - 1989 proceedings, IEEE Comp. Sci Press, Washington, pp 307 - 315

[68] Weiser, M. D., "Program Slices: Formal Psychological and Practical Investigations of an Automatic Program Abstraction Method", Ph.D. Thesis, University of Michigan, Ann Arbour, 1979.

[69] Xerox, "Xerox ViewCards: User Handbook", VP Series Reference Library, Version 2.0, 1989

[70] Yang, H., "How Does the "Maintainers Assistant" Start?", Centre for Software Maintenance, University of Durham, March 1991

[71] Zvegintzov, N., "Nanotrends", Datamation, August 1983, pp 106-116