# Inspection of Software Deliverables: an infoMap-based Methodology

Benoît Deslauriers

A Thesis
in
The Department
of
Computer Science

Presented in Partial Fulfilment of the Requirements for
the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

September 1991

ISBN 0-315-68749-5

# Abstract

# Inspection of Software Deliverables: an infoMap-based Methodology

Benoît Deslauriers
Computer Science Department
Concordia University
1991

Software quality is one of the biggest challenge of today's software engineering society. High quality in a software product is achieved through a continuous verification of the product during the development process. For over a decade, software inspections have been employed with some success as a verification technique in a number of large companies including IBM and AT&T.

This thesis proposes an inspection methodology based on Fagan's traditional approach augmented with the infoStructure concept, a set-oriented knowledge representation technique. While inspections has been conventionally applied to final products, i.e. source code, the proposed inspection process efficiently deals with all products of the software development life cycle.

The infoStructure allows the inspection of any software deliverable to be carried out systematically and hierarchically. Defect types and detection strategies are directly derived from the infoStructure notation. Better visualisation of defects is obtained through the use of the infoStructure elements.

The methodology is validated with two examples of deliverable issue from the waterfall model. The inspection of a Software Requirements Document (SRD) and a Software Code Document (SCD) are detailed. The inspection of remaining software deliverables is not shown as they represent intermediate products.

# Acknowledgement

I would like to thank my thesis supervisor, Dr. W.M. Jaworski, for his permission to use copyrighted infoSchemas in the Figures and Appendices of this thesis, and for his guidance throughout this research. The confidence he showed in me has been a strong motivation factor for the completion of this thesis. I am also thankful to Dr. V.S. Alagar and Spyros Kattou for providing me with pertinent Software Requirements Documents, and Thompson Cummings for his suggestive comments.

Table of Contents

Introduction 1
Chapter 1 The Software Development Process 3
1.1 Software Verification and Validation 4
1.2 Software Maintenance vs Software Upgrade 5
1.3 A New Software Development Process 7
Chapter 2 State of the Art of Software Inspections 9
2.1 The Traditional Inspection Method 9
2.2 Inspection Benefits 13
2.3 Inspection Drawbacks 15
2.4 The Two-Person Inspection Method 15
2.5 Comparing Inspections with Other Verification Methods 16
Chapter 3 The Proposed Inspection Method 18
3.1 Participants 19
3.2 Planning 21
3.3 Overview 21
3.4 Normalization 21
3.4.1 Normalization vs Preparation 22
3.5 Analysis 22
3.5.1 Analysis vs Paraphrasing 23
3.6 Rework 24
3.7 Follow-up 24
Chapter 4 The infoStructure 25
4.1 The History 25
4.2 The Notation 27
4.3 Characteristics 31
4.4 Training 32
Chapter 5 Requirements Inspection 33
5.1 SRD Normalization 34
5.1.1 Sets 34
5.1.2 Views 36
5.1.3 Results 37

5.2 SRD Analysis 37
5.2.1 Defect Classification 39
5.2.2 Defect Detection Strategies 44
Chapter 6 Code Inspection 48
6.1 SCD Normalization 49
6.1.1 Sets 49
6.1.2 Views 51
6.1.3 Results 52
6.2 SCD Analysis 52
6.2.1 Defect Classification 53
6.2.2 Defect Detection Strategies 56
Chapter 7 Results Analysis 59
7.1 Evaluating the Proposed Inspection Method 59
7.1.1 Participants 59
7.1.2 Normalization 60
7.1.3 Analysis 61
7.2 Discussion 63
Conclusion 65
References 67
Appendices
Appendix A The infoMap for the Waterfall Model
Appendix B The infoMap for the infoStructure Notation
Appendix C Software Requirement Document for HyperDoc
Appendix D The infoMap for the HyperDoc SRD
Appendix E The convert Program SCD [51]
Appendix F The infoMap for the convert Program SCD
Appendix G Software Requirement Document for New Editor (NED)
Appendix H The infoMap for the NED SRD

## List of Figures


Fig. 1.1 The Waterfall Model of the Software Development Process [106]. 4

Fig. 1.2 The Software Maintenance Process. 7

Fig. 2.1 The Traditional Inspection Method. 10

Fig. 2.2 Fagan's Defect Types [89]. 11

Fig. 2.3 Myers' Defect Types [75]. 11

Fig. 2.4 Inspection Progress Rates. 12

Fig. 2.5 An Example of Information Collected Throughout the Inspection Process [79]. 13

Fig. 2.6 Defects Detection Rates. 14

Fig. 3.1 The infoMap for the Proposed Inspection Method. 20

Fig. 3.2 Defect Detection Strategies for the Structure Level. 23

Fig. 3.3 Defect Detection Strategies for Both Analysis Levels. 23

Fig. 4.1 The infoSchema for the Job Description of Garage X's Employees. 27

Fig. 4.2 The infoMap for the Job Description of Garage X's Employees. 29

Fig. 4.3 The infoSchema for the Waterfall Model of the Software Development Process. 30

Fig. 4.4 A Partial infoMap for the Waterfall Model of the Software Development Process. 31

Fig. 5.1 The SRD infoFrame. 34

Fig. 5.2 The infoSchema for the HyperDoc SRD. 38

Fig. 5.3 Defect Types for the Structure Level Analysis of an SRD. 39

Fig. 5.4 A Cardinality Defect for the HyperDoc SRD. 39

Fig. 5.5 Defect Types for the View Level Analysis of an SRD. 40

Fig. 5.6 A Definition Defect for the HyperDoc SRD. 40

Fig. 5.7 An Assignment Defect for the HyperDoc SRD. 41

Fig. 5.8 A Sequence Defect for the HyperDoc SRD. 42
Fig. 5.9 A Guard Defect for the HyperDoc SRD. 43
Fig. 5.10 A Flow Defect for the HyperDoc SRD. 45
Fig. 5.11 Defect Detection Strategies for the Structure Level. 46
Fig. 5.12 Defect Detection Strategies for Both Analysis Levels. 46

Fig. 6.1 The SCD infoFrame. 50
Fig. 6.2 The infoSchema for the convert Program SCD. 52
Fig. 6.3 Defect Types for the Structure Level Analysis of an SCD. 53
Fig. 6.4 Defect Types for the View Level Analysis of an SCD. 54
Fig. 6.5 A Guard Defect for the convert Program. 55
Fig. 6.6 A Typing Defect for the convert Program. 57
Fig. 6.7 Defect Detection Strategies for the Structure Level. 57
Fig. 6.8 Defect Detection Strategies for Both Analysis Levels. 58

Fig. 7.1 The infoSchema for the NED SRD. 61
Fig. 7.2 A Comparison of Defect Types. 62

# Introduction

The objective of this thesis is to provide a standard software inspection methodology for the software engineers. The inspection technique presented is based on Fagan's inspection method and the infoStructure concept. While the method developed by Fagan [89] has gained success in the industry, the notion of the infoStructure has barely reached the business world. However, it has been used for a number of years in the Computer Science Department at Concordia University.

The motivation to produce an enhanced software inspection methodology is attributed to several factors. The impact of the actual software crisis is the primary cause. The time and effort spent for the maintenance of software products has become far too expensive. The current software development process has difficulty in producing high quality software. The techniques employed during the verification and validation process do not supply the software engineers with a proof that programs are free from defects.

This thesis proposes an inspection methodology aimed at achieving higher software quality and reliability. The proposed inspection method is considered to be an enhancement to Fagan's technique. The infoStructure representation of the software deliverables improves the visual recognition of defects. Defect types and detection strategies used during the inspection process are directly derived from this representation. Their generalization to all software deliverables is a step towards standardization.

This thesis consists of seven chapters and eight appendices. Chapter 1 discusses the particulars of the software development process. The emphasis is on both the verification and validation process and the maintenance process. The current practices in software inspection are detailed in chapter 2. The inspection technique introduced by Fagan is fully described. Chapter 3 presents the inspection method as proposed by the author. The infoStructure concept is explained in chapter 4. The next two chapters constitute the proof that the proposed methodology is useable and good. The application of the proposed

inspection method to a Software Requirements Document (SRD) and a Software Code Document (SCD) are demonstrated in chapters 5 and 6 respectively. Chapter 7 concludes with a discussion about the benefits and weaknesses of the proposed technique. Appendix A shows the infoStructure representation of the waterfall model of the software development process. The infoStructure notation used throughout this thesis is detailed in appendix B. The original textual format of two SRDs and one SCD are displayed in appendices C, G, and E. Their infoStructure counterpart are displayed in appendices D, H, and F respectively.

# Chapter 1
# The Software Development Process

A software system inherits the complexity of the application and embedding system. Complexity is an intrinsic part of software. Better notational technology and "visualisation" helps eliminate only avoidable complexity [110]. The software development process currently in use is also complex. It involves a number of professionals, sometimes up to a few thousand, grouped into specialized teams. Interaction among teams and team members themselves is crucial to the development of a complete software product. Communication with users is also necessary to fully understand the application and its requirements.

The most common software development process currently employed is the waterfall model, Figure 1.1, or one of its derivatives [22], [48], [55], [106]. Its products take the form of Software Deliverables (SD) such as Software Requirement Document (SRD), Software Specification Document (SSD), Software Design Document (SDD), and Software Code Document (SCD). A significant amount of information created during the process is not recorded in any of the SDs. There exists a strong interdependence among SDs and a significant overlap of information in and among them, revealing a high level of redundancy. Furthermore, the notation or language used to produce each SD often differs from one document to the other. In order to perform consistently and avoid misinterpretation, a lot of time and effort is spent on communicating, understanding and updating the dependent and redundant SDs. In spite of the effort, the conventional notations and existing software processes are unable to assure consistency among SDs.

The remainder of the chapter discusses two phases of the software development process. First, the verification and validation process is defined. The various techniques employed for verification and validation are introduced. Then, a detailed definition of the maintenance process follows. A distinction is made between software maintenance and software upgrade. Finally, the properties of a new software development process are enumerated.

## 1.1 Software Verification and Validation

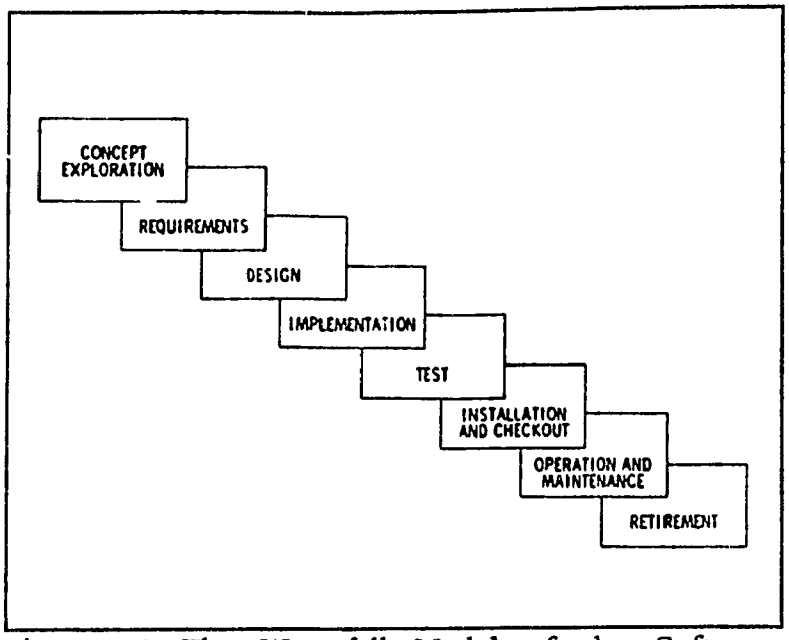



Figure 1.1 The Waterfall Model of the Software Development Process [106].

Verification is the process of determining whether or not the products of a given phase of the software development cycle fulfil the requirements established during the previous phase [106]. Validation is the process of evaluating software at the end of the software development process to ensure compliance with software requirements [106]. The objectives of both verification and validation is to detect defects [26]. A defect is defined as a non-conformance to requirements [80], [106].

The verification process attempts to find defects as early as possible in the software life cycle [26], [69]. Verification consists of checking a product against four criteria: completeness, consistency, feasibility, and testability [65], [69]. The methods of achieving this goal can be grouped in three categories: formal verification, reviews, and testing.

The process of verifying a product by mathematical proofs is called formal verification. This method requires the use of a formal specification language and a verification method (axioms and inferences) [26], [69], [73]. The biggest advantage of formal verification is its high reliability [69], [73]. However, there exists a number of imperfections which limits its usage [59], [61], [69], [73]. Moreover, formal verification is difficult to perform, specially on sizable products.

A review is a human-driven activity aimed to find defects in a product [26], [73]. It is divided into two phases. The first phase, management review, is a review of the development process itself, while the second phase is a review of the product at various stages during the development process. Walkthroughs and inspections are two special kinds of review [26]. The former is a peer review technique. The later is a more formal review with a specific process management function. The advantages of review methods include flexibility and suitability to verify completeness and consistency [69], [73]. Reviews suffer from weaknesses such as a high cost in terms of person-time, and a dependence upon the expertise and commitment of the participants [73]. Nevertheless, review methods are commonly used in today's industry. Software inspections are the main concern of this thesis. The current techniques are further described in the next chapter.

Testing is a destructive activity with the objective of causing a product to fail. It is mainly performed on programs. Testing does not prove the absence of defects, but finds defects. The testing process includes unit testing and integration testing. Integration testing techniques are bottom-up, top-down, and sideways integration [26]. The testing process is a lengthy process which can not conceivably be performed manually, specially on large products. To assist in the process, there exists a number of tools. They can be categorized into four classes: static analyzers [26], [71], [72], [73], dynamic analyzers [26], [61], [72] [73], symbolic evaluators [26], [70], [72], [73], and reliability analysis models [62], [64], [68], [74]. Each class of tools has a number of restrictions. Therefore, only limited assistance can be provided.

## 1.2 Software Maintenance vs Software Upgrade

Today, most companies spend 50 to 80 percent of their information system effort and budget on software maintenance [94], [95], [97], [104]. Some industries are turning into maintenance shops [95]. Software maintenance has become a real burden for many. Users complaints include system bugs and failures, excessive code size, lack of documentation, inadequate training and the lack of flexibility of software where it comes to adding enhancements and new

functions. Moreover, the critical software maintenance jobs are still often filled in by young inexperienced programmers or "development washouts".

There is a common belief that the process currently used to maintain software suffers from a lack of up-to-date maintenance documentation [97], [104]. The problem was identified many years ago and still exists today. The update of all of the SDs for a product is an impossible task, as it would ask for a continuous update of each deliverable. Moreover, the consistency among the SDs would not be assured. Therefore, in practice, the only maintenance documentation th used, updated, and trusted during maintenance is Software Code Document . Other SDs are, in view of a continuous source code modification, less and less consistent and relevant. They usually were not initially designed for maintenance at all.

The National Bureau of Standards studies as well as the Lientz and Swanson surveys [94], [104] show that the process of software maintenance includes corrective maintenance (20%), adaptive maintenance (25%), perfecting maintenance (50%), and preventive maintenance (5%). This is shown on Figure 1.2. Corrective maintenance includes diagnosis and fixing design, logic, or coding defects. Adaptive maintenance consists of changes to a product to adjust to an external environment. Perfecting maintenance deals with enhancements to a product. Future reliability and maintainability is handled in preventive maintenance. Consequently, 80% of software maintenance is in reality a continuation, usually by a new team, of the software development process.

It is quite important to make a distinction between software maintenance and **software upgrade**. The former shall be used to represent what is currently known as corrective maintenance. The later shall incorporate adaptive, perfecting, and preventive maintenance. Since software upgrade is a continuation of the software development process, the techniques employed for software development are also valid methods for software upgrade.

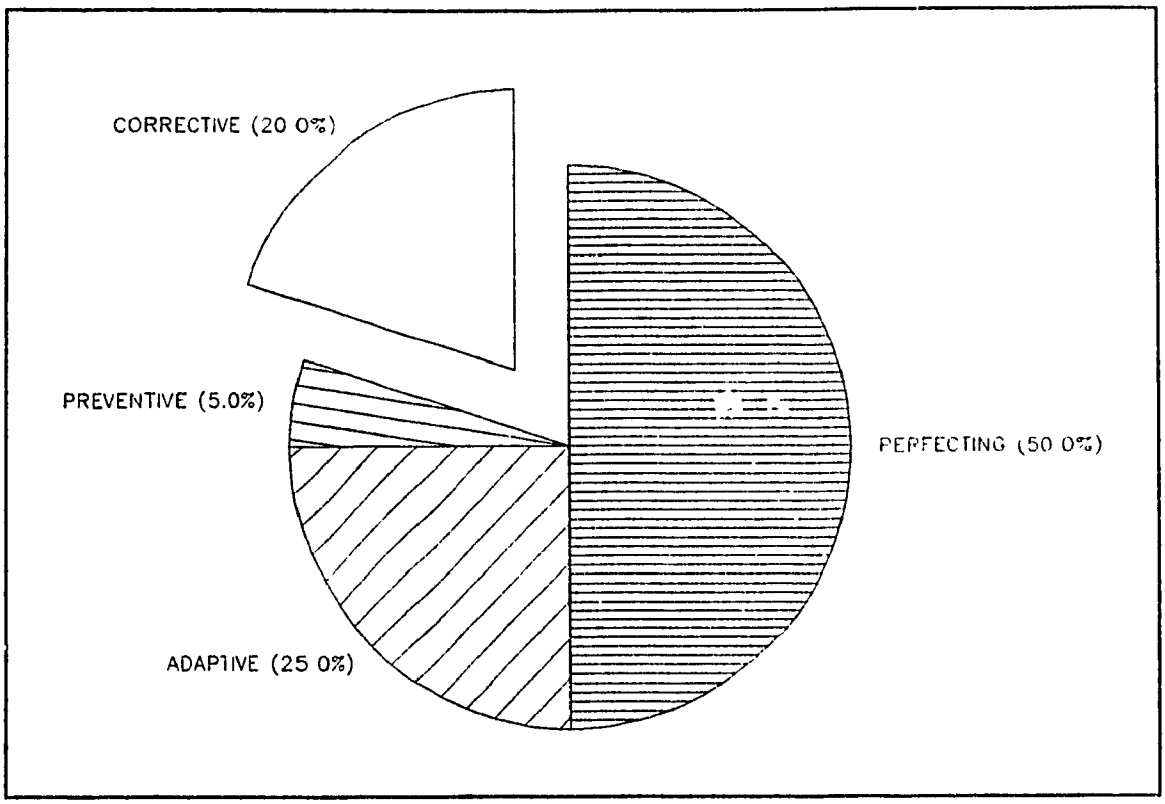



**Figure 1.2** The Software Maintenance Process.

## 1.3 A New Software Development Process

To solve the actual software crisis, a new software development process is necessary, one which:

a. recognizes that software requirements are incomplete and will continue to be incomplete [68], [94], [96];

b. designs software that facilitates change implementation [68], [96];

c. designs software for maintainability [104];

d. includes more solid mathematical foundations [94];

e. closes the gaps between steps of the software development process [94],

[95]; and

f. makes a clear distinction between software maintenance and software upgrade.

The major challenge in developing such a new process is considered by Webster to be the **Design Information Space** providing a good mechanism for "capture, representation, elaboration, reuse and presentation of information throughout the entire process, for cooperative use by people and machines within a surrounding organizational, social and educational structure" [109].

# Chapter 2
# State of the Art of Software Inspections

M.E. Fagan first introduced code and design inspections in software development in 1976 [89]. Fagan showed that a structured inspection methodology can increase productivity and improve final program quality. Since then, reports show that structured inspection has been used successfully at IBM [89], Bell Laboratories [87], AT&T Bell Laboratories [83], and Bell Northern Research (BNR) [76].

The methodology requires that the software development process be defined as a set of ordered operations [80], [85], [87], [89]. For each operation, an exit criteria is specified. This exit criteria must be satisfied for completion of the operation. To verify a program, checkpoints are inserted in the software development process as well as in each operation. Rework of all known errors to a checkpoint must be complete before the checkpoint can be claimed to be successful. This is due to the fact that the cost of rework in programs becomes higher the later the rework is done in the process.

The following sections of this chapter first describe the traditional inspection method of Fagan [89], along with its benefits and drawbacks. Then, an introduction to the two-person inspection method by Bisant and Lyle [77] follows. The chapter concludes with a summary of a number of articles comparing software inspections with other verification techniques.

## 2.1 The Traditional Inspection Method

An structures inspection is a formal process to detect defects in software products [81], [89]. Fagan uses inspections for program development verification. His analysis consists in the application of inspections for three particular checkpoints in the software development process:

a. after internal specification (I0),

b. after the design is complete (I1), and

c. after the code is written (I2).

Inspections are conducted by a team of four to eight people, depending on the product to be inspected. It is recommended that each team member has a specific role. Fagan describes four roles. The moderator is the key person of the team. He/she is a competent software developer from an unrelated project who offers leadership. The moderator should be specially trained as he/she manages the inspection team. The designer is the programmer responsible for producing the design for the inspected product. The coder or implementor is the programmer responsible for translating the design into code for the inspected product. Finally, the tester is the programmer responsible for writing and/or executing test cases. In [82], recommended individuals for specific inspections are displayed in two tables.

Fagan's inspection method consists of five steps: Overview, Preparation, Inspection, Rework, and Follow-up. Ackerman, Fowler, and Ebenau add Planning to this list as the initial step [87]. It is later confirmed by Fagan [80]. Figure 2.1 shows the traditional inspection method.

| | Fagan [89] | Ackerman [87] |
|---|---|---|
| Planning | | x |
| Overview | x | x |
| Preparation | x | x |
| Inspection | x | x |
| Rework | x | x |
| Follow-up | x | x |

Figure 2.1 The Traditional Inspection Method.

The Planning step embodies management procedures. First, a moderator is chosen. Then, the question is raised as to whether or not the Overview is necessary. Next, other members of the inspection team are selected, and the Overview and Inspection meetings are scheduled [80], [87].

The objectives of the Overview are the education of the inspection team, and communication among team members. The role of each member of the

inspection team is first defined. Then, the designer describes to the whole team the product to be inspected. At the end of the presentation, proper documentation is given to each of the team members for the next step, so that everyone in the inspection team can prepare for subsequent meetings [79], [80], [87], [89].

Education is the main objective of the Preparation phase. Here, all participants of the inspection team individually use the documentation supplied by the designer in the Overview to familiarize themselves with the product. The aim is to try to understand the intent and the logic of the design [80], [87], [89].

| Defect Id | Defect Definition |
|---|---|
| LO | Logic |
| TB | Test and Branch |
| EL | External Linkages |
| RU | Register Usage |
| SU | Storage Usage |
| DA | Data Area Usage |
| PU | Program Language |
| PE | Performance |
| MN | Maintainability |
| DE | Design Error |
| PR | Prologue |
| CC | Code Comments |
| OT | Other |

**Figure 2.2** Fagan's Defect Types [89].

Data-Reference Errors
Data-Declaration Errors
Computation Errors
Comparison Errors
Control-Flow Errors
Interface Errors
Input/Output Errors
Other Checks

**Figure 2.3** Myers' Defect Types [75].

The Inspection phase is where the team members attempt to find defects in the product. One member of the inspection team, designated as the reader, paraphrases the design. Every piece of logic must be covered at least once. Every branch must be taken at least once. Other members of the inspection team try to locate defects while the paraphrasing is done. For every defect discovered, the moderator records its type, class, and severity. If a solution is obvious, it is also recorded. But keep in mind that the aim of inspection is not to repair errors, but detect them. A list of common defect types, as shown on Figures 2.2 and 2.3, guides the team members in finding defects. A maximum of two sessions of two hours each per day is

recommended for this phase. At the end of each day, the moderator produces a summary report [79], [80], [87], [89].

In Rework, the author of the product fixes the defects found in the previous phase [80], [87], [89].

Follow-up is to ensure that all fixes are applied correctly. The moderator ensures that all of the reported defects are now repaired. If Rework covers more than five percent of the original design, a complete re-inspection is recommended. Otherwise, it is up to the moderator [80], [87], [89].

A progress rate is associated to each of the inspection steps. Figure 2.4 shows progress rates as proposed by a Fagan and Ackerman.

| Inspection Step | Progress Rate (NCSS/HR) | | | | |
|---|---|---|---|---|---|
| | I1 [89] | I2 [89] | I2 [80] | Code [87] | High/level and Low/level design[80] |
| Overview | 500 | | 500 | | |
| Preparation | 100 | 125 | 125 | 100 | |
| Inspection | 130 | 150 | 90 | | |
| Overall | | | 125 | | 250 |

Note: NCSS/HR = Non Commentary Source Statement per Hour.

**Figure 2.4** Inspection Progress Rates.

Throughout the inspection process, a collection of data is assembled. Figure 2.5 shows an example of information which is gathered. As explained in the next section, this collection of data is necessary for further analysis.

The following data should be collected for each inspection:

1. The date the product is distributed for inspection;
2. The date of the meeting;
3. The date the rework is complete;
4. Wether it is an initial inspection or a reinspection;
5. The type of inspection;
6. The identity of the product inspected;
7. The size of the material inspected;
8. The name of the moderator;
9. The number of inspectors;
10. Wether an Overview is held;
11. The Preparation time;
12. The meeting duration;
13. The number of defects of each type;
14. The severity of each defect; and
15. The disposition (pass/rework/reinspect) decided on by the inspection team.

**Figure 2.5** An Example of Information Collected Throughout the Inspection Process [79].

## 2.2 Inspection Benefits

Inspections as described by Fagan have numerous results. First, productivity and cost effectiveness are directly affected by inspections [75], [76], [77], [79], [82], [83], [86], [89]. Obviously, productivity highly depends on the progress rate of the inspection. Figure 2.6 displays the number of defects detected per inspection type from a number of authors. Buck and Ackerman [79], [86] report that the number of defects detected is independent of the type of code, the project, the individuals involved, the management team, and the customer. Three to five man-hours are necessary to detect a single major defect. Koontz [83] concludes that less than two percent of the total Series 5 software development effort was dedicated to inspections. More recently at BNR, Russell [76] reports that only one man-hour is necessary to find a single defect. Fagan determines that approximately two thirds of all defects reported during development are found by I1 and I2 inspections prior to machine testing. In his second article, Fagan reports

that inspections find from 60 to 90 percent of all defects. Consequently, rework tends to be managed more during the first half of the schedule. Without the use of inspections, the cost of defect rework is 10 to 100 times higher and is accomplished in large part during the last half of the schedule [85], [89]. Fagan's experiment shows a 23 percent increase in productivity for the coding operation. He also averages to 15 percent of the total project cost the cost of design and code inspections [80]. This figure is experimentally confirmed by Russell at BNR.

| Author | Detection Rate (Defect/KNCSS) | | | |
|---|---|---|---|---|
| | High-Level Design | Low-Level Design | Code | System Test |
| Buck [86] | 8-12 | 8-12 | 8-12 | 8-12 |
| Koontz [83] | 11 | 26 | 28 | 3 |
| Russell [76] | | | | |
| 150 NCSS/HR | | | 37 | |
| 750 NCSS/HR | | | < 8 | |
| Ackerman [79] | 7-20 | 7-20 | 7-20 | 7-20 |

Note: KNCSS = Thousand of Non Commentary Source Statements.

**Figure 2.6** Defects Detection Rates.

Second, the quality of software is also improved from the utilization of inspections throughout the development process [75], [76], [77], [79], [81], [82], [83], [85], [89]. For a seven months testing period, 38 percent less errors were discovered in a sample of a product developed with inspections than in a sample of the same product developed with walkthroughs [89]. With inspections, there exists a detailed feedback on product quality on a real-time basis, which directly leads to an improvement from programmers.

Third, analysis of the inspection results help for a continuous control and update of the inspection process itself [77], [81], [85], [87], [89]. For example, statistics on the number of defects by type help in determining and updating a list of the most common defects. The list can be used to guide inspection teams

to concentrate on certain defect types [75], [79], [80], [82], [84], [89].

Four, inserting inspections in the software development process facilitates the evaluation of project completeness and quality, as estimates can be done earlier in the process. Therefore, as inspections data is statistically collected, inspection methods sustain the software development process and project management [77], [78], [79], [80], [81], [82], [86], [87], [89].

Five, the quality of the programmers' work can be evaluated through inspections [87]. However, Fagan clearly specifies that inspections should never be used for the evaluation of a programmer performance [80], [89].

Last, individuals involved in the inspection process share coding information as well as programming expertise [75], [83].

## 2.3 Inspection Drawbacks

Results in [88] show that inspections concentrate too much on the logic aspect of programs. More attention should be placed on input/output anomalies. Myers suggests that programmers be trained to focus their attention on programs' data rather than exclusively on programs' logic.

Some participants of Myers experiment also report that there are cases where the inspection method would be enhanced by executing the program from a terminal.

## 2.4 The Two-Person Inspection Method

Bisant and Lyle [77] introduce an alternative to the traditional inspection method as developed by Fagan. The inspection process is basically similar to the one described above. The difference relies on the individuals involved in the inspections. The moderator is removed from the inspection team. The inspection is carried out by two individuals only, two programmers, one of them being the

author.

The experiment, as described by Bisant and Lyle, shows comparable results to Fagan's technique. Both lead to high quality products and to improved productivity. Also, the two-person inspection method shows significant individual productivity improvement. This effect is most evident on weaker programmers.

The two-person inspection method described in [77] is obviously more economic for an organization. It reduces the logistic needs required for inspections as well as the number of personnel involved.

## 2.5 Comparing Inspections with Other Verification Methods

The main difference between inspection techniques and other peer review methodologies is the data collection and analysis process [80], [82]. Throughout the inspection process, data is manually gathered through the use of standard forms [81], [82], [86], [87], [89]. Analysis can then be performed from the aggregation of data collected.

In [88], formal inspections are compared with methods which employed pairs of subjects who test a program independently and then pool their findings. Although the methods show similar efficiency in finding defects, the labour cost per defect is much higher for inspections. However, this result is obtained under specific conditions.

An interesting conclusion of Myers is that there exists a significant variability in the number and types of defects found by each person. Therefore, a team composed of two independent individuals finds more errors in testing than a single person, while their effectiveness is similar, in terms of labour cost. Inspection methods revealed less variability.

Inspection methods are adaptable. Their versatility enables them to work with primitive methodologies [87]. The methods have been successfully applied

to software as well as hardware [76], [79], [80], [82]. Inspections are best used when applied to structured documents such as design or code [76], [79], [80], [82], [83], [87]. In contrast, walkthroughs are preferred for unstructured documents such as requirements [87].

At BNR, an analysis shows that inspection methods approximately find one defect per man-hour, which is two to four times faster than defect detection using testing techniques. Ackerman also states that inspections are more cost-effective than testing for detecting defects, and from 2 to 10 times more efficient for removing defects [79]. The effect of inspections is not to eliminate testing techniques. It is rather proposed that inspections be perceived as a supplement to testing methods [75], [76], [79], [80], [87], [88].

# Chapter 3
# The Proposed Inspection Method

The inspection method proposed in this document is an enhancement to Fagan's traditional inspection method. The major change is that the inspection is carried out on a normalized form of the deliverable to be inspected. As a consequence, inspections are always performed on the same type of document. Inspection teams do not have to be confronted with different document standards, languages, or styles. Furthermore, considering that the normalized form of a deliverable is processable, time necessary for Rework and Follow-up can be considerably decreased.

The transformation of the deliverable to be inspected into a standard format is called Normalization. To be advantageous, the normalized form of a deliverable must possess a number of characteristics. First, it must be able to represent various types of documents. Within the software development process, such documents include the Software Requirements Document (SRD), Software Specification Document (SSD), Software Design Document (SDD), and the Software Code Document (SCD). Second, the normalized form must ensure consistency. Contradiction of information within the normalized form must be easily detected. Third, it must provide representation of analysis instruments such as hierarchy, flow diagrams, and state diagrams.

The normalized form used throughout this thesis is built using the infoStructure. The infoStructure possesses all of the characteristics necessary to the normalized form. Furthermore, it exhibits an additional interesting feature which can only improve the inspection process. The infoStructure elements are processable, which means that a number of views can be generated from them. Some of these views are documents such as an SRD, diagrams such as data flow diagram, or source code. The infoStructure concept is explained in great detail in chapter 4.

The inspection method proposed in this thesis is shown in Figure 3.1 using the infoStructure notation. Basically, it consists of the following six steps:

a. Planning,
b. Overview,
c. Normalization,
d. Analysis,
e. Rework, and
f. Follow-up.

The aim of each of these phases remains as described in the traditional inspection method. The Normalization procedure takes over the Preparation step, while the Inspection meeting is replaced by an Analysis meeting. All of the other steps must be adapted to the normalization concept.

The remainder of this chapter first identifies the participants to the proposed inspection method. Then, a detailed description of each of the six inspection steps follows. The discussion on View generation is left for future research. It is briefly covered in the conclusion.

## 3.1 Participants

A minimum of two individuals must be involved in the inspection process. It is preferred that roles be defined for each participant [89]. In the proposed inspection method, two roles are essential: the author and the inspector. The author is the person who is responsible for the production of the deliverable to be inspected. The inspector is an individual who works in a similar project as the author. The inspector possesses knowledge and experience about the normalization process, since one of his duty is to normalize the deliverable to be inspected.

As an example, the inspection of an SRD includes the author of the SRD, and an inspector. Additional persons may be part of the inspection team for a SRD inspection, such as the intended user, the developers, or the system test representatives [82], [87].

| | A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | A | A | A | A | 1 | { IDENTIFICATION } | |
| 2 | v | v | v | v | v | v | v | v | v | | | 1. infoMap for the Proposed Inspection Method. |
| 3 | A | A | A | A | A | A | A | A | A | 1 | { REFERENCE } | |
| 4 | v | v | v | v | v | v | v | v | v | | | 1. B. Deslauriers Master's Thesis. |
| 5 | A | A | A | A | A | A | A | A | A | 2 | { VIEW } | |
| 6 | v | v | v | v | v | v | v | v | v | | | 1 Proposed Inspection Process |
| 7 | . | . | . | . | . | . | . | . | . | v | | 2. Set Cardinality |
| 8 | O | O | O | O | O | O | O | O | O | 9 | { TRANSITION } | |
| 9 | o | . | . | . | . | . | . | . | . | | | 1. Start the inspection process. |
| 10 | . | o | . | . | . | . | . | . | . | | | 2. Wait for the entry criteria to be satisfied. |
| 11 | . | . | o | . | . | . | . | . | . | | | 3. Perform Overview. |
| 12 | . | . | . | o | . | . | . | . | . | | | 4. Normalize the deliverable. |
| 13 | . | . | . | . | o | . | . | . | . | | | 5. Perform Structure Level Analysis. |
| 14 | . | . | . | . | . | o | . | . | . | | | 6. Perform View Level Analysis. |
| 15 | . | . | . | . | . | . | o | . | . | | | 7. Perform Rework. |
| 16 | . | . | . | . | . | . | . | o | . | | | 8. Re-inspect the deliverable. |
| 17 | . | . | . | . | . | . | . | . | o | | | 9. Terminates the inspection process. |
| 18 | L | L | L | L | L | L | L | L | L | 8 | { INSPECTION STEP } | |
| 19 | s | . | . | . | . | . | . | . | . | | | 1. Planning |
| 20 | d | 1 | s | . | . | . | . | . | . | | | 2. Overview |
| 21 | . | . | d | s | . | . | . | d | . | | | 3. Normalization |
| 22 | . | . | . | d | s | . | . | . | . | | | 4. Structure Level Analysis |
| 23 | . | . | . | . | d | s | . | . | . | | | 5. View Level Analysis |
| 24 | . | . | . | . | e | e | s | . | e | | | 6. Rework |
| 25 | . | . | . | . | . | . | d | s | s | | | 7. Follow-up |
| 26 | . | . | . | . | . | d | . | . | d | | | 8. End |
| 27 | G | G | G | G | G | G | G | G | G | 3 | { PRE-CONDITION } | |
| 28 | . | f | t | . | . | . | . | . | . | | | 1. Entry criteria satisfied? |
| 29 | . | . | . | . | . | . | . | +f | t | | | 2. View Level Analysis done? |
| 30 | . | . | . | . | . | . | . | +f | t | | | 3. Rework involved less than 5% of deliverable? |
| 31 | S | S | S | S | S | S | S | S | S | 21 | { ACTION } | |
| 32 | 1 | . | . | . | . | . | . | . | . | | | 1. Choose participants. |
| 33 | 2 | . | . | . | . | . | . | . | . | | | 2. Schedule Overview meeting. |
| 34 | 3 | . | . | . | . | . | . | . | . | | | 3. Schedule Normalization. |
| 35 | 4 | . | . | . | . | . | . | . | . | | | 4. Schedule Analysis meeting. |
| 36 | 5 | . | . | . | . | . | . | . | . | | | 5. Schedule Rework. |
| 37 | 6 | . | . | . | . | . | . | . | . | | | 6. Schedule Follow-up. |
| 38 | . | . | 1 | . | . | . | . | . | . | | | 7. Assign a role to each participant. |
| 39 | . | . | 2 | . | . | . | . | . | . | | | 8. Introduce the deliverable to be inspected. |
| 40 | . | . | 3 | . | . | . | . | . | . | | | 9. Give documentation to each participant. |
| 41 | . | . | . | 1 | . | . | . | . | . | | | 10. Transform the deliverable to be inspected into a normalized form. |
| 42 | . | . | . | . | 1 | . | . | . | . | | | 11. Examine the composition of the normalized form. |
| 43 | . | . | . | . | 1 | . | . | . | . | | | 12. Perform Cardinality Analysis. |
| 44 | . | . | . | . | 1 | . | . | . | . | | | 13. Perform Structure Analysis. |
| 45 | . | . | . | . | . | 1 | . | . | . | | | 14. Examine the detailed information for each view of the normalized form. |
| 46 | . | . | | . | 1 | 1 | . | . | . | | | 15. Perform Redundancy Analysis. |
| 47 | . | | . | . | 1 | 1 | . | . | . | | | 16. Perform Consistency Analysis. |
| 48 | . | . | . | . | 1 | 1 | . | . | . | | | 17. Perform Flow Analysis. |
| 49 | . | . | . | . | 1 | 1 | . | . | . | | | 18. Perform Intention Analysis. |
| 50 | . | . | . | . | 1 | 1 | . | . | . | | | 19. Produce a report. |
| 51 | . | . | . | | . | . | 1 | | . | | | 20. Correct defects. |
| 52 | | . | . | . | . | . | . | 1 | 1 | | | 21. Ensure modifications are implemented properly. |
| 53 | G | G | G | G | G | G | G | G | G | 3 | { POST-CONDITION } | |
| 54 | . | . | . | . | f | . | . | | . | | | 1. Is a major defect found? |
| 55 | . | | . | . | . | f | | | . | | | 2. Is any defect found? |
| 56 | . | . | | | . | . | . | . | t | | | 3. Is Rework done properly? |

Figure 3.1 The infoMap for the Proposed Inspection Method.

## 3.2 Planning

As stipulated in chapter 1, the aim of the Planning step is management. Participants to the inspection are chosen. Each activity of the inspection process is scheduled, including an Overview meeting, Normalization, Analysis meeting, Rework, and Follow-up.

The difference with the traditional inspection method relies on the participants. As in the two-person method, the proposed inspection method does not include a moderator.

## 3.3 Overview

Education and communication remain the soul of the Overview. Each participant of the inspection team is assigned a specific role. Then, the author introduces the deliverable to be inspected to the other members of the team. At the conclusion of the meeting, appropriate documentation is given to each participant. Such documents include the deliverable to be inspected, as well as those documents required for the Normalization process as described in the next section.

## 3.4 Normalization

The essence of the Normalization step is familiarization. The inspector's goal is to transform the deliverable to be inspected into a normalized form. To accomplish this task, an inspector is required to possess not only knowledge but also experience on the transformation process. To assist the inspector, a template of the normalized form for the type of deliverable to be inspected is provided. The normalized form of the deliverable to be inspected is the result of the Normalization step.

This thesis uses the infoMaps as normalized forms. Templates are called infoFrames. Both infoMaps and infoFrames are part of the infoStructure concept

which is described in chapter 4.

The time and effort spent on normalization can be minimized if the Software Deliverables (SDs) are produced using the infoStructure notation.

### 3.4.1 Normalization vs Preparation

Normalization is an enhancement to the Preparation step of the traditional inspection method. No technique of how inspectors should familiarize themselves with the deliverable to be inspected is ever mentioned in current publications. Normalization is one such technique.

Normalization is a formal process in which specific information (views) about a particular document is delineated. Examples of such views for requirements are definitions and functional requirements, and for code, hierarchy and data flow. Furthermore, this information is permanently stored and is readily available for reference. Finally, the normalized form of the document is processable. Therefore, views can be illustrated under a number of different formats, such as text, tables, diagrams, graphs, and code.

## 3.5 Analysis

The purpose of the Analysis step is to find defects. Using the normalized form of the deliverable produced by the Normalization phase, the inspection team attempt to detect any defects.

The Analysis is performed at two distinct levels. They are identified as the Structure Level and the View Level. The analysis performed at the Structure Level consists in examining the composition of the normalized form. The inspector introduces the structure of the normalized form to the inspection team. The six detection strategies shown in Figures 3.2 and 3.3 are used to find possible defects. If a major defect is detected at the Structure Level, the analysis is interrupted. The deliverable is re-inspected after Rework is done.

If no major defect is detected at the Structure Level, the Analysis carries on at the View Level. Here, the inspector presents to the inspection team the detailed information for each view of the normalized form. The team members utilize the four detection strategies displayed in Figure 3.3 to detect defects.

1. Cardinality Analysis
2. Structure Analysis

**Figure 3.2** Defect Detection Strategies for the Structure Level.

3. Redundancy Analysis
4. Consistency Analysis
5. Flow Analysis
6. Intention Analysis

**Figure 3.3** Defect Detection Strategies for Both Analysis Levels.

When a defect is found, one member of the inspection team logs in its type, class and severity, along with the analysis level. A list of common defect types which depends on the category of the deliverable inspected and the analysis level, is provided during the Analysis to guide the inspection team. The two defect severity levels are identified as minor and major [76], [80], [82], [89]. A major defect is one which would result in a program malfunction. The three defect classes are wrong, missing, and extra [80], [82], [87], [89]. A report is produced at the end of each day.

Detection strategies identified above are common to all types of deliverable. Defect types vary from one deliverable to another. Both detection strategies and defect types are directly derived from the infoStructure notation. They are explained in details during the Requirements inspection in chapter 5.

### 3.5.1 Analysis vs Paraphrasing

The traditional inspection method suggests paraphrasing to detect defects. Each segment of the deliverable to be inspected is interpreted each time it is referred to. Moreover, the paraphrasing method does not supply the inspection

team with global views. Although this technique has been successful with code documents, its application to other types of documents can be difficult.

The analysis process as described above provides the inspection team with specific means of finding defects. As opposed to paraphrasing, no interpretation is necessary. The information contained in the document is not changed, but displayed in a structured fashion. The two levels of analysis provide the inspection team with global and detailed analysis techniques. Furthermore, the inspection team is supplied with six defect detection strategies which can be applied to a large number of document types.

### 3.6 Rework

The defects found by the inspection team are corrected during the Rework step. The author of the deliverable makes the rectifications in the original document. However, the modifications could be made on the normalized form. Since the infoStructure elements are processable, the SRD can be regenerated once the inspection process is complete.

### 3.7 Follow-up

The objective of the Follow-up step is to ensure that the modifications are implemented properly. The inspector is responsible to verify that all reported defects were fixed. Re-inspection is necessary when the Analysis was performed only at the Structure Level. It is also recommended when Rework involves more than five percent of the original document.

# Chapter 4
# The infoStructure

To introduce the fundamentals of the infoStructure technology, two examples are used. The first sample consists of describing the jobs for the employees of a garage. The second deals with the waterfall model of the software development process. All of the infoStructure elements are built using a commercial spreadsheet package.

The infoStructure has three elements: infoMaps, infoSchemas, and infoFrames. An infoMap contains detailed information about a particular document. An entire SRD can be transformed into a single infoMap. As an example, a complete infoMap for the job description of Garage X's employees is displayed in Figure 4.2. An infoSchema is associated to a single infoMap. The infoSchema shows how the information incorporated in the corresponding infoMap is organized. Figure 4.1 illustrates the infoSchema of the infoMap of Figure 4.2. An infoFrame is the generalization of an infoSchema over a number of similar infoMaps. The infoFrame serves as a template to construct other infoMaps of the same type. infoSchemas of these infoMaps are then used to update the original infoFrame. Therefore, there exists a dynamic cycle within the infoStructure.

The remainder of the chapter is organized as follows. The roots of the infoStructure are first identified. Then, the infoStructure notation is introduced. Two examples are utilized to help the reader visualize the infoStructure concept. Next, the characteristics of the InfoStructure elements are presented. Finally, the chapter closes with an observation about the training required to manipulate the infoStructure elements.

## 4.1 The History

The origin of the infoStructure concept is found in a 1976 paper co-written by W.M. Jaworski and G. Belkin [12]. Their work consisted of decision tables (TBL) to represent source code functionalities, and an interpretative experimental processor (TAP/X) to execute and analyze the tables.

The idea behind TBL was soon upgraded in the early 1980's, with the creation of relation tables (ABL). The ABL tables were used to capture information at a number of abstraction levels, from natural language to microcode [10]. The ABL based methodology (SOS) was successfully utilized by a specialized software company to specify, develop, and produce software [11].

Introduced in 1987 [8], [9], the J-Maps were the next step towards the actual infoStructure. The J-Maps were tables based on sets, where roles were identified for the sets and their elements. As with the ABL tables, information could be abstracted at many levels. At that time, the J-Maps were limited to the representation of the relation types found in a Entity-Relationship model: one-to-one, one-to-many, many-to-many, and unary relationships. In addition, a Meta-model was developed to illustrate the structure and size of the J-Maps relations [8].

The jMaps described in [7] constituted a positive enhancement to the former J-Maps. First, the Meta-model is formally named Schema. Then, the notation is extended. A number of roles for the sets and the set members are added. As a direct result, additional relationships such as hierarchy and flow can be represented. Finally, the article presents a formalization of the jMaps concept.

In 1990, the concept is enhanced once again. An extension is made to the jMap notation. jMaps are renamed infoMAPs, and Schemas infoSCHEMAs. An additional structure is introduced. The infoFRAME is a template employed to incrementally build an infoMAP. In [6], a classroom environment based on the three structures is described.

The latest modification to the three structures occurred in 1991. The notation was once more extended. The infoMAPs were renamed infoMaps, infoSCHEMAs infoSchemas, and infoFRAMEs infoFrames. This thesis utilizes the term infoStructure to represent the three structures.

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | A | A | A | 1 | { IDENTIFICATION } | |
| 2 | v | v | v | | | 1. infoSchema for the Job Description of Garage X's Employees. |
| 3 | A | A | A | 3 | { VIEW } | |
| 4 | v | v | . | | | 1. Company Structure |
| 5 | . | . | v | | | 2. Employee Job Description |
| 6 | . | . | . | v | | 3. Set Cardinality |
| 7 | O | . | . | 3 | { DEPARTMENT } | |
| 8 | M | H | O | 7 | { EMPLOYEE } | |
| 9 | . | . | F | 5 | { DELIVERABLE } | |
| 10 | . | . | M | 6 | { TASK } | |

Figure 4.1 The infoSchema for the Job Description of Garage X's Employees.

## 4.2 The Notation

The notation of the infoStructure is based on set theory. An infoSchema may be viewed as a collection of sets and their relationships. The infoSchema shown in Figure 4.1 has 6 sets, listed in column E. The cardinality of these sets appears in column D. The set { VIEW } has 3 members. They are **Company Structure**, **Employee Job Description**, and **Set Cardinality**.

Capital letters are used to represent relationships among the sets. The letter "A" allows the user to denote that a cluster of columns has been grouped into a horizontal relationship. The letter "O" is used to represent the dominant set of a relationship. The letter "v" is simply used as a column marker. An example of a one-to-many relationship is shown in column A. Entries "A", "v", "A", "v", "O", "M" mean that within the **Company Structure** view, O-ne member of the set { DEPARTMENT } is vertically related to M-any members of the set { EMPLOYEE }. The letter "S" replaces the letter "M" when sequential ordering of the set members is relevant. In the case of a one-to-one relationship, letters "O" and "T" are utilized.

The letter "H" indicates a vertical hierarchy among the set members. For example, in column B, entries "A", "v", "A", "v", "H" indicate that the **Company Structure** view is composed of a hierarchy relationship within the members of the set { EMPLOYEE }.

The flow of information is represented by the letter "F". For example, entries "A", "v", "A", "v", "O", "F", "M" in column C specify that within **Employee Job Description** view, O-ne member of the set { EMPLOYEE } is vertically related to a F-low of members of the set { DELIVERABLE }, and to M-any members of the set { TASK }.

The **Set Cardinality** view reveals the number of members in each set of the infoStructure element. This view is part of all infoSchemas and infoMaps shown in this report.

An infoSchema specifies the relationships between sets. An infoMap records relationships between the members of the sets. Lower-case letters are used to symbolize relationships among set members. Figure 4.2 displays the infoMap associated with the infoSchema of Figure 4.1. The structure of column A to C of the infoMap is specified by the column A of the infoSchema. The affiliation of the employees within the each department of the company is presented in these three columns. For example, column C shows that employees **I.O. You**, **I. Workman**, and **U. Too**, all belong to the **Automobile** department of Garage X.

The structure of column D through F of the infoMap is defined from column B of the infoSchema of Figure 4.1. The employee hierarchy of Garage X is completely represented by these three columns. Letters "p" and "c" are used to indicate the p-arent and its c-hildren. For example, in column D, **U.R. Rich** is the supervisor of employees **I.M. Next** and **I.O. You**. Columns E and F show that both of these employees are themselves supervisors of a two other employees.

The job description for each employee of Garage X is displayed in column G through M of the infoMap of Figure 4.2. Letters "i", "o", and "b" are employed to represent the flow of information. For instance, column H shows that the tasks of employee **I.M. Next** are to **File bills** and **Fill in vehicles with fuel**. An i-nput deliverable associated with **I.M. Next** job description is a **Written Order**, while an o-utput deliverable consists of a **Verbal Order**. **Bills** are b-oth input and output for

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | A | A | A | A | A | A | A | A | 1 | { IDENTIFICATION } | |
| 2 | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 1. infoMap for the Job Description of Garage X's Employees |
| 3 | A | A | A | A | A | A | A | A | A | A | A | A | A | 3 | { VIEW } | |
| 4 | v | v | v | v | v | v | | . | . | . | . | | | | | 1 Company Structure |
| 5 | . | . | . | . | . | . | v | v | v | v | v | v | v | | | 2 Employee Job Description |
| 6 | . | . | . | . | . | . | . | . | . | | . | . | . | v | | 3. Set Cardinality |
| 7 | O | O | O | . | . | . | . | . | | . | . | . | . | 3 | { DEPARTMENT } | |
| 8 | o | . | . | . | . | | . | . | | . | . | . | . | | | 1. Management |
| 9 | . | o | . | . | . | . | . | . | . | . | . | | . | | | 2. Fuel |
| 10 | . | . | o | . | . | . | . | . | . | . | . | . | . | | | *3. Automobile* |
| 11 | M | M | M | H | H | H | O | O | O | O | O | O | O | 7 | { EMPLOYEE } | |
| 12 | v | . | . | p | . | . | o | . | . | . | . | . | . | | | *1. U.R. Rich* |
| 13 | . | v | . | c | p | | . | o | . | . | . | . | | | | *2. I.M. Next* |
| 14 | . | v | . | . | c | . | | . | o | . | | . | . | | | 3 D.J Cool |
| 15 | . | v | . | . | c | . | . | . | . | o | . | . | . | | | 4 U.C. Birds |
| 16 | . | . | v | c | | p | . | . | | | o | . | . | | | *5. I.O. You* |
| 17 | . | . | v | . | | c | . | . | | | . | o | . | | | *6. I. Workman* |
| 18 | . | . | v | . | . | c | . | . | | . | . | . | o | | | *7. U. Too* |
| 19 | . | . | . | . | . | | F | F | F | F | F | F | F | 5 | { DELIVERABLE } | |
| 20 | . | . | . | . | . | . | b | . | . | . | | . | . | | | 1. Business plan |
| 21 | . | . | . | . | | . | o | i | . | . | i | | . | | | *2. Written order* |
| 22 | . | . | . | . | . | . | . | o | i | i | o | i | i | | | *3. Verbal order* |
| 23 | . | . | . | . | | . | i | b | o | o | b | o | o | | | *4. Bills* |
| 24 | . | . | . | . | . | . | . | . | . | . | b | o | o | | | 5 Estimates |
| 25 | . | . | . | . | . | . | M | M | M | M | M | M | M | 6 | { TASK } | |
| 26 | . | . | . | . | | . | . | . | | . | v | v | v | | | 1. Estimate damages on vehicles |
| 27 | . | . | . | . | . | . | . | . | . | | v | v | v | | | 2. Fix vehicles upon client approbation. |
| 28 | . | . | . | . | . | . | . | . | . | | v | | | | | 3 File estimates |
| 29 | . | . | . | . | . | . | v | v | . | . | v | . | . | | | *4. File bills.* |
| 30 | . | . | . | . | | . | . | v | v | v | . | . | . | | | *5. Fill in vehicles with fuel.* |
| 31 | . | . | . | . | . | . | v | . | . | | . | . | . | | | 6 Perform administrative and management functions |

Figure 4.2 The infoMap for the Job Description of Garage X's Employees.

I.M. Next's job.

Figure 4.3 presents the infoSchema for the waterfall model of the software development process. This infoSchema possesses 8 sets and two views. As in the previous example, the **Set Cardinality** view exhibits the number of members in each set of the infoSchema. The other view, **Software Life Cycle Process**, is an example of a state machine. The { STATE } set represents the nodes while the arcs between them are symbolized by the { TRANSITION } set. Criteria which must be met to allow a transition to fire are part of the sets { PRE-CONDITION } and { EVENT }. The set { ACTION } refers to the tasks that are performed when a transition takes place. In column A, entries "A", "v", "A", "v", "A", "v", "O", "L", "G", "G", "S" indicate that the O-ne member of the set { TRANSITION } L-inks some members of the set { STATE } if the G-uards from the { PRE-CONDITION } and { EVENT } are fulfilled. In doing so, a S-equence of members of the set { ACTION } are performed.

A part of the infoMap for the waterfall model of the software development process is shown Figure 4.4. The entire infoMap is illustrated in appendix A. Column A through AJ are defined from column A of its corresponding infoSchema, displayed in Figure 4.3. Letters "s" and "d" correspond respectively to the source and the destination state. The letter "l" is used when the source and the destination state is the same, i.e. when the transition l-oops on a state. Letters "t" and "f" stand respectively for true and false.

| | A | B | C | D |
|---|---|---|---|---|
| 1 | *A* | 1 | { IDENTIFICATION } | |
| 2 | *v* | | | 1. infoSchema for the Waterfall Model |
| 3 | . | | | of the Software Development Process. |
| 4 | *A* | 1 | { REFERENCE } | |
| 5 | *v* | | | 1. IEEE Software Engineering Standards [55] |
| 6 | *A* | 2 | { VIEW } | |
| 7 | *v* | | | 1. Software Life Cycle Process |
| 8 | . | v | | 2. Set Cardinality |
| 9 | *O* | 36 | { TRANSITION } | |
| 10 | *L* | 9 | { STATE } | |
| 11 | *G* | 7 | { PRE-CONDITION } | |
| 12 | *G* | 7 | { EVENT } | |
| 13 | *S* | X | { ACTION } | |

Figure 4.3 The infoSchema for the Waterfall Model of the Software Development Process.

For instance, column J of Figure 4.4 shows that the transition from the **Design** to the **Implementation** states takes places if the pre-condition **Design phase completed?** is t-rue. In column T, the transition which l-oops on the **Test** state occurs if the pre-condition **Test phase completed?** is f-alse OR the event **Test change** is t-rue.

Appendix B presents the infoMap for the infoStructure notation used throughout this thesis.

| | J | K | L | M | N | O | P | Q | R | S | T | U | AK | AL | AM |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | A | A | A | A | A | A | A | 1 | { IDENTIFICATION } | |
| 2 | v | v | v | v | v | v | v | v | v | v | v | v | | | 1. Partial infoMap for the Waterfall Model |
| 3 | | | | | | | | | | | | | | | of the Software Development Process. |
| 4 | A | A | A | A | A | A | A | A | A | A | A | A | 1 | { REFERENCE } | |
| 5 | v | v | v | v | v | v | v | v | v | v | v | v | | | 1. IEEE Software Engineering Standards [55] |
| 6 | A | A | A | A | A | A | A | A | A | A | A | A | 2 | { VIEW } | |
| 7 | v | v | v | v | v | v | v | v | v | v | v | v | | | 1. Software Life Cycle Process |
| 8 | . | . | . | . | . | . | . | . | . | . | . | . | v | | 2. Set Cardinality |
| 9 | O | O | O | O | O | O | O | O | O | O | O | O | 36 | { TRANSITION } | |
| 19 | . | . | . | . | . | . | . | . | . | . | . | . | | | 10 |
| 20 | . | o | . | . | . | . | . | . | . | . | . | . | | | 11 |
| 21 | . | . | o | . | . | . | . | . | . | . | . | . | | | 12 |
| 22 | . | . | . | o | . | . | . | . | . | . | . | . | | | 13 |
| 23 | . | . | . | . | o | . | . | . | . | . | . | . | | | 14 |
| 24 | . | . | . | . | . | o | . | . | . | . | . | . | | | 15 |
| 25 | . | . | . | . | . | . | o | . | . | . | . | . | | | 16 |
| 26 | . | . | . | . | . | . | . | o | . | . | . | . | | | 17 |
| 27 | . | . | . | . | . | . | . | . | o | . | . | . | | | 18 |
| 28 | . | . | . | . | . | . | . | . | . | o | . | . | | | 19 |
| 29 | . | . | . | . | . | . | . | . | . | . | o | . | | | 20 |
| 30 | . | . | . | . | . | . | . | . | . | . | . | o | | | 21 |
| 46 | L | L | L | L | L | L | L | L | L | L | L | L | 9 | { STATE } | |
| 48 | . | d | . | . | . | . | d | . | . | . | . | . | | | 2. Concept |
| 49 | . | . | d | . | . | . | . | d | . | . | . | . | | | 3. Requirements |
| 50 | s | . | . | d | . | . | . | . | d | . | . | . | | | ***4. Design*** |
| 51 | d | s | s | s | I | s | . | . | . | d | . | . | | | ***5. Implementation*** |
| 52 | . | . | . | . | . | d | s | s | s | s | I | s | | | 6. Test |
| 53 | . | . | . | . | . | . | . | . | . | . | . | d | | | 7. Installation and checkout |
| 56 | G | G | G | G | G | G | G | G | G | G | G | G | 7 | { PRE-CONDITION } | |
| 59 | t | . | . | . | . | . | . | . | . | . | . | . | | | ***3. Design phase completed*** |
| 60 | . | . | . | . | +f | t | . | . | . | . | . | . | | | 4. Implementation phase completed? |
| 61 | . | . | . | . | . | . | . | . | . | . | +f | t | | | ***5. Test phase completed?*** |
| 64 | G | G | G | G | G | G | G | G | G | G | G | G | 7 | { EVENT } | |
| 66 | . | t | . | . | . | . | t | . | . | . | . | . | | | 2. Concept change |
| 67 | . | . | t | . | . | . | . | t | . | . | . | . | | | 3. Requirements change |
| 68 | . | . | . | t | . | . | . | . | t | . | . | . | | | 4. Design change |
| 69 | . | . | . | . | +t | . | . | . | . | t | . | . | | | 5. Implementation change |
| 70 | . | . | . | . | . | . | . | . | . | . | +t | . | | | ***6. Test change*** |
| 72 | S | S | S | S | S | S | S | S | S | S | S | S | X | { ACTION } | |

Figure 4.4 A Partial infoMap for the Waterfall Model of the Software Development Process.

## 4.3 Characteristics

The infoStructure is based on set theory. The infoSchemas are used to define the relationships between sets and serve to prescribe the information structures within the Design Information Space [109]. The infoFrames are derived automatically from the infoSchemas and provide the Information Space into which

information is inserted forming infoMaps.

The infoStructure notation exploits the properties of fundamental mathematical concepts such as sets, relations, graphs, and functions. The infoStructure provides a mechanism which facilitates the reuse of existing information. Therefore, the use of the infoStructure removes artificial borders, minimizes redundancy and complexity, and ensures consistency.

The elements of the infoStructure are processable. The processes and products represented with the infoStructure are also processable. Software deliverables are easily derived from the infoStructure elements, and their consistency can be assured by careful inspection of the elements.

Finally, the infoStructure notation is flexible. The expressions utilized to represent relationships are at the user's discretion.

## 4.4 Training

The infoStructure notation is quite different from commonly used notations. No other notation encompasses so much information under one document. The multidimensional relations created in the infoStructure elements are often unfamiliar to most software engineers. Therefore, training is essential to manipulate infoStructure elements efficiently.

The time required for training on infoStructure elements and notation has not been clearly identified. A very limited number of individuals are currently trained to deal efficiently with the infoStructure concept. Most of these individuals have been training on a part time basis. Since there exists no formal courses on the infoStructure concept, the knowledge is acquired through a hands-on training method. Therefore, the time taken by these persons to become familiar with the methodology greatly varies. It is estimated that training requires from two to six months.

# Chapter 5
# Requirements Inspection

This chapter represents a validation of the proposed methodology by construction and comparison. The proposed inspection method is directly applied to a Software Requirements Document (SRD).

The first phase of most models of the software development process is the Requirements phase [22], [48], [55]. The output of the Requirements phase is a Software Requirements Document (SRD). This initial phase is unique as it does not possess any previous document for verification.

An appropriate entry criteria for the inspection of requirements is a complete SRD [82]. The exit criteria is that all defects found during the inspection process must be corrected and verified. This exit criteria is similar for the inspection of any document.

In addition to the author of the SRD and an inspector, supplementary participants are necessary for the inspection of requirements. Individuals such as the intended user, developers, and system test representatives are strongly encouraged to join the inspection team [82], [87].

The remainder of this chapter is structured as follow. The Normalization process for a SRD is first described. The second section explains the Analysis process for a SRD. The SRD for HyperDoc as shown in appendix C is used to illustrate both processes. This SRD was done by a group of graduate students from Concordia University as an assignment for a Software Engineering course. The Planning, Overview, Rework, and Follow-up steps of the proposed inspection method were intentionally left out. They were explained in chapter 3 and do not necessitate further elaboration.

## 5.1 SRD Normalization

The inspector is directly responsible for the Normalization of the SRD. The normalized form is based on IEEE Standard 830-1984 [47]. Figure 5.1 shows the infoFrame for SRDs complying with this standard. Six views and 18 sets are necessary to completely map all of the information of a SRD.

| | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | A | A | A | 1 | { IDENTIFICATION } | |
| 2 | v | v | v | v | v | v | v | v | | | 1. SRD infoFrame. |
| 3 | A | A | A | A | A | A | A | A | X | { REFERENCE } | |
| 4 | A | A | A | A | A | A | A | A | 6 | { VIEW } | |
| 5 | v | . | . | . | . | . | . | . | | | 1. Definitions |
| 6 | . | v | v | . | . | . | . | . | | | 2. Product Perspective |
| 7 | . | . | . | v | v | . | . | . | | | 3. Product Functions |
| 8 | . | . | . | . | . | v | v | . | | | 4. Functional Requirements |
| 9 | . | . | . | . | . | . | . | v | | | 5. Product Users |
| 10 | . | . | . | . | . | . | . | . | v | | 6. Set Cardinality |
| 11 | O | . | . | . | . | . | . | . | X | { DEFINITION } | |
| 12 | . | . | O | . | . | . | . | . | X | { TRANSITION } | |
| 13 | . | . | L | . | . | . | . | . | X | { STATE } | |
| 14 | T | . | . | . | . | . | . | O | X | { USER TYPE } | |
| 15 | . | . | . | . | . | H | O | . | X | { OPERATION } | |
| 16 | T | H | F | . | . | . | F | . | X | { EXTERNAL OBJECT } | |
| 17 | T | H | F | O | O | . | F | M | X | { INTERNAL OBJECT } | |
| 18 | . | . | . | G | . | . | . | G | X | { PRE-GUARD } | |
| 19 | . | . | G | . | . | . | G | . | X | { PRE-CONDITION } | |
| 20 | . | . | . | . | . | . | G | . | X | { EVENT } | |
| 21 | . | . | M | . | M | H | . | . | X | { TASK } | |
| 22 | . | . | . | . | . | . | S | . | X | { ACTION } | |
| 23 | . | . | G | . | . | . | G | . | X | { POST-CONDITION } | |
| 24 | . | . | . | G | . | . | G | . | X | { POST-GUARD } | |
| 25 | . | . | . | . | M | . | M | . | X | { REMARK } | |

Figure 5.1 The SRD infoFrame.

### 5.1.1 Sets

The sets which are part of the SRD infoFrame are listed in column J of Figure 5.1. The { IDENTIFICATION } set is used to identify the infoStructure element. The { REFERENCE } set lists the reference documents cited in the SRD.

Also included in this set is IEEE Standard 830-1984.

Definitions of terms, acronyms and abbreviations are the members of the { DEFINITION } set. The { OPERATION } set contains the name of each operation listed in the Functional Requirements section of [47]. Objects identified in a SRD are grouped into two sets, { EXTERNAL OBJECT } and { INTERNAL OBJECT }. The external and internal attributes are relative to the product itself. Members of these two sets are mainly found in the Definition and Notations sections of [47]. However, additional objects may appear throughout the SRD. The { PRE-GUARD } set embodies global conditions of the product which must be satisfied before the design/operation of the product. Members of this set are found in the Assumptions and Dependencies, and General Constraints sections of [47]. The tasks of the product are the members of the set { TASK }. They are located in the Product Functions section of [47].

The { PRE-CONDITION }, { EVENT }, { ACTION }, and { POST-CONDITION } set members come from each operation defined in the Functional Requirements section of [47]. Conditions and events which must be satisfied prior to the execution of an operation respectively form the members of the { PRE-CONDITION } and { EVENT } sets. Specific actions performed by an operation constitute the members of the { ACTION } set. Specific conditions that must be met to terminate an operation compose the { POST-CONDITION } set.

The members of the set { REMARK } are comments associated to the tasks of the product, or to the specific operations.

The { POST-GUARD } set contains global conditions of the product that must be fulfilled after the design/operation of the product. It also includes general conditions which must be satisfied to terminate an operation. Members of this set come from the Purpose, Scope, General Constraints, Performance Requirements, and Attributes sections of [47], or from the Functional Requirements section. Intended users of the product are the members of the set { USER TYPE }. They can be found in the User Characteristics section of [47].

It is not obvious how to detect the members of the sets { TRANSITION } and { STATE } in a conventional SRD. Most of the time, they are just not included. Therefore, when necessary, the members of these two sets are simply represented with identification numbers.

### 5.1.2 Views

The first view, **Definitions**, associates a definition to each ambiguous term employed to identify an external/internal object or a user type.

The **Product Perspective** view consists of two relation types. First, the hierarchy among external and internal objects is shown. Second, a state diagram is used to show the relationship between the product and its environment. Although this view is part of the IEEE standard, it is rarely included in a SRD.

The **Product Functions** view lists the conditions attached to the design of the product, and its main tasks. Two relations are necessary to illustrate this view. The first relation displays a number of pre-guards which must be satisfied before the design of the product can start. Also included are the post-guards which have to be met before the design terminates. The second view shows the tasks that the product have to perform. These tasks may be part of the current or future requirements for the product.

The hierarchy between the product tasks and the operations appearing in the Functional Requirements section of [47] forms the first relation of the **Functional Requirements** view. The details of each operation is reflected in the second relation. Each operation is associated with:

a. a number of pre-conditions and events which must be fulfilled prior to the execution of the operation;

b. a sequential list of actions to be performed by the operation;

c. a number of post-guards and post-conditions which must be met before the operation is allowed to conclude;

d. the flow of external and internal objects (input and output) for the operation; and

e. remarks concerning the operation.

The **Product Users** view depicts the intended users for the product. A user may have to meet certain conditions to properly utilize objects within the product.

Final.y, the **Set Cardinality** set simply states the number of members in each set of the infoStructure.

### 5.1.3 Results

The Normalization step concludes when an infoSchema and an infoMap of the SRD to be inspected are completed. Both infoStructure elements are necessary for the Analysis step. Figure 5.2 shows an example of infoSchema for the HyperDoc SRD. The corresponding infoMap is displayed in appendix D. The whole document in appendix C was normalized with the exception of the Functional Requirements section. The functional requirements involving security within HyperDoc, i.e. sections 3.1.45 to 3.1.56 of appendix C, were the only ones normalized. The normalization of that entire section would have significantly expanded the infoMap but not induced any additional features.

## 5.2 SRD Analysis

For best results, all of the members of the inspection team must be present for the Analysis step. The inspector is the one who conducts the Analysis process. Analysis is performed on the normalized form obtained from the Normalization step. The infoSchema is analyzed first (Structure Level). The inspection team tries to find defects to the infoSchema using the list of common defect types

| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | A | A | 1 | { IDENTIFICATION } | |
| 2 | v | v | v | v | v | v | v | | | 1. infoSchema for the HyperDoc SRD. |
| 3 | A | A | A | A | A | A | A | 8 | { REFERENCE } | |
| 4 | A | A | A | A | A | A | A | 6 | { VIEW } | |
| 5 | v | . | . | . | . | . | . | | | 1. Definitions |
| 6 | . | v | . | . | . | . | . | | | 2. Product Perspective |
| 7 | . | . | v | v | . | . | . | | | 3. Product Functions |
| 8 | . | . | . | . | v | v | . | | | 4. Functional Requirements |
| 9 | . | . | . | . | . | . | v | | | 5. Product Users |
| 10 | . | . | . | . | . | . | . | v | | 6. Set Cardinality |
| 11 | O | . | . | . | . | . | . | 29 | { DEFINITION } | |
| 12 | T | . | . | . | . | . | O | 6 | { USER TYPE } | |
| 13 | . | . | . | . | H | O | . | 12 | { OPERATION } | |
| 14 | . | H | . | . | . | . | . | 1 | { EXTERNAL OBJECT } | |
| 15 | T | H | O | O | . | F | M | 45 | { INTERNAL OBJECT } | |
| 16 | . | . | G | . | . | . | . | 11 | { PRE-GUARD } | |
| 17 | . | . | . | . | . | G | . | 15 | { PRE-CONDITION } | |
| 18 | . | . | . | M | H | . | . | 11 | { TASK } | |
| 19 | . | . | . | . | . | S | . | 19 | { ACTION } | |
| 20 | . | . | . | . | . | G | . | 8 | { POST-CONDITION } | |
| 21 | . | . | G | . | . | G | . | 27 | { POST-GUARD } | |
| 22 | . | . | . | M | . | M | . | 5 | { REMARK } | |

Figure 5.2 The infoSchema for the HyperDoc SRD.

shown in Figure 5.3. If a major defect is found at this level, the Analysis terminates and the deliverable is sent for Rework.

If no major defect is detected in the infoSchema, the infoMap is analyzed (View Level). The inspector displays each view of the infoMap to the other participants. The list of common defect types shown in Figure 5.5 is used to guide the inspection team in finding defects.

Defects found at either levels of the Analysis are recorded by type and severity. At the end of the Analysis step, a report which summarizes all of the defects discovered is produced.

### 5.2.1 Defect Classification

At the Structure Level, there exists two types of defect which can occur for a SRD. Figure 5.3 presents the list of these defect types.

A Relationship Defect (RD) signify a fault in one of the relations of the infoSchema. The infoSchema shown in Figure 5.2 contains a RD. It is missing a relationship, the one in column C of Figure 5.1.

RD : Relationship Defect
CD : Cardinality Defect

Figure 5.3 Defect Types for the Structure Level Analysis of an SRD.

A Cardinality Defect (CD) appears when the number of members of a set is suspicious. For example, if the set { PRE-CONDITION } has only one member, a CD is recorded. Figure 5.4 illustrates another instance of a CD. The cardinality of the set { DEFINITION } in column H is different from the sum of the cardinality of the sets { EXTERNAL OBJECT }, { INTERNAL OBJECT }, and { USER TYPE }. The meaning associated to this particular defect is that there exist terms referring to objects or user types which are used in the SRD but not defined.

| | A | H | I | J |
|---|---|---|---|---|
| 1 | A | 1 | { IDENTIFICATION } | |
| 2 | v | | | 1. Partial infoSchema for the HyperDoc SRD. |
| 3 | A | 8 | {REFERENCE} | |
| 4 | A | 6 | { VIEW } | |
| 5 | v | | | 1. Definitions |
| 10 | . | 4 | | 6. Set Cardinality |
| 11 | O | 29 | *{ DEFINITION }* | |
| 12 | T | 6 | *{ USER TYPE }* | |
| 14 | . | 1 | *{ EXTERNAL OBJECT }* | |
| 15 | T | 45 | *{ INTERNAL OBJECT }* | |

Figure 5.4 A Cardinality Defect for the HyperDoc SRD.

The number of defect types for the View Level is obviously larger. As shown in Figure 5.5, each defect type is associated with an infoStructure notation.

A Definition Defect (DD) appears in the Definitions view of the infoMap of a SRD. An example of a Definition Defect is highlighted in Figure 5.6 for the HyperDoc SRD. Here, more than one term is used to identify the same user type. The user type User leader is defined in column B but never referred to, while class leader is used throughout the SRD but not defined. The Definition Defect is associated with the letter "T" of the infoStructure notation.

| Defect Type | infoStructure Notation |
|---|---|
| DD: Definition Defect | T |
| AD: Assignment Defect | M |
| SD: Sequence Defect | S |
| HD: Hierarchy Defect | H |
| GD: Guard Defect | G |
| FD: Flow Defect | F |
| LD: Link Defect | L |
| TD: Typing Defect | - |

Figure 5.5 Defect Types for the View Level Analysis of an SRD.

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | 1 | | { IDENTIFICATION } |
| 2 | v | v | v | v | v | | | 1. Partial infoMap for the HyperDoc SRD. |
| 3 | A | A | A | A | A | 8 | | { REFERENCE } |
| 4 | A | A | A | A | A | 6 | | { VIEW } |
| 5 | v | v | . | . | . | | | 1. Definitions |
| 6 | . | x | v | v | v | | | 5. Product Users |
| 7 | . | x | . | . | . | v | | 6. Set Cardinality |
| 8 | O | O | . | . | . | 29 | | { DEFINITION } |
| 9 | o | . | . | . | . | | | 25. A user with full rights at all times in HyperDoc system. |
| 10 | . | o | . | . | . | | | ***28. A supervisor defined for a user class having the*** |
| 11 | | | | | | | | ***authority to manage the rights of users in that user class.*** |
| 12 | T | T | O | O | O | 6 | | { USER TYPE } |
| 13 | v | . | o | . | . | | | 1. Super user |
| 14 | . | . | . | o | . | | | 2. Builder |
| 15 | . | . | . | . | o | | | 3. Consulter |
| 16 | . | v | . | . | . | | | ***4. User leader*** |
| 17 | . | . | . | . | . | | | ***5. Class leader*** |
| 18 | . | . | . | . | . | | | 6. Author |
| 19 | T | T | M | M | M | 45 | | { INTERNAL OBJECT } |
| 20 | . | . | v | v | v | | | 1. HyperDoc |

Figure 5.6 A Definition Defect for the HyperDoc SRD.

An Assignment Defect (AD) occurs when a task, an action, or a remark is badly assigned. Figure 5.7 illustrates an example of an Assignment Defect for the

HyperDoc SRD. In column B, the remark about the notation <user id, object id, access right set> does not apply to this part of the Assign_Object_And_Rights_To_Manager operation. This type of defect is associated with the letter "M" of the infoStructure notation.

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | A | A | 1 | | { IDENTIFICATION } |
| 2 | v | v | | | 1. Partial infoMap for the HyperDoc SRD. |
| 3 | A | A | 8 | | { REFERENCE } |
| 4 | A | A | 6 | | { VIEW } |
| 5 | v | v | | | 4. Functional Requirements |
| 6 | . | . | v | | 6. Set Cardinality |
| 7 | O | O | 12 | | { OPERATION } |
| 8 | o | o | | | ***9. Assign_Object_And_Rights_To_Manager*** |
| 9 | F | F | 45 | | { INTERNAL OBJECT } |
| 10 | . | i | | | 22. User class id |
| 11 | . | l | | | 25. User class list |
| 12 | i | . | | | 27. User id |
| 13 | i | . | | | 31. User list |
| 14 | b | b | | | 35. Manager-object relation list |
| 15 | i | f | | | 36. Access rights set (Read, Write, Delete, etc .) |
| 16 | G | G | 15 | | { PRE-CONDITION } |
| 17 | t | t | | | 2. The user performing the operation is the SUPER user. |
| 18 | . | t | | | 3. The user class id is in the user class list. |
| 19 | t | . | | | 7. The user id is in the user list |
| 20 | f | . | | | 11. The <user id, object id> relation exists in the manager-object relation list |
| 21 | . | f | | | 12 The <user class id, object id> relation exists in the manager-object relation list. |
| 22 | S | S | 19 | | { ACTION } |
| 23 | 2 | 2 | | | 2 Inform the SUPER user of the results of the operation |
| 24 | 1 | . | | | 13. Add the <user id, object id, access rights set> relation in the manager-object |
| 25 | | | | | relation list. |
| 26 | . | f | | | 14. Add the <user class id, object id, access rights set> relation in the |
| 27 | | | | | manager-object relation list |
| 28 | G | G | 8 | | { POST-CONDITION } |
| 29 | t | t | | | 7. New manager-object relation list |
| 30 | G | G | 27 | | { POST-GUARD } |
| 31 | t | . | | | 15. To make a user eligible to access an object with certain access privileges |
| 32 | . | t | | | 16. To make a user class eligible to access an object with certain access privileges |
| 33 | M | M | 5 | | { REMARK } |
| 34 | v | v | | | ***3. The notation <user id, object id, access rights set> associates a user*** |
| 35 | | | | | ***with an object in the way specified by the access rights set.*** |
| 36 | v | v | | | 4. Current requirements |

Figure 5.7 An Assignment Defect for the HyperDoc SRD.

Sequence Defects (SD) arise when the order of actions to be performed is faulty. For example, the sequence of actions of operation Update_User shown in column A of Figure 5.8 should include an additional action. The Super user must be informed of the results of the operation when he/she is the originator of the

operation. The letter "S" of the infoStructure notation is associated to the Sequence Defect.

| | A | B | C | D |
|---|---|---|---|---|
| 1 | A | 1 | | { IDENTIFICATION } |
| 2 | v | | | 1. Partial infoMap for the HyperDoc SRD. |
| 3 | A | 8 | | { REFERENCE } |
| 4 | A | 6 | | { VIEW } |
| 5 | v | | | 4. Functional Requirements |
| 6 | | v | | 6. Set Cardinality |
| 7 | O | 12 | | { OPERATION } |
| 8 | o | | | ***6. Update_User*** |
| 9 | F | 45 | | { INTERNAL OBJECT } |
| 10 | t | | | 25. User class list |
| 11 | t | | | 27. User id |
| 12 | +b | | | 28. User name |
| 13 | +b | | | 29. User password |
| 14 | +b | | | 30. User description |
| 15 | G | 15 | | { PRE-CONDITION } |
| 16 | +t | | | 2. The user performing the operation is the SUPER user. |
| 17 | t | | | 5. The new values are different than the old ones. |
| 18 | +t | | | 8. The user performing the operation is the User himself. |
| 19 | t | | | 15. The user id is in the user class list. |
| 20 | S | 19 | | { ACTION } |
| 21 | . | | | ***2. Inform the SUPER user of the results of the operation.*** |
| 22 | 2 | | | ***9. Inform the user of the results of the operation.*** |
| 23 | t | | | ***10. Replace the old name and/or password and/or description*** |
| 24 | | | | ***of user with new values.*** |
| 25 | G | 8 | | { POST-CONDITION } |
| 26 | t | | | 5. New modified user attributes. |
| 27 | G | 27 | | { POST-GUARD } |
| 28 | t | | | 12. To modify the name |
| 29 | M | 5 | | { REMARK } |
| 30 | v | | | 2. A user class id cannot be modified |
| 31 | v | | | 4. Current requirements |

Figure 5.8 A Sequenc Defect for the HyperDoc SRD.

A Hierarchy Defect (HD) is found in the first part of the **Product Perspective** view of a SRD. An erroneous object hierarchy is the source of an Hierarchy Defect. This situation is clearly shown in the infoMap for the HyperDoc SRD in appendix D. Some objects, such as **anchor**, are not part of the hierarchy structure. This type of defect is associated with the letter "H" of the infoStructure notation.

Guard Defects (GD) happen when conditions are not constructed properly. Conditions listed in the sets { PRE-GUARD }, { PRE-CONDITION }, { POST-CONDITION }, and { POST-GUARD } are subject to Guard Defects. Figure 5.9 shows two examples of Guard Defects for an operation from the HyperDoc SRD. In column A, the pre-condition The user id is not in the user class list for the Update_User operation is inconsistent with the rest of the SRD. The user class list

| | A | B | C | D |
|---|---|---|---|---|
| 1 | A | 1 | { IDENTIFICATION } | |
| 2 | v | | | 1. Partial infoMap for the HyperDoc SRD. |
| 3 | A | 8 | { REFERENCE } | |
| 4 | A | 6 | { VIEW } | |
| 5 | v | | | 4. Functional Requirements |
| 6 | . | v | | 6. Set Cardinality |
| 7 | O | 12 | { OPERATION } | |
| 8 | o | | | ***6. Update_User*** |
| 9 | F | 45 | { INTERNAL OBJECT } | |
| 10 | i | | | 25. User class list |
| 11 | i | | | 27. User id |
| 12 | +b | | | 28. User name |
| 13 | +b | | | 29. User password |
| 14 | +b | | | 30. User description |
| 15 | G | 15 | { PRE-CONDITION } | |
| 16 | +t | | | 2. The user performing the operation is the SUPER user. |
| 17 | t | | | ***5. The new values are different than the old ones.*** |
| 18 | +t | | | 8. The user performing the operation is the User himself. |
| 19 | t | | | ***15. The user id is in the user class list.*** |
| 20 | S | 19 | { ACTION } | |
| 21 | . | | | 2. Inform the SUPER user of the results of the operation. |
| 22 | 2 | | | 9. Inform the user of the results of the operation. |
| 23 | 1 | | | 10. Replace the old name and/or password and/or description |
| 24 | . | | | of user with new values. |
| 25 | G | 8 | { POST-CONDITION } | |
| 26 | t | | | 5. New modified user attributes. |
| 27 | G | 27 | { POST-GUARD } | |
| 28 | t | | | 12. To modify the name |
| 29 | M | 5 | { REMARK } | |
| 30 | v | | | 2. A user class id cannot be modified |
| 31 | v | | | 4. Current requirements |

Figure 5.9 A Guard Defect for the HyperDoc SRD.

contains user class ids, not user ids. Moreover, the pre-condition **The new values are different than the old ones** for the same operation, should be true not false. Guard Defects are associated with the letter "G" of the infoStructure notation.

Faults about the flow of objects are called Flow Defects (FD). For instance, Figure 5.10 displays a data Flow Defect for the **Update_User** operation. In column A, the object **User class list** listed as an input for the operation is faulty. The **User list** is the correct input. Flow Defects are associated with the letter "F" of the infoStructure notation.

A Link Defect (LD) arises in a state diagram. A typical Link Defect occurs to a transition which leaves the system in an improper state. Unfortunately, the HyperDoc SRD does not show any Link Defects. This type of defect is associated with the letter "L" of the infoStructure notation.

The last type of defect is the Typing Defect (TD). Typing Defects simply relates to the faults made on the spelling of words in a SRD. Typing Defects should be reported only on key words of the SRD, such as objects.

### 5.2.2 Defect Detection Strategies

Detection strategies may be categorized in a number of ways. Figures 5.11 and 5.12 present a categorization of detection strategies for both levels of Analysis. This classification was previously introduced in chapter 3.

Both analysis techniques listed in Figure 5.11 are used at the Structure Level only. The Structure Analysis is a verification of the role of each set into every relationship of the infoSchema. Typical defects encountered through the Structure Analysis are:

a. a wrong dominant set for a relationship;

b. the letter "M" used where a more specific notation can be employed;

| | A | B | C | D |
|---|---|---|---|---|
| 1 | A | 1 | | { IDENTIFICATION } |
| 2 | v | | | 1. Partial infoMap for the HyperDoc SRD. |
| 3 | A | 8 | | { REFERENCE } |
| 4 | A | 6 | | { VIEW } |
| 5 | v | | | 4. Functional Requirements |
| 6 | | v | | 6. Set Cardinality |
| 7 | O | 12 | | { OPERATION } |
| 8 | a | | | ***6. Update_User*** |
| 9 | F | 45 | | { INTERNAL OBJECT } |
| 10 | f | | | ***25. User class list*** |
| 11 | f | | | 27. User id |
| 12 | +b | | | 28. User name |
| 13 | +b | | | 29. User password |
| 14 | +b | | | 30. User description |
| 15 | | | | ***31. User list*** |
| 16 | G | 15 | | { PRE-CONDITION } |
| 17 | +t | | | 2. The user performing the operation is the SUPER user. |
| 18 | t | | | 5. The new values are different than the old ones. |
| 19 | +t | | | 8. The user performing the operation is the User himself. |
| 20 | t | | | 15. The user id is in the user class list. |
| 21 | S | 19 | | { ACTION } |
| 22 | | | | 2. Inform the SUPER user of the results of the operation. |
| 23 | 2 | | | 9. Inform the user of the results of the operation. |
| 24 | t | | | 10. Replace the old name and/or password and/or description |
| 25 | | | | of user with new values. |
| 26 | G | 8 | | { POST-CONDITION } |
| 27 | t | | | 5. New modified user attributes. |
| 28 | G | 27 | | { POST-GUARD } |
| 29 | t | | | 12. To modify the name |
| 30 | M | 5 | | { REMARK } |
| 31 | v | | | 2. A user class id cannot be modified |
| 32 | v | | | 4. Current requirements |

Figure 5.10 A Flow Defect for the HyperDoc SRD.

c. a necessary set which is not part of a particular relationship; and

d. a necessary relationship which is not part of the infoSchema.

Defects detected during the Structure Analysis are recorded as Relationship Defects. They usually are major defects.

The Cardinality Analysis utilizes the **Set Cardinality** view of the infoSchema to detect defects. The Cardinality Defect of Figure 5.4 is easily located by comparing the cardinality of the set { DEFINITION } with the other sets part of the **Definitions** view. Similarly, a cardinality of zero for both { POST-GUARD } and { PRE-CONDITION } sets substantiates a major CD.

1. Cardinality Analysis
2. Structure Analysis

**Figure 5.11** Defect Detection Strategies for the Structure Level.

Figure 5.12 shows the analysis techniques utilized at the Structure Level and at View Level. The Redundancy Analysis searches for redundancy occurrences in the infoSchema or the infoMap. Analogous relationships in the infoSchema should be further investigated. Similarities among set members in the infoMap may also induce defects. An example is shown in Figure 5.6 for a Definition Defect. Both minor and major defects can be found from Redundancy Analysis.

The Consistency Analysis looks for inconsistencies within an infoStructure element. This technique requires more attention from the inspection team. Inconsistency may emerge from a set, a relationship, a view, or a combination of these. Figure 5.9 shows an example of a Guard Defect introduced from inconsistency between objects. Inconsistency can lead to both minor and major defects.

3. Redundancy Analysis
4. Consistency Analysis
5. Flow Analysis
6. Intention Analysis

**Figure 5.12** Defect Detection Strategies for Both Analysis Levels.

The Flow Analysis is intended to detecting defects related to the flow of objects. The method involves an examination of each object to verify where it is employed as an input and/or output. Defects are detected when, for instance, an object is referred to (input) but not initially declared (output). A Flow Analysis can discover both major and minor defects.

The last strategy for finding defects is the Intention Analysis. It consists of checking each relationship against the intentions of the author of the SRD. Obviously, both the author and the intended user are the key players in this technique. Their presence is of prime importance.

The application of all six detection strategies by the inspection team is recommended. As seen previously, each technique leads to different types of defects, and/or to different defects of the same type.

Software Specification Documents (SSDs) and Software Design Documents (SDDs) would be inspected using the same approach as for the SRD.

# Chapter 6
# Code Inspection

This chapter is the second stage of the validation of the proposed methodology by demonstration. In the previous chapter, the inspection method introduced in chapter 3 is applied on a SRD. Here, the same inspection process is performed on a Software Code Document (SCD).

The Software Code Document (SCD) is the principal deliverable of the Design phase of the software development process [48], [55]. Most reports on software inspections deal almost exclusively with code inspections [75], [76], [77], [79], [80], [81], [82], [83], [84], [85], [86], [87], [88], [89]. Although a few authors believe in the feasibility of inspection for different documents, results concerning other deliverables are nonexistent. Therefore, it was felt important to cover code inspection in this report.

Entry criteria for the inspection of code are reported in [80], [83], [87]. For the proposed inspection method, the entry criteria consists of the following conditions:

a. a clean compile of the code must be completed;

b. all changes reported in previous inspections must be reflected in the code; and

c. requirements, design and change documentation have to be part of the inspection package.

As for the Requirements phase of the software development process, the exit criteria for code inspection is that all defects found during the inspection process are corrected and verified.

The inspection team consists of four persons: the individual responsible for the design of the inspected product, the author of the Software Code Document,

the programmer responsible for the testing, and an inspector [82], [83], [87], [89]. It is recommended that the inspector be a programmer with experience in similar projects. However, the inspector should not be involved in the project being inspected.

The remainder of this chapter is organized in the following way. The first section explains the Normalization process for a Software Code Document. The Analysis process is covered in the next section. Both processes utilize the SCD of the program **convert** shown in appendix E as an example. Program **convert** reads from the input medium until it finds a digit, reads more digits until it reaches the end of the number, and performs the conversion to an internal form [51]. As with the requirements inspection, the Planning, Overview, Rework, and Follow-up steps are not covered in this chapter.

## 6.1 SCD Normalization

The Normalization of the SCD is the responsibility of the inspector. The infoFrame for a SCD is shown in Figure 6.1. It is in harmony with the research done in [1]. All of the information contained in a conventional Software Code Document can be captured using five views and 12 sets.

### 6.1.1 Sets

The sets for the SCD infoFrame are listed in column I of Figure 6.1. As for the SRD infoFrame, the { IDENTIFICATION } and { REFERENCE } sets are used respectively to identify the infoStructure element, and the reference documents.

The { OBJECT/ALGORITHM } set contains the name of all programs of the SCD. This is necessary for complex systems which include a number of programs. Sub-routine, procedure, and function names form the set { PROCEDURE }. Variable types are the members of the set { TYPE }. Examples are BOOLEAN and INTEGER for the Pascal programming language. The set { ATTRIBUTE } is made of the variables names declared in the SCD.

| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | A | A | 1 | { IDENTIFICATION } | |
| 2 | v | v | v | v | v | v | v | | | 1. SCD infoFrame. |
| 3 | A | A | A | A | A | A | A | X | { REFERENCE } | |
| 4 | A | A | A | A | A | A | A | 5 | { VIEW } | |
| 5 | v | . | . | . | . | . | . | | | 1. Hierarchy |
| 6 | . | v | v | v | . | . | . | | | 2. Data Model |
| 7 | . | . | . | . | v | v | . | | | 3. Data Flow |
| 8 | . | . | . | . | . | . | v | | | 4. Control Flow |
| 9 | . | . | . | . | . | . | . | v | | 5. Set Cardinality |
| 10 | H | O | . | . | O | . | . | X | { OBJECT/ALGORITHM } | |
| 11 | . | . | . | O | . | . | . | X | { TYPE } | |
| 12 | . | M | M | M | F | O | . | X | { ATTRIBUTE } | |
| 13 | . | . | . | . | . | . | O | X | { TRANSITION } | |
| 14 | . | . | . | . | . | . | L | X | { STATE } | |
| 15 | . | . | . | . | . | F | G | X | { PRE-CONDITION } | |
| 16 | H | . | O | . | . | F | S | X | { PROCEDURE } | |
| 17 | . | . | . | . | . | F | S | X | { ACTION } | |
| 18 | . | . | . | . | . | F | G | X | { POST-CONDITION } | |

Figure 6.1 The SCD infoFrame.

The Boolean test statements found in the SCD are grouped into two categories. The set { PRE-CONDITION } encloses the test statements executed prior to the actions it governs. For instance, test statements part of an IF-THEN-ELSE code construct are members of this set. The second category of test statements are those which are executed after the actions they control. These test statements form the set { POST-CONDITION }. The test statements found in the REPEAT-UNTIL code construct is an example.

The members of the set { ACTION } are those statements of a SCD which require a specific action to be performed. The most common statement which is part of this set is the assignment statement.

As opposed to the SRD, the members of the sets { TRANSITION } and { STATE } are easily detected. They represent respectively the transitions and states of the process described by a SCD. Both set members are dictated by the sequence of code statements of a SCD. An identification number is associated with each

member of both sets. The inspector may extend each set member with a brief description to better illustrate the meaning of the state/transition.

### 6.1.2 Views

The first view, **Hierarchy**, illustrates the hierarchy among the programs and the procedures/functions/sub-routines. It does not represent the order in which the modules are declared, but rather shows how the modules "call" each other.

The **Data Model** view is made of three different relations. The first two relations list the variables which are declared in each program or procedure/function/sub-routine. The third view associates a type to each variable found in a SCD.

The **Data Flow** view consists of two relations. The flow of variables in and out of each program is shown in the first relation. The second relation exposes the statements of a program where each variable is referenced and/or defined. These statements may be conditional, procedural, or imperative [1].

The one relation forming the **Control Flow** illustrates a state machine or state diagram for the SCD. Each transition is associated with:

a. a departure state and a termination state.

b. a number of pre-conditions which must be fulfilled prior to the execution of the transition;

c. a sequential list of procedure calls and actions to be performed during the transition; and

d. a number of post-conditions which must be met before the transition is allowed to conclude.

Finally, the Set Cardinality set reports the number of members in each set of the infoStructure element.

### 6.1.3 Results

The Normalization step concludes when an infoSchema and an infoMap of the SCD to be inspected are completed. Both infoStructure elements are necessary for the Analysis step. An example of an infoSchema for a Pascal SCD is displayed in Figure 6.2. Appendix F shows the corresponding infoMap.

| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | A | A | 1 | { IDENTIFICATION } | |
| 2 | v | v | v | v | v | v | v | | | 1. infoSchema for the "convert" Program SCD. |
| 3 | A | A | A | A | A | A | A | 1 | {REFERENCE} | |
| 4 | A | A | A | A | A | A | A | 5 | { VIEW } | |
| 5 | v | . | . | . | . | . | . | | | 1. Hierarchy |
| 6 | . | v | v | v | . | . | . | | | 2. Data Model |
| 7 | . | . | . | . | v | v | . | | | 3. Data Flow |
| 8 | . | . | . | . | . | . | v | | | 4. Control Flow |
| 9 | . | . | . | . | . | . | . | v | | 5. Set Cardinality |
| 10 | H | O | . | . | O | . | . | 1 | {OBJECT/ALGORITHM} | |
| 11 | . | . | . | O | . | . | . | 5 | { TYPE } | |
| 12 | . | M | M | M | F | O | . | 15 | { ATTRIBUTE } | |
| 13 | . | . | . | . | . | . | O | 10 | { TRANSITION } | |
| 14 | . | . | . | . | . | . | L | 7 | { STATE } | |
| 15 | . | . | . | . | . | F | G | 4 | { PRE-CONDITION } | |
| 16 | H | . | O | . | . | F | S | 4 | {PROCEDURE} | |
| 17 | . | . | . | . | . | F | S | 17 | { ACTION } | |
| 18 | . | . | . | . | . | F | G | 2 | { POST-CONDITION } | |

Figure 6.2 The infoSchema for the convert Program SCD.

## 6.2 SCD Analysis

As for the Requirements inspection, the whole inspection team participates in the Analysis step. The inspector takes control of the Analysis process. The Structure Level analysis comes first. The infoSchema is scrutinized with the aim of finding defects. A list of common defect types, as shown in Figure 6.3, guides the inspection team. The discovery of a major defect results in a halt of the inspection process. In this case, the deliverable is directed to the Rework step.

When no major defect is found at the Structure Level analysis, the inspection team carries on with the View Level analysis. Each view of the infoMap is displayed to all team members. Figure 6.4 lists the common defect types which are employed by the inspection team to detect defects.

The same recording procedure applies to the code inspection. The type and severity of each defect found are registered. A summary report is produced upon conclusion of the Analysis step.

### 6.2.1 Defect Classification

The two types of defect identified for the Requirements inspection at the Structure Level also apply to the Code inspection. These two defect types are repeated in Figure 6.3.

There is no evidence of a Relationship Defect (RD) nor a Cardinality Defect (CD) in the infoSchema of Figure 6.2. An example of a CD for an infoMap representing some source code, is the cardinality of the set { STATE } greater than the cardinality of the set { TRANSITION } plus one. The meaning associated to this particular defect is that there exists at least one state which is never reached.

RD : Relationship Defect
CD : Cardinality Defect

**Figure 6.3** Defect Types for the Structure Level Analysis of an SCD.

The defect types for the View Level slightly differs from those for the SRD inspection. Only the Definition Defect (DD) was removed. The relation between each defect type and the infoStructure notation is illustrated in Figure 6.4.

An Assignment Defect (AD) occurs when a attribute is badly assigned to a program, a procedure or function. An erroneous type associated to a attribute also causes an AD. No instance of an Assignment Defect is detected in the infoMap of the program **convert**. The Assignment Defect is associated with the letter "M" of

the infoStructure notation.

| Defect Type | infoStructure Notation |
|---|---|
| AD: Assignment Defect | M |
| SD: Sequence Defect | S |
| HD: Hierarchy Defect | H |
| GD: Guard Defect | G |
| FD: Flow Defect | F |
| LD: Link Defect | L |
| TD: Typing Defect | - |

**Figure 6.4** Defect Types for the View Level Analysis of an SCD.

Sequence Defects (SD) develop in the **Control Flow** view. A SD results from a fault in the order of the actions and procedures to be performed when a transition takes place. The infoMap shown in appendix F displays no indication of Sequence Defects. The letter "S" of the infoStructure notation is associated to the Sequence Defect.

Hierarchy Defects (HD) appears in the **Hierarchy** view. A HD is detected when a program or a procedure "calls" erroneously a sub-routine, function, or procedure. Unfortunately, no HD was identified for the Pascal program **convert**. This type of defect is associated with the letter "H" of the infoStructure notation.

Guard Defects (GD) are all contained in the **Control Flow** view. A GD arises when a test statement is not formulated properly. Members of the set { PRE-CONDITION } and { POST-CONDITION } are all subject to Guard Defects. Figure 6.5 illustrates an example of a Guard Defect for transition **2** within the program **convert**. The details of this transition is displayed in column AC. The post-conditions **zero** <= **ch** and **ch** <= **nine** are insufficient to cover all input cases. For instance, if the number .84 is entered, the decimal point is skipped and the first character considered is the number eight. As a result, program **convert** treats .84 as 84. This observation is valid for all inputs of the form ".X", where X is one or a series of digits. Guard Defects are associated with the letter "G" of the infoStructure notation.

Faults concerning the flow of attributes are known as Flow Defects (FD). This type of defect can be found in both relations of the **Data Flow** view. A

| | AB | AC | AD | AE | AF | AG | AH | AI | AJ | AK | AL | AM | AN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | A | A | A | A | A | 1 | | { IDENTIFICATION } |
| 2 | v | v | v | v | v | v | v | v | v | v | | | 1. Partial infoMap for the "convert" Program SCD. |
| 3 | A | A | A | A | A | A | A | A | A | A | 1 | | { REFERENCE } |
| 5 | A | A | A | A | A | A | A | A | A | A | 5 | | { VIEW } |
| 9 | v | v | v | v | v | v | v | v | v | v | | | 4. Control Flow |
| 10 | . | | . | . | . | . | . | . | . | . | v | | 5. Set Cardinality |
| 35 | O | O | O | O | O | O | O | O | O | O | 10 | | { TRANSITION } |
| 36 | o | | . | . | . | . | . | . | . | . | | | 1 |
| 37 | . | o | . | . | . | . | . | . | . | . | | | 2 |
| 38 | . | | o | . | . | . | . | . | . | . | | | 3 |
| 39 | . | | . | o | . | . | . | . | . | . | | | 4 |
| 40 | . | | . | . | o | . | . | . | . | . | | | 5 |
| 41 | . | | . | . | . | o | . | . | . | . | | | 6 |
| 42 | . | | . | . | . | . | o | . | . | . | | | 7 |
| 43 | . | | . | . | . | . | . | o | . | . | | | 8 |
| 44 | . | | . | . | . | . | . | . | o | . | | | 9 |
| 45 | . | | . | . | . | . | . | . | . | o | | | 10 |
| 46 | L | L | L | L | L | L | L | L | L | L | 7 | | { STATE } |
| 47 | s | | . | . | . | . | . | . | . | . | | | 1. start convert |
| 48 | d | l | . | . | . | . | . | . | . | . | | | 2 |
| 49 | . | e | l | s | . | . | . | . | . | . | | | 3 |
| 50 | . | | . | d | s | s | . | . | . | . | | | 4 |
| 51 | . | | . | . | . | d | l | s | . | . | | | 5 |
| 52 | . | | . | . | . | . | . | d | l | s | | | 6 |
| 53 | . | | . | . | d | . | . | . | . | d | | | 7. end convert |
| 54 | G | G | G | G | G | G | G | G | G | G | 4 | | { PRE-CONDITION } |
| 55 | . | | t | +f | . | . | t | +f | . | . | | | 1. zero <= ch |
| 56 | . | | t | +f | . | . | t | +f | . | . | | | 2. ch <= nine |
| 57 | . | | . | . | f | t | . | . | . | . | | | 3. ch = point |
| 58 | . | | . | . | . | . | . | . | t | f | | | 4. scale > 0 |
| 59 | S | S | S | S | S | S | S | S | S | S | 4 | | { PROCEDURE } |
| 60 | . | | 2 | . | . | . | 2 | . | . | . | | | 1. ord |
| 61 | . | | 5 | . | 2 | . | 5 | . | . | . | | | 2. ord |
| 62 | 7 | 7 | 8 | . | . | 3 | 8 | . | . | . | | | 3. read |
| 63 | . | | . | . | . | . | . | . | . | 2 | | | 4. writeln |
| 64 | S | S | S | S | S | S | S | S | S | S | 17 | | { ACTION } |
| 65 | 1 | | . | . | . | . | . | . | . | . | | | 1. zero = '0'; |
| 66 | 2 | | . | . | . | . | . | . | . | . | | | 2. nine = '9'; |
| 67 | 3 | | . | . | . | . | . | . | . | . | | | 3. point = '.'; |
| 68 | 4 | | . | . | . | . | . | . | . | . | | | 4. radix = '10'; |
| 69 | 5 | | . | . | . | . | . | . | . | . | | | 5. result := 0; |
| 70 | . | | 1 | . | . | . | 1 | . | . | . | | | 6. DummyOrdIn := ch; |
| 71 | . | | 3 | . | . | . | 3 | . | . | . | | | 7. result := radix * result + DummyOrdOut, |
| 72 | . | | 4 | . | . | . | 4 | . | . | . | | | 8. DummyOrdIn := zero, |
| 73 | . | | 6 | . | . | . | 6 | . | . | . | | | 9 result := result - DummyOrdOut; |
| 74 | . | | . | . | . | 1 | . | . | . | . | | | 10. scale := 0; |
| 75 | 6 | 1 | 7 | . | | 2 | 7 | . | . | . | | | 11. DummyFileRead := input; |
| 76 | 8 | 3 | 9 | . | . | 4 | 9 | . | . | . | | | 12. ch := DummyRead, |
| 77 | . | | . | . | . | | 10 | . | . | . | | | 13. scale := scale + 1, |
| 78 | . | | . | . | | . | . | . | 1 | . | | | 14. result := result /radix; |
| 79 | . | | . | . | | . | . | . | 2 | | | | 15 scale := scale - 1, |
| 80 | . | | . | . | 1 | . | | . | . | 1 | | | 16 DummyWriteln := result, |
| 81 | . | | . | . | 3 | . | . | . | . | 3 | | | 17 output := DummyFileWrite; |
| 82 | G | G | G | G | G | G | G | G | G | G | 2 | | { POST-CONDITION } |
| 83 | . | +f | . | . | . | . | . | . | . | . | | | *1. zero <= ch* |
| 84 | . | +f | . | . | . | . | . | . | . | . | | | *2. ch <= nine* |

Figure 6.5 A Guard Defect for the convert Program.

typical FD is a faulty input attribute for a procedure. The program **convert** does not reveal any FD. Flow Defects are associated with the letter "F" of the infoStructure notation.

Link Defects (LD) take place in the **Control Flow** view. Link Defects are caused by a faulty combination of transitions and states in a state machine or state diagram. A typical Link Defect occurs when a state is declared but never reached. Unfortunately, the infoMap of the program **convert** does not disclose any Link Defects. The Link Defect is associated with the letter "L" of the infoStructure notation.

Finally, Typing Defects (TD) are concerned with the syntax of the code. Although one of the entry criteria specifies that the program must go through a clean compile before inspection, there still are certain syntactical aspects which remain to be inspected. These include the standard imposed by the organization for the procedure and variable names. For example, the attribute name **ch** as shown in Figure 6.6 might not meet the standard fixed by a particular organization. It would then be considered as a TD.

### 6.2.2 Defect Detection Strategies

As stated in chapter 3, the six detection strategies identified in the previous chapter also apply to a Software Code Document. Figures 6.7 and 6.8 list the methods. Since each technique was fully described in chapter 5, this section is aimed at providing specific examples relative to a SCD.

Defects detected through the Structure Analysis are common to the infoSchema of all types of deliverables. However, particular defects for a SCD exist for the Cardinality Analysis. The example of a state that is declared but never reached, as stated previously in this chapter, is an instance of defect found through the Cardinality Analysis.

| | G | H | I | J | K | AL | AM | AN |
|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | 1 | { IDENTIFICATION } | |
| 2 | v | v | v | v | v | | | 1. Partial infoMap for the "convert" Program SCD. |
| 3 | A | A | A | A | A | 1 | {REFERENCE} | |
| 5 | A | A | A | A | A | 5 | { VIEW } | |
| 7 | v | v | v | v | v | | | 2. Data Model |
| 10 | . | . | . | . | . | v | | 5. Set Cardinality |
| 13 | O | O | O | O | O | 5 | {TYPE} | |
| 14 | o | . | . | . | . | | | 1. real |
| 15 | . | o | . | . | . | | | 2. integer |
| 16 | . | . | o | . | . | | | 3. char |
| 17 | . | . | . | o | . | | | 4. list of char strings |
| 18 | . | . | . | . | o | | | 5. file of char |
| 19 | M | M | M | M | M | 15 | { ATTRIBUTE } | |
| 20 | . | . | v | . | . | | | 1. zero |
| 21 | . | . | v | . | . | | | 2. nine |
| 22 | . | . | v | . | . | | | 3. point |
| 23 | . | v | . | . | . | | | 4. radix |
| 24 | v | . | . | . | . | | | 5. result |
| 25 | . | v | . | . | . | | | 6. scale |
| 26 | . | . | v | . | . | | | *7. ch* |
| 27 | . | . | . | . | v | | | 8. input |
| 28 | . | . | . | . | v | | | 9. output |
| 29 | . | . | . | v | . | | | 10. DummyRead |
| 30 | . | . | v | . | . | | | 11. DummyOrdIn |
| 31 | . | . | . | v | . | | | 12. DummyWriteIn |
| 32 | . | v | . | . | . | | | 13. DummyOrdOut |
| 33 | . | . | . | . | v | | | 14. DummyFileRead |
| 34 | . | . | . | . | v | | | 15. DummyFileWrite |

Figure 6.6 A Typing Defect for the **convert** Program.

Figure 6.8 lists the four defect detection strategies which are applicable at the View Level. Here, the attention is drawn on the Flow Analysis. An example of a defect found by using this technique is the case where an attribute is referred to but never defined. This defect can be easily detected in the second relation of the Data Flow view. Each attribute identified in the set

3. Redundancy Analysis

4. Consistency Analysis

5. Flow Analysis

6. Intention Analysis

Figure 6.7 Defect Detection Strategies for Both Analysis Levels.

{ ATTRIBUTE } must be utilized as an input and an output at least once in one of the sets { PRE-CONDITION }, { PROCEDURE }, { ACTION }, or { POST_CONDITION }. A simpler instance of a defect detected through a Flow Analysis is an attribute which is never referred to nor defined.

1. Cardinality Analysis
2. Structure Analysis

**Figure 6.8** Defect Detection Strategies for the Structure Level.

# Chapter 7
# Results Analysis

As with the traditional inspection method, the proposed inspection technique is composed of six steps. They have been identified and described in chapter 3. However, there is still a number of elements that remains to be discussed.

The remainder of this chapter examines the three main areas where the proposed inspection method differs from the traditional inspection technique. They are the members of the inspection team, the Normalization process, and the Analysis process. Thereafter, a brief discussion on the inspection technique as a whole concludes the chapter.

## 7.1 Evaluating the Proposed Inspection Method

### 7.1.1 Participants

The members of the inspection team are chosen in the Planning step. Although the proposed inspection method does not consider the moderator role as being necessary to the inspection process, it is agreed that someone must exercise some control for each of the inspection steps. As discussed in chapter 3, the author is responsible for the Overview and the Rework steps, while the inspector handles the Normalization, Analysis, and Follow-up steps. Planning is performed by the management group of the organization.

The inspector role is played by a single person. The author is also represented by a unique individual. Therefore, an inspection team is composed of only two persons. It reduces the personnel and logistic needs required for the inspection process to a minimum [77]. The removal of the author and/or the inspector role would obviously result in a complete elimination of the whole inspection process.

### 7.1.2 Normalization

The Normalization step replaces the Preparation step of the traditional inspection method. The deliverable to be inspected is transformed into infoStructure elements. The infoFrame helps the inspector in determining precisely what should go into the deliverable [68]. The infoSchema presents the structural content of the deliverable, while the infoMap illustrates all the details. Therefore, the infoStructure elements form a hierarchy with respect to the depth of information they embody. This hierarchy allows the analysis to be carried out at different levels, which facilitates the inspection process.

In [65], the Entity-Relationship (ER) model is used to represent an application design. The ER model easily pictures simple relationships like a one-to-one, one-to-many, or hierarchy relation. However, there exist difficulties in representing complex relationships such as a complicated hierarchy or a flow relation. The infoStructure element deals efficiently with complex relationships.

The infoFrame for the SRD presented in chapter 5 was built from two different SRDs. The HyperDoc SRD was normalized first. Its infoSchema was used as an infoFrame to normalize the second SRD, the SRD for the New Editor (NED). This SRD is shown in appendix G. Its infoSchema, which appears in Figure 7.1, was utilized to update the SRD infoFrame.

The SCD infoFrame was deduced from past experience with the Normalization of a number of SCDs. Research done in [1] also influenced the composition of the SCD infoFrame.

The main drawback of the Normalization step consists in the possibility of it introducing defects. Any manual transformation of a deliverable suffers from the risk of modifying the content. This is mainly caused by human inadvertence or misinterpretation. As depicted in chapter 1, this is one of the primary problems of today's software development process. In our case, the problem can be minimized by expressing the entry criteria for the Requirements inspection up as

| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | A | A | 1 | { IDENTIFICATION } | |
| 2 | v | v | v | v | v | v | v | | | 1. infoSchema for the NED SRD. |
| 3 | A | A | A | A | A | A | A | 6 | { REFERENCE } | |
| 4 | A | A | A | A | A | A | A | 6 | { VIEW } | |
| 5 | v | . | . | . | . | . | . | | | 1. Definitions |
| 6 | . | v | . | . | . | . | . | | | 2. Product Perspective |
| 7 | . | . | v | v | . | . | . | | | 3. Product Functions |
| 8 | . | . | . | . | v | v | . | | | 4. Functional Requirements |
| 9 | . | . | . | . | . | . | v | | | 5. Product Users |
| 10 | . | . | . | . | . | . | . | v | | 6. Set Cardinality |
| 11 | O | . | . | . | . | . | . | 41 | { DEFINITION } | |
| 12 | . | . | . | . | . | . | O | 2 | { USER TYPE } | |
| 13 | . | . | . | . | H | O | . | 10 | { OPERATION } | |
| 14 | T | H | . | . | . | F | . | 11 | { EXTERNAL OBJECT } | |
| 15 | T | H | O | O | . | F | M | 42 | { INTERNAL OBJECT } | |
| 16 | . | . | G | . | . | . | G | 9 | { PRE-GUARD } | |
| 17 | . | . | . | . | . | G | . | 18 | { PRE-CONDITION } | |
| 18 | . | . | . | . | . | G | . | 1 | { EVENT } | |
| 19 | . | . | . | M | H | . | . | 18 | { TASK } | |
| 20 | . | . | . | . | . | S | . | 24 | { ACTION } | |
| 21 | . | . | . | . | . | G | . | 30 | { POST-CONDITION } | |
| 22 | . | . | G | . | . | G | . | 31 | { POST-GUARD } | |
| 23 | . | . | . | M | . | M | . | 6 | { REMARK } | |

Figure 7.1 The infoSchema for the NED SRD.

an infoStructure representation of the SRD or else, through the automation of the Normalization process.

### 7.1.3 Analysis

There exist four main analysis criteria: completeness, consistency, testability, and feasibility [65], [69]. The proposed inspection method, as the traditional inspection technique, deals principally with the completeness and consistency of a deliverable.

The infoStructure allows the inspection to be performed hierarchically. The proposed inspection method prescribes an analysis at the Structure Level and the View Level. The method can be augmented with additional levels. The sets in the

SRD infoFrame (Figure 5.1) reflect this hierarchy concept. The set { TASK } is at a higher level than the set { ACTION }. Similarly, the sets { PRE-GUARD } and { POST-GUARD } are one level above the sets { PRE-CONDITION } and { POST-CONDITION } respectively. This hierarchy in the analysis process lets the inspection team deal with a deliverable using a top-down approach. Consequently, defects detected at the top level should lead to major defects. This concept was expressed in chapters 5 and 6 for the analysis at the Structure Level.

To assist the inspection team in the Analysis process, defects are categorized. A list of defect types is necessary for each analysis level. At the Structure Level, the defect types were found to be the same for both the SRD and SCD (Figures 5.3 and 6.3). At the View Level, the defect types identified for the SCD (Figure 6.4) is a subset of those associated to the SRD (Figure 5.5). These observations indicate a possible generalization of the defect types, one which would be independent of the deliverable inspected.

The defect types of the SCD at the View Level (Figure 6.4) are comparable to the defect types identified by Myers [75]. Figure 7.2 establishes the relationship which exists between the defect types of the proposed inspection method and Myers'.

The grouping of defects elaborated in chapters 5 and 6 could be further develop to include additional defect types for supplementary analysis levels. For example, Flow Defects (FD) could be subdivided into two categories, Data Flow Defects (DFD) and Control Flow Defects (CFD).

| Deslauriers View Level | Myers [75] |
|---|---|
| FD | Data-Reference Errors |
| AD | Data-Declaration Errors |
| SD | Computation Errors |
| GD | Comparison Errors |
| FD/LD | Control-Flow Errors |
| FD/HD | Interface Errors |
| FD | Input/Output Errors |
| TD | Other Checks |

**Figure 7.2** A Comparison of Defect Types.

The defect detection strategies introduced in this report are equally applicable to both SRDs and SCDs. A set of strategies is associated to each analysis level. They provide the inspection team with practical techniques for finding defects. Each detection strategy serves in identifying specific types of defects.

The analysis of a number of deliverable of the same type illustrates some interesting features. An example is the analysis of the HyperDoc SRD and the NED SRD. Both SRDs were done by a group of students from Concordia University. The members of the two groups had basically the same background. Both SRDs were composed as an assignment for a Software Engineering course taught by the same professor. One could have expected the structure of these two SRDs to be similar. However, a Structure Analysis performed on each SRD shows that their infoSchemas are different in a number of ways. For example, the set { EVENT } shown in column J, row 18 of the infoSchema for the NED SRD (Figure 7.1) does not appear in the infoSchema for the HyperDoc SRD (Figure 5.2).

## 7.2 Discussion

The inspection method presented in this thesis benefits from the same advantages as the traditional inspection method. The detection of defects early in the software development process, in software deliverables such as an SRD, results in a significant reduction in the development cost, along with an increase of the total productivity. The inspection process gets continuously updated as inspections are carried out. The infoFrames are being revised as infoSchemas are produced. Moreover, the infoStructure elements allow a better visualisation of program structures and characteristics. This is also a property of the matrices used by Lichtman to analyze program structures [60].

A detailed inspection process was defined in chapter 5 for the first formal deliverable of the software development process, the Software Requirements Document. In chapter 6, an equivalent exercise led the reader through a code

inspection. During the other phases of the software development process, a number of intermediate deliverables, such as the Software Specification Document and Software Design Document, are produced. Each of these deliverables represents the final software product at a specific level of detail, and most of the time, under a different format. Since the proposed inspection method was proven valid for the SRD and SCD, it is also assumed valid for any deliverable in between. Therefore, the inspection process as described in this thesis can be applied to any software deliverable.

# Conclusion

This thesis has proposed an inspection methodology aimed at reaching higher software quality and reliability. The proposed inspection method is considered as an enhancement to Fagan's technique. While the traditional inspection method has been conventionally utilized with source code, the proposed inspection process is applicable to any software deliverable. The technique was demonstrated for two IEEE software documents, but the other deliverables are intermediate products of these two.

The infoStructure representation of the software deliverables improves the visual recognition of defects. Defect types and detection strategies used during the inspection process are directly derived from this representation.

A first attempt was made to formalize the Preparation and Inspection phases of the traditional inspection method. The Normalization step is a transformation process governed by explicit rules, while the Analysis step is built from specific defect types and detection strategies. Both processes are manipulating processable infoStructure elements representing a software deliverable. Therefore, the automation of these two processes is achievable, which could lead to a fully automated inspection process.

The work done in this thesis opens up a wide range of fields of interest for future research in the inspection process itself or else, in the software development process. These are:

a. the Normalization of a large number of SRDs to verify the proposed SRD infoFrame;

b. the Normalization of other software deliverables to establish their corresponding infoFrames;

c. the generalization of the defect types over all software deliverables;

d. the construct of a View Generation tool to assist the inspection team in visualising the different views of the infoStructure representation of the software

deliverables.

Such a tool would take a user defined area of an infoStructure element as an input, and output a predefined form of the selected area. Instances of these forms are diagrams (control flow diagram, data flow diagram, state diagram, etc...), or a whole software deliverable (SRD, SSD, SCD, etc...). A View Generation tool is particularly useful for the Overview and Rework phase of the inspection process;

e. the automation of the proposed inspection method; and

f. the entry criteria defined in chapter 5 for the Requirements inspection could be loosened.

In this case, the Analysis step of the inspection process identifies the content which is missing from a partial SRD.

Rework can be done on the infoStructure elements to solve the detected defects. In fact, it is possible to build an SRD from the very beginning by using the infoStructure concept. The infoFrame serves as a template which leads the software engineer in the right direction. The infoMap is produced from the infoFrame by "filling the blanks". This technique for producing an SRD is applicable to the other software deliverables as well. Its consequences to the proposed inspection method is to eliminate the Normalization step. This would considerably speed up the whole inspection process, and completely remove the chance of inducing defects during Normalization.

# References

A. infoStructure


[1] T. Cummings, "A Knowledge Acquisition Method: Transformation of Algorithms and Programs with infoMaps (TAPi)", Master of Computer Science Thesis, Computer Science Department, Concordia University, Montreal, Quebec, Canada, May 1991.

[2] W.M. Jaworski, T. Cummings, "Program Normalization and Optimization: Using infoMaps as an Inspection and Program Processing Tool", Computer Science Department, Concordia University, Montreal, Quebec, Canada, 1991.

[3] W.M. Jaworski, T. Cummings, "Program Analysis, Processing and Optimization: Spreadsheet based infoMaps", Computer Science Department, Concordia University, Montreal, Quebec, Canada, January 1991.

[4] W.M. Jaworski, B. Deslauriers, "Software: Process, Maintenance, Products, infoMaps", Proceedings of the 3rd Canadian Conference on Electrical and Computer Engineering, Ottawa, Ontario, Canada, September 4-6, 1990.

[5] W.M. Jaworski, P.D. Grogono, "infoMaps: a Pragmatic Environment for Seamless and Nondeterministic Software Development", Computer Science Department, Concordia University, Montreal, Quebec, Canada, June 1990.

[6] W.M. Jaworski, "Systems Analysis and Design in the Classroom: infoMaps Factory", Proceedings of the Modeling and Simulation Conference, Pittsburgh, Pa., May 3-4, 1990.

[7] P.D. Grogono, W.M. Jaworski, "Software Development as Knowledge Acquisition using jMaps", Proceedings of the Canadian Conference on Electrical and Computer Engineering, Montreal, Quebec, Canada, September 17-20, 1989.

[8] W.M. Jaworski, E. Farell, "j-MAP Notational Technology: Application to Development of Spectrum Management Systems", Montech '87 Proceedings, Conference on Communications, Montreal, Quebec, Canada, Novemnber 9-11, 1987.

[9] W.M. Jaworski, T. Radhakrishnan, "Modelling of System Development Methodologies", CompInt '87 Proceedings, Conference on Computer-Aided Technologies, Montreal, Quebec, Canada, November 9-11, 1987.

[10] W.M. Jaworski, L. Ficocelli, K.S. O'Mara, "The ABL/W4 Metrhodology for System Modelling", Systems Research Journal, vol. 4, no.1, pp. 23-37, 1987.

[11] W.M. Jaworski, M. Virard, "Converting a Software Comapny to a New Technology", Proceedings of the 1986 Canadian Conference on Industrila Computer Systems, Montrel, Quebec, Canada, May 28-30, 1986.

[12] G. Belkin, W.M. Jaworski, "Towards Logic and Performance Analysis of Unwritten Programs", Proceedings of the Canadian Computer Conference, Montreal, Quebec, Canada, May 17, 1976.

B. Software process

[13] F.J. Buckley, "Do standards cause software productivity problems?", Computer, vol. 24, no. 1, pp. 97-98, January 1991.

[14] S.P. Overmyer, "The Impact of DoD-Std-2167A on Iterative Design Methodologies: Help or Hinder?", ACM SIGSOFT Software Engineering Notes, vol. 15 no. 5, pp. 50-59, October 1990.

[15] E. Souza, "The New CASE Development Life Cycle", Software Engineering: Tools, Techniques, Practice, vol. 1, no. 3, pp. 14-21, September/October 1990.

[16] P. Freeman, "Establishing a System for Developing Software", Software Engineering: Tools, Techniques, Practice, vol. 1, no. 2, pp. 14-22, July/August 1990.

[17] R.W. Matthews, W.C. McGee, "Data modeling for software development", IBM Systems Journal, vol. 29, no. 2, pp. 228-235, 1990.

[18] J.A. Hager, "Software Cost Reduction Methods in Practice", IEEE Trans. on SE, vol. 15, no. 12, pp. 1638-1644, December 1989.

[19] W.S. Humphrey, "Improving the Software Development Process", Datamation, vol. 35, no. 7, pp. 28-30,52, April 1, 1989.

[20] E.J. Joyce, "Is Error-Free Software Achievable?", Datamation, vol. 35, no. 4, pp. 53-56, February 15, 1989.

[21] W.S. Humphrey, M.I. Kellner, "Software Process Modeling: Principles of Entity Process Models", Proc. 11th Conf. Software Eng., Pittsburgh, Pa., New-York: IEEE Press, pp. 331-342, 1989.

[22] A.M. Davis, E.H. Bersoff, E.R. Comer, "A Strategy for Comparing Alternative Software Development Life Cycle Models", IEEE Trans. on SE, vol. 14, no. 10, pp. 1453-1461, October 1988.

[23] W.B. Bohem, P.N. Papaccio, "Understanding and Controlling Software Costs", IEEE Trans. on SE, vol. 14, no. 10, pp. 1462-1477, October 1988.

[24] D.N. Card, F.E. MC Garry, G.T. Page, "Evaluating Software Engineering Technologies", IEEE Trans. on SE, vol. 13, no. 7, pp. 845-851, July 1987.

[25] S.S. Yau, J.J. Tsai, "Knowledge Representation of Software Component Interconnection Information for Large-Scale Software Models", IEEE Trans. on SE, vol. 13, no. 3, pp. 355-361, March 1987.

[26] B. Ratcliff, Software Engineering: Principles and Methods, Blackwell Scientific Publications, 1987.

[27] B.G. Silverman, "Software Cost and Productivity Improvements: An Analogical View", Computer, vol. 18, no.5, pp. 86-96, May 1985.

[28] W.S. Humphrey, "The IBM large-systems software development process: Objectives and direction", IBM Systems Journal, vol. 24, no. 2, pp. 76-78, 1985.

[29] G.F. Hoffnagle, W.E. Beregi, "Automating the software development process", IBM Systems Journal, vol. 24, no. 2, pp. 102-120, 1985.

[30] L.L. Beck, T.E. Perkins, "A Survey of Software Engineering Practice: Tools, Methods, and Results", IEEE Trans. on SE, vol. 9, no. 3, pp. 541-561, September 1983.

[31] U. Montanari, "Towards an Integration Between Language and Software Development Environment", Theory and Practice of Software Technology, North-Holland Publishing Company, 1983, pp. 195-202.

[32] N. Zvegintzov, "What life? What cycle?", AFIPS Conference Proceedings, vol 51, pp. 561-568, 1982.

[33] B.W. Boehm, "Software and Its Impact: A Quantitative Assessment", Datamation, pp. 48-59, May 1973.

C. Requirements

[34] H.B. Reubenstein, R.C. Waters, "The Requirements Apprentice: Automated Assistance for Requirements Acquisition", IEEE Trans. on SE", vol. 17, no. 3, pp. 226-240, March 1991.

[35] "SRD for HyperDoc", COMP648 Course Project, Computer Science Department, Concordia University, December 1990.

[36] "SRD for NED", COMP354 Assignment, Computer Science Department, Concordia University, October 1990.

[37] W. Rzepka, Y. Ohno, "Requirements Engineering Environments: Software Tools for Modeling User Needs", Computer, pp. 9-12, April 1985.

[38] G.-C. Roman, "A Taxonomy of Current Issues in Requirements Engineering", Computer, pp. 14-21, April 1985.

[39] D.T. Ross, "Applications and Extensions of SADT", Computer, pp. 25-34, April 1985.

[40] M. Afford, "SREM at the Age of Eight; The Distributed Computing Design System", Computer, pp. 36-46, April 1985.

[41] P.A. Scheffer, A.H. Stone, W.E. Rzepka, "A Case Study of SREM", Computer, pp. 47-54, April 1985.

[42] G.E. Sievert, T.A. Mizell, "Specification-Based Software Engineering with TAGS", Computer, pp. 56-65, April 1985.

[43] S.M. White, J.Z. Lavi, "Embedded Computer System Requirements Workshop", Computer, pp. 67-70, April 1985.

[44] M. Chandrasekharan, B. Dasarathy, Z. Kishimoto, "Requirements-Based Testing of Real-Time Systems: Modeling for Testability", Computer, pp. 71-80, April 1985.

[45] A. Borgida, S. Greenspan, J. Mylopoulos, "Knowledge Representation as the Basis for Requirements Specifications", Computer, pp. 82-91, April 1985.

[46] R.G. Mays, L.S. Orzech, W.A. Ciarfella, R.W. Phillips, "PDM: A requirements methodology for software system enhancements", IBM System Journal, vol. 24, no. 2, pp. 134-149, 1985.

[47] "IEEE Guide to Software Requirements Specifications", ANSI/IEEE Std 830-1984, Software Engineering Standard, Third Edition, IEEE, October 1989.

D. Design

[48] "Defense System Software Development", DOD-STD-2167A, 29 February 1988.

[49] S.S. Yau, J.J.-P. Tsai, "A Survey of Software Design Techniques", IEEE Trans. on SE, vol. 12, no. 6, pp. 713-721, June 1986.

[50] D.N. Card, V.E. Church, W.W. Agresti, "An empirical Study of Software Design Practices", IEEE Trans. on SE, vol. 12, no. 2, pp. 264-271, February 1986.

[51] P. Grogono, Programming in Pascal, Revised Edition, Addison Wesley, Philippines, 1980.

E. Verification

[52] L. Russell, "Requirements Testing: Broadening the Definition", Software Engineering: Tools, Techniques, Practice, vol. 1, no. 4, pp. 43-45, November/December 1990.

[53] D. Darst, "Balancing Productivity and Quality", Datamation, vol. 36, no. 18, pp. 117-119, September 15, 1990.

[54] Y. Levendel, "Reliability Analysis of Large Software Systems: Defect Data Modeling", IEEE Trans. on SE, vol. 16, no. 2, pp. 141-152, February 1990.

[55] "Software Verification and Validation Plans", ANSI/IEEE Std 1012-1986, Software Engineering Standard, Third Edition, IEEE, October 1989.

[56] D.P. Sidhu, T.-K. Leung, "Formal Methods for Protocol Testing: A Detailed Study", IEEE Trans. on SE, vol. 15, no. 4, pp. 413-426, April 1989.

[57] R. Lewis, D.W. Beck, "Assay - A Tool To Support Regression Testing", pp. 487-496, ...

[58] V.R. Basili, R.W. Selby, "Comparing the Effectiveness of Software Testing Strategies", IEEE Trans. on SE, vol. 13, no. 12, pp. 1278-1296, December 1987.

[59] D. Craigen, "Strengths and Weaknesses of Program Verification Systems", 1st European Software Engineering Conference, pp. 397- 404, 1987.

[60] Z.L. Lichtman, "Generation and Consistency Checking of Design and Program Structures", IEEE Trans. on SE, vol. 12, no. 1, pp. 172-181, January 1986.

[61] A.L. Goel, "Software Reliability Models: Assumptions, Limitations, and Applicability", IEEE Trans. on SE, vol. 11, no. 12, pp. 1411-1423, December 1985.

[62] S. Yamada, S. Osaki, "Software Reliability Growth Modelling: Models and Applications", IEEE Trans. on SE, vol. 11, no. 12, pp. 1431-1437, December 1985.

[63] J.P. Cavaro, "Toward High Confidence Software", IEEE Trans. on SE, vol. 11, no. 12, pp. 1449-1455, December 1985.

[64] R. Troy, R. Moawad, "Assessment of Software Reliability Models", IEEE Trans. on SE, vol. 11, no. 9, pp. 839-849, September 1985.

[65] H.M. Sneed, A. Merey, "Automated Software Quality Assurance", IEEE Trans. on SE, vol. 11, no. 9, pp. 909-916, September 1985.

[66] M. Ohba, "Software reliability analysis models", IBM Journal of Research and Development, vol 28, no. 4, pp. 428-443, July 1984.

[67] V.R. Basili, B.T. Perricone, "Software Errors and Complexity: an Empirical Investigation", Communications of the ACM, vol. 27, no. 1, pp. 42-52, January 1984.

[68] F.J. Buckley, R. Poston, "Software Quality Assurance", IEEE Trans. on SE, vol. 10, no. 1, pp. 36-41, January 1984.

[69] B.W. Boehm, "Verifying and Validating Software Requirements and Design Specifications", IEEE Software, vol. 1, no. 1, pp. 75-88, January 1984.

[70] L.A. Clarke, D.J. Richardson, "Symbolic Evaluation - An Aid to Testing and Verification", Software Validation, Elsevier Science Publishers B.V., New York, 1984, pp. 141-166.

[71] S.H. Saib, "RXVP - Today and Tomorrow", Software Validation, Elsevier Science Publishers B.V., New York, 1984, pp. 103-125.

[72] L. Osterweil, "Integrating the Testing, Analysis and Debugging of Programs", Software Validation, Elsevier Science Publishers B.V., New York, 1984, pp. 73-102.

[73] H.L. Hausen, M. Mullerburg, "An Introduction to Quality Assurance and Control of Software", Software Validation, Elsevier Science Publishers B.V., New York, 1984, pp. 3-9.

[74] P.N. Misra, "Software reliability analysis", IBM Systems Journal, vol. 22, no. 3, 1983.

[75] G.J. Myers, The Art of Software Testing, John Wiley & Sons, 1979.

F. Inspection

[76] G.W. Russell, "Experience with Inspection in Ultralarge-Scale Developments", IEEE Software, pp. 25-31, January 1991.

[77] D.B. Bisant, J.R. Lyle, "A Two-Person Inspection Method to Improve Programming Productivity", IEEE Trans. on SE, vol. 15, no. 10, pp. 1294-1304, October 1989.

[78] S.M. Stevens, "Intelligent Iteractive Video Simulation of a Code Inspection", Communications of the ACM, vol. 32, no. 7, pp. 832-843, July 1989.

[79] A.F. Ackerman, L.S. Buchwald, F.H. Lewski, "Software Inspections: An Effective Verification Process", IEEE Software, pp. 31-36, May 1989.

[80] M.E. Fagan, "Advances in Software Inspections", IEEE Trans. on SE, vol. 12, no. 7, pp. 744-751, July 1986.

[81] M.E. Graden, P.S. Horsley, T.C. Pingel, "The Effects of Software Inspections on a Major Telecommunications Project", AT&T Technical Journal, vol. 65, no. 3, pp. 32-40, May/June 1986.

[82] P.J. Fowler, "In-Process Inspections of Workproducts at AT&T", AT&T Technical Journal, vol. 65, no. 2, pp. 102-112, March/April 1986.

[83] W.L.G. Koontz, "Experience with Software Inspections in the Development of Firmware for a Digital Loop Carrier System", IEEE International Conference on Communications 1986 Conference Record, IEEE, New York, pp. 1188-1189, 1986.

[84] G.M. Weinberg, D.P. Freedman, " Reviews, Walkthroughs, and Inspections", IEEE Trans. on SE, vol. 10, no. 1, pp. 68-72, January 1984.

[85] H. Remus, "Integrated Software Validation in the View of Inspections/Reviews", Software Validation, Elsevier Science Publishers B.V., New York, 1984, pp. 57-64.

[86] R.D. Buck,J.H. Dobbins, "Application of Software Inspection Methodology in Design and Code", Software Validation, Elsevier Science Publishers B.V., New York, 1984, pp. 41-56.

[87] A.F. Ackerman, P.J. Fowler, R.G. Ebenau, "Software Inspections and the Industrial Production of Software", Software Validation, Elsevier Science Publishers B.V., New York, 1984, pp. 13-40.

[88] G.J. Myers, "A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections", Communications of the ACM, vol. 21, no. 9, pp. 760-768, September 1978.

[89] M.E. Fagan, "Design and code inspections to reduce errors in program development", IBM Systems Journal, vol. 15, no. 3, pp. 182-211, 1976.

G. Validation

[90] J.T. Nosek, R.B. Schwartz, "User Validation of Information System Requirement: some Empirical Results", IEEE Trans. on SE, vol. 14, no. 9, pp. 1372-1375, September 1988.

[91] H.-L. Hausen, "Comments on Practical Constraints of Software Validation Techniques", Software Validation, Elsevier Science Publishers B.V., New York, 1984, pp. 323-333.

[92] E. Ploedereder, "Symbolic Evaluation as a Basis for Integrated Validation", Software Validation, Elsevier Science Publishers B.V., New York, 1984, pp. 167-185.

H. Maintenance

[93] J.E. Moore, "A Four-Phase Methodology for Software Maintenance", Software Engineering: Tools, Techniques, Practice, vol. 1, no. 5, pp. 13-23, January/February 1991.

[94] "Scaling Up: A Research Agenda for Software Engineering", CSTB Report, Communications of the ACM, vol. 33, pp. 281-293, March 1990.

[95] J. Moad, "Maintaining the Competitive Edge", Datamation, pp. 61-66, February 1990.

[96] F.J. Buckley, "A Standard Environment for Software Production", IEEE Computer, pp. 75-77, January 1990.

[97] F.J. Buckley, "Some Standards for Software Maintenance", IEEE Computer, pp. 69-70, November 1989.

[98] S.S. Yau, R.A. Nicholl, J.J.-P. Tsai, S.-S. Liu, "An integrated Life-Cycle Model for Software Maintenance", IEEE Trans. on SE, vol. 14, no. 8, pp. 1128-1144, August 1988.

[99] R.P. Hall, "Seven Ways to Cut Software Maintenance Costs", Datamation, pp. 81-84, July 15, 1987.

[100] N.T. Schneidewind, "The State of Software Maintenance", IEEE Trans. on SE, vol. 13, no. 3, pp. 303-310, March 1987.

[101] S. Bendifallah, W. Scacchi, "Understanding Software Maintenance Work", IEEE Trans. on SE, vol. 13, no. 3, pp. 311-323, March 1987.

[102] S.S. Yau, J.S. Collofello, "Design Stability Measures for Software Maintenance", IEEE Trans. on SE, vol. 11, no. 9, pp. 849-856, September 1985.

[103] I. Vessey, R. Weber, "Some Factors Affecting Program Repair Maintenance: An Empirical Study", Communications of the ACM, vol. 26, no. 2, pp. 128-134, February 1983.

[104] J. Martin, C. McClure, Software Maintenance: The Problem and Its Solution, Prentice-Hall, 1983, ch. 1,2, pp. 3-40.

I. Miscellaneous

[105] M.E. Crosby, J.Stelovsky, "How Do We Read Algorithms? A Case Study", Computer, vol. 23, no. 1, pp. 24-35, January 1990.

[106] "Glossary of Software Engineering Terminology", ANSI/IEEE Std 729-1983, Software Engineering Standard, Third Edition, IEEE, October 1989.

[107] V.R. Basili, H.D. Mills, "Understanding and Documenting Programs", IEEE Trans. on SE, vol. 8, no. 3, pp. 270- 283, May 1982.

[108] G. Birks, P. Davis, "Compound Electronic Document Creation, Storage, Retrieval, and Delivery in Bell-Northen Research and Northern Telecom", pp. 16-19, ...

[109] D.E. Webster, "Mapping the Design Representation Terrain", Technical Report STP-093-87, MCC, Software Technology Program, July 1987.

[110] F.P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering", IEEE Computer, pp. 10-19, April 1987.

# Appendix A
# The infoMap for the Waterfall Model

| | A | B | C | D |
|---|---|---|---|---|
| 1 | A | 1 | { IDENTIFICATION } | |
| 2 | v | | | 1. infoSchema for the Waterfall Model |
| 3 | . | | | of the Software Development Process. |
| 4 | A | 1 | { REFERENCE } | |
| 5 | v | | | 1. IEEE Software Engineering Standards [55] |
| 6 | A | 2 | { VIEW } | |
| 7 | v | | | 1. Software Life Cycle Process |
| 8 | . | v | | 2. Set Cardinality |
| 9 | O | 36 | { TRANSITION } | |
| 10 | L | 9 | { STATE } | |
| 11 | G | 7 | { PRE-CONDITION } | |
| 12 | G | 7 | { EVENT } | |
| 13 | S | X | { ACTION } | |

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | AA | AB | AC | AD | AE | AF | AG | AH | AI | AJ | AK | AL | AM |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | 1 | {IDENTIFICATION} |
| 2 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | 1 InfoMap for the Waterfall Model |
| 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | of the Software Development Process |
| 4 | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | 1 | {REFERENCE} |
| 5 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | 1 IEEE Software Engineering Standards [56] |
| 6 | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | 2 | {VIEW} |
| 7 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | 1 Software Life Cycle Process |
| 8 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | v | 2 Set Cardinality |
| 9 | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | 36 | {TRANSITION} |
| 10 | o | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 1 |
| 11 | | o | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 2 |
| 12 | | | o | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 3 |
| 13 | | | | o | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 4 |
| 14 | | | | | o | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 5 |
| 15 | | | | | | o | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 6 |
| 16 | | | | | | | o | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 7 |
| 17 | | | | | | | | o | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 8 |
| 18 | | | | | | | | | o | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 9 |
| 19 | | | | | | | | | | o | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 10 |
| 20 | | | | | | | | | | | o | | | | | | | | | | | | | | | | | | | | | | | | | | | | 11 |
| 21 | | | | | | | | | | | | o | | | | | | | | | | | | | | | | | | | | | | | | | | | 12 |
| 22 | | | | | | | | | | | | | o | | | | | | | | | | | | | | | | | | | | | | | | | | 13 |
| 23 | | | | | | | | | | | | | | o | | | | | | | | | | | | | | | | | | | | | | | | | 14 |
| 24 | | | | | | | | | | | | | | | o | | | | | | | | | | | | | | | | | | | | | | | | 16 |
| 25 | | | | | | | | | | | | | | | | o | | | | | | | | | | | | | | | | | | | | | | | 16 |
| 26 | | | | | | | | | | | | | | | | | o | | | | | | | | | | | | | | | | | | | | | | 17 |
| 27 | | | | | | | | | | | | | | | | | | o | | | | | | | | | | | | | | | | | | | | | 18 |
| 28 | | | | | | | | | | | | | | | | | | | o | | | | | | | | | | | | | | | | | | | | 19 |
| 29 | | | | | | | | | | | | | | | | | | | | o | | | | | | | | | | | | | | | | | | | 20 |
| 30 | | | | | | | | | | | | | | | | | | | | | o | | | | | | | | | | | | | | | | | | 21 |
| 31 | | | | | | | | | | | | | | | | | | | | | | o | | | | | | | | | | | | | | | | | 22 |
| 32 | | | | | | | | | | | | | | | | | | | | | | | o | | | | | | | | | | | | | | | | 23 |
| 33 | | | | | | | | | | | | | | | | | | | | | | | | o | | | | | | | | | | | | | | | 24 |
| 34 | | | | | | | | | | | | | | | | | | | | | | | | | o | | | | | | | | | | | | | | 25 |
| 35 | | | | | | | | | | | | | | | | | | | | | | | | | | o | | | | | | | | | | | | | 26 |
| 36 | | | | | | | | | | | | | | | | | | | | | | | | | | | o | | | | | | | | | | | | 27 |
| 37 | | | | | | | | | | | | | | | | | | | | | | | | | | | | o | | | | | | | | | | | 28 |
| 38 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | o | | | | | | | | | | 29 |
| 39 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | o | | | | | | | | | 30 |
| 40 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | o | | | | | | | | 31 |
| 41 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | o | | | | | | | 32 |
| 42 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | o | | | | | | 33 |
| 43 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | o | | | | | 34 |
| 44 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | o | | | | 35 |
| 45 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | o | | | 36 |
| 46 | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | L | 9 | {STATE} |
| 47 | s | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 1 Concept Exploration |
| 48 | d | l | s | d | | | d | | | | d | | | | | d | | | | | | d | | | | | | | | d | | | | | | | | | 2 Concept |
| 49 | | | d | s | l | s | | d | | | | d | | | | | d | | | | | | d | | | | | | | | d | | | | | | | | 3 Requirements |
| 50 | | | | | | d | s | s | l | s | | | d | | | | | d | | | | | | d | | | | | | | | d | | | | | | | 4 Design |
| 51 | | | | | | | | | | d | s | s | s | l | s | | | | d | | | | | | d | | | | | | | | d | | | | | | 5 Implementation |
| 52 | | | | | | | | | | | | | | | d | s | s | s | s | l | s | | | | | d | | | | | | | | d | | | | | 6 Test |
| 53 | | | | | | | | | | | | | | | | | | | | | d | s | s | s | s | s | l | s | | | | | | | | d | | | 7 Installation and checkout |
| 54 | | | | | | | | | | | | | | | | | | | | | | | | | | | | d | s | s | s | s | s | s | s | l | s | | 8 Operation and maintenance |
| 55 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | d | | 9 Retirement |
| 56 | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | 7 | {PRE CONDITION} |
| 57 | | •l | l | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 1 Concept phase completed? |
| 58 | | | | | •l | l | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 2 Requirements phase completed? |
| 59 | | | | | | | | | •l | l | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 3 Design phase completed |
| 60 | | | | | | | | | | | | | | •l | l | | | | | | | | | | | | | | | | | | | | | | | | 4 Implementation phase completed? |
| 61 | | | | | | | | | | | | | | | | | | | | •l | l | | | | | | | | | | | | | | | | | | 5 Test phase completed? |
| 62 | | | | | | | | | | | | | | | | | | | | | | | | | | | •l | l | | | | | | | | | | | 6 Installation and checkout phase completed? |
| 63 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | l | l | | | 7 Operation and maintenance phase completed? |
| 64 | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | 7 | {EVENT} |
| 65 | l | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 1 Process start |
| 66 | | •l | | l | | | l | | | | l | | | | | l | | | | | | l | | | | | | | | l | | | | | | | | | 2 Concept change |
| 67 | | | | | •l | | | l | | | | l | | | | | l | | | | | | l | | | | | | | | l | | | | | | | | 3 Requirements change |
| 68 | | | | | | | | | •l | | | | l | | | | | l | | | | | | l | | | | | | | | l | | | | | | | 4 Design change |
| 69 | | | | | | | | | | | | | | •l | | | | | l | | | | | | l | | | | | | | | l | | | | | | 5 Implementation change |
| 70 | | | | | | | | | | | | | | | | | | | | •l | | | | | | l | | | | | | | | l | | | | | 6 Test change |
| 71 | | | | | | | | | | | | | | | | | | | | | | | | | | | •l | | | | | | | | l | | | | 7 Installation and checkout change |
| 72 | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | x | {ACTION} |

# Appendix B

## The infoMap for the infoStructure Notation

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | A | A | A | A | A | A | A | | { IDENTIFICATION } |
| 2 | v | v | v | v | v | v | v | v | v | v | v | v | | 1. infoMap for the infoStructure Notation. |
| 3 | A | A | A | A | A | A | A | A | A | A | A | A | | { VIEW } |
| 4 | v | v | v | v | v | v | v | v | v | v | v | | | 1. Hierarchy |
| 5 | . | . | . | . | . | . | . | . | . | . | | v | | 2. Set Cardinality |
| 6 | H | H | H | H | H | H | H | H | H | H | H | 11 | | { SET ROLE } |
| 7 | p | . | . | | . | . | . | . | . | . | . | | | 1. A ::= Association of relations |
| 8 | | p | . | . | | . | . | . | . | . | . | | | 2. F ::= Flow |
| 9 | . | | p | . | . | . | . | . | . | . | . | | | 3. G ::= Guard |
| 10 | . | | . | p | | . | . | . | . | . | . | | | 4. H ::= Hierarchy |
| 11 | . | . | . | . | p | . | . | . | . | . | . | | | 5. L ::= Directed graph with cycles anf forks |
| 12 | . | . | . | . | . | p | . | . | . | . | . | | | 6. M ::= Many |
| 13 | . | . | | . | . | . | p | . | . | . | . | | | 7. O ::= Dominant set |
| 14 | . | . | . | . | . | . | . | p | . | . | . | | | 8. S ::= Sequence |
| 15 | . | . | | . | . | . | . | . | p | . | . | | | 9. T ::= One |
| 16 | . | . | . | . | . | . | . | . | . | p | . | | | 10. V ::= Value |
| 17 | . | . | . | . | . | . | . | . | . | . | p | | | 11. X ::= Unidentified value |
| 18 | H | H | H | H | H | H | H | H | H | H | H | 9 | | { SET MEMBER ROLE } |
| 19 | c | . | . | . | . | c | | . | c | . | . | | | 1. v ::= Column marker |
| 20 | . | c | . | . | | . | . | . | . | . | . | | | 2. i,o,b ::= Flow direction: "in", "out", "both" |
| 21 | . | . | c | | . | . | . | . | . | . | . | | | 3. t,f ::= Guard status: "true", "false" |
| 22 | . | | . | c | . | . | . | . | . | | . | | | 4. p,c ::= Hierarchical structure: "parent", "child" |
| 23 | . | . | . | c | . | . | . | c | . | . | . | | | 5. 1..n ::= Place in sequence |
| 24 | . | . | . | . | c | . | . | . | | . | . | | | 6. s,d,l ::= Direction: "source", "destination", "loop" |
| 25 | . | . | . | . | . | . | c | . | . | . | . | | | 7. o ::= Dominant set member |
| 26 | . | . | . | . | . | . | . | . | . | c | . | | | 8. Integer, real, character,... |
| 27 | . | . | . | . | . | . | . | . | . | . | c | | | 9. x ::= unidentified value |

# Appendix C

# Software Requirement Document for HyperDoc

Prepared by
Juliette D'Almeida
Spyros Kattou
Marie Wallace
Anukon Wongyai

submitted in partial fulfilment
of the requirements of
COMP 648
Concordia University
November 1990

TABLE OF CONTENTS


1 INTRODUCTION..... C-5

1.1 Purpose..... C-5

1.2 Scope..... C-6

1.3 Definitions..... C-7

1.4 References..... C-10

1.5 Overview..... C-11

2 GENERAL DESCRIPTION..... C-12

2.1 Product Perspective..... C-12

2.1.1 Nodes..... C-13
2.1.2 Links..... C-13
2.1.3 Organizations..... C-13
2.1.4 Networks..... C-14
2.1.5 Cooperative Work..... C-14
2.1.6 Security in HyperDoc..... C-15
2.1.7 System Model..... C-15

2.2 Product Functions..... C-16

2.3 User Characteristics..... C-18

2.4 Special Constraints..... C-19

2.5 Assumptions..... C-20

3 SPECIFIC REQUIREMENTS..... C-22

3.1 Functional Requirements..... C-22

3.1.1 Show_Outgoing_Links..... C-22
3.1.2 Show_Incoming_Links..... C-23
3.1.3 Follow_Link..... C-24
3.1.4 Move_To_Unconnected_Node..... C-26
3.1.5 Backtrack_One_Step..... C-27
3.1.6 Display_Overview_Diagram..... C-28
3.1.7 Display_Network_Attributes..... C-29
3.1.8 Display_Organization_Attributes..... C-30
3.1.9 Display_Node_Attributes..... C-31
3.1.10 Display_Link_Attributes..... C-32
3.1.11 Destroy_Information_Window..... C-33

3.1.12 Change_Node..... C-34
3.1.13 Lock_Node..... C-35
3.1.14 Unlock_Node..... C-36
3.1.15 Search_For_String..... C-37
3.1.16 Search_For_Next_Occurrence_Of_String..... C-38
3.1.17 Search_For_An_Attribute..... C-39
3.1.18 Assign_Link_Attribute..... C-40
3.1.19 Assign_Node_Attribute..... C-41
3.1.20 Assign_Network_Attribute..... C-42
3.1.21 Assign_Organization_Attribute..... C-43
3.1.22 Assign_Region_Attribute..... C-44
3.1.23 Create_Node..... C-45
3.1.24 Delete_Node..... C-46
3.1.25 Split_Node..... C-47
3.1.26 Insert_Node..... C-48
3.1.27 Save_Node_Under_Current_Version..... C-49
3.1.28 Assign_Node_To_Network..... C-50
3.1.29 Create_Region..... C-51
3.1.30 Delete_Region..... C-52
3.1.31 Create_Link..... C-53
3.1.32 Delete_Link..... C-54
3.1.33 Create_Network..... C-55
3.1.34 Delete_Network..... C-56
3.1.35 Create_Organization..... C-57
3.1.36 Delete_Organization..... C-58
3.1.37 Invoke_HyperDoc..... C-59
3.1.38 Shutdown_HyperDoc ..... C-60
3 1.39 Show_Network_List..... C-61
3.1.40 Show_List_Of_Organizations_Within_A_Network.. C-62
3.1.41 Select_Network... ..... C-63
3.1.42 Move_To_Next_Node..... C-64
3.1.43 Save_Node_Under_New_version..... C-65
3.1.44 Retrieve_Previous_Version_Of_A_Node..... C-66
3.1.45 Create_User_Class..... C-67
3.1.46 Delete_User_Class..... C-68
3.1.47 Update_User_Class..... C-69
3.1.48 Create_User..... C-70
3.1.49 Delete_User..... C-71
3.1.50 Update_User..... C-72
3.1.51 Assign_User_To_Class..... C-73
3.1.52 Delete_User_From_Class..... C-74
3.1.53 Assign_Object_And_Rights_To_Manager..... C-75
3.1.54 Delete_Object_And_Rights_From_Manager..... C-76
3.1.55 Update_Object_Rights_Of_Manager..... ... C-77
3.1.56 Show_Object_Access_Rights..... C-78
3.1.57 Select_Organization..... C-79
3.1.58 Display_Region_Attributes..... C-80
3.1.59 Display_Assigned_Object_Ids..... C-81
3.1.60 Future Requirements..... C-82

3.2 Overall Performance Requirements..... C-83

3.2.1 Timing Constraints..... C-83

3.3 Attributes..... C-84

3.3.1 Availability..... C-84
3.3.2 Security..... C-84

3.4 Acceptance Criteria..... C-85

# 1. INTRODUCTION

## 1.1 Purpose

The purpose of this SRD is to bring out the functional and performance requirements for the design of a hyperdocument system, called HyperDoc. Hypertext is a well-known technique for information representation and management in which data is stored in networks of nodes connected by links. The nodes are fragments of information such as text, graphics, images, voice and animation. The links reflect relationships between these information fragments. A user can browse a hyperdocument by traversing the network and view the information fragments as they are visited, thus allowing the user to have non-sequential access to a text or multimedia document.

This SRD is written by a group of four students as part of the project for Specifications of Software Systems course (COMP648) at Concordia University. The document will be submitted to Dr. Alagar for verification and evaluation of the functional and performance requirements.

## 1.2 Scope

HyperDoc will work under a multi-user and multi-tasking environment. Although HyperDoc may be used for several purposes, it's main purpose as a support environment for the software life cycle will be outlined in this document. HyperDoc will perform under a multiple window workstation.

HyperDoc will provide, among others, capabilities to allow for the documentation and the management of the software development process. Handling such documents with HyperDoc provides a corporate memory for the development history while permitting the simultaneous viewing of multiple sections of the same document of similar, related or referenced documents, by more than one user. From now on, a document built with HyperDoc will be referred to as a hyperdocument.

Generalizing, HyperDoc will provide facilities for:

a. navigating through hyperdocuments;

b. versioning of hyperdocuments;

c. editing hyperdocuments as well as storage and retrieval facilities;

d. defining the structure of a hyperdocument (links and nodes);

e. providing security management to hyperdocuments or parts of the hyperdocuments; and

f. providing support for cooperative work.

The objective of HyperDoc is to make the users feel they can move freely through the information according to their own needs as well as providing manipulation facilities for the document's organization and the document's content. Particularly, it provides a tool for Software developers to manage their interconnected multimedia documents.

## 1.3 Definitions

| | |
|---|---|
| Anchor: | Node or region in a node which can be selected by the user in order to use a link. The anchor selected is the departure point of a link. Anchors are created within a node by authorized users. |
| Attribute: | An attribute is a pair (label, value) existing for links, nodes, regions, networks and organizations used to describe them. Each of the previous has some fixed labels, and can support an unlimited number of user-defined attribute pairs. The labels must be unique. For example, some attributes that describe a node would be Author: "John Smith", Date: "Oct. 20 1990", Description: "The New Release of Srd". Author, Date and Description are labels, and "John Smith", "Oct. 20 1990" and "The New Release of Srd" are the corresponding values. |
| Cursor position: | A place marker in the node window. |
| Departure point: | The source point of a link. It can be a node or a region in a node. |
| Destination point: | The destination in a hyperdocument that a link leads to. It can be a node or a region in a node. |
| Hyperdocument: | The entire HyperDoc information space (contents of all nodes of all networks). |
| Information window: | Window used to display any information about HyperDoc's objects (links, nodes, networks, organizations). The user can only browse the information and is not allowed to change it. An information window contains a header with its name. |
| Link: | Means of relating two pieces of information. A link is defined by its departure point and its destination point. The link provides a way for moving from one point to another. |

| | |
|---|---|
| Link attributes: | The list of attribute labels minimally required for links is: id, departure point, destination point, author, last modified date, type (respond_to, agree, object_to...), description. |
| Lock attribute: | An attribute of the node which when set to true by a user disables the other users from changing the content of the node. When this attribute is false no users are modifying the node. For example, when a user needs to change the content of a node, he/she must first lock the node to prevent other users from changing it at the same time. |
| Map: | Similar to an index, where a list of user defined keywords map into nodes. |
| Navigation path: | The ordered list of nodes visited by the user in the current network. |
| Network: | Set of nodes and links which have some logical association. A certain concept can be captured within a network. A network can have several organizations. |
| Network attributes: | The list of attribute labels minimally required for networks is: id, author, last modified date, description. |
| Node: | A piece of information of a specific medium. It is the fundamental unit of HyperDoc. Nodes are represented as node windows on the screen. |
| Node attributes: | The list of attribute labels minimally required for nodes is: id, description, author, date, type (text, graphic...), last modified date, version number, lock (true or false). |
| Node window: | A means of displaying a node on the screen. A node window is provided the appropriate tool to edit or browse the content of a node. A node window contains a header with the node's id. |
| Organization: | An arrangement of nodes of a network as a set, a map or a sequence. An organization can only be one of the above. |

| | |
|---|---|
| Organization attributes: | The list of attribute labels minimally required for organizations is: id, author, creation date, type (map, set, sequence or user-defined). |
| Overview diagram: | A bird's eye view of the network (nodes and links) with the current navigation path highlighted. |
| Region: | A region is a section in a node of contiguous text, voice, video, graphics, etc defined by the user. A region serves as an endpoint (source or destination) of a link. A region can be of any size. |
| Region attributes: | The list of attribute labels minimally required for regions is: id, parent node, description, type (text, graphic...). |
| Sequence: | A user defined ordering that is imposed on some nodes in a network, i.e the nodes can be accessed in this order. |
| Set: | A user defined collection of nodes that may be accessed in any order. |
| Super user: | A user with full rights at all time in HyperDoc system. |
| User: | A person belonging to at least one user class. |
| User class: | Describes the role of people and defines the scope of their work within the software life cycle context. Contains a list of users assigned to it. |
| User leader: | A supervisor defined for a user class having the authority to manage the rights of users in that user class. |

## 1.4 References


[ALAGAR] V.S. Alagar, P. Goyal, "CONSLT: A Software Life Cycle Tool", Concordia University, Montreal, Canada.

[GARG1] Pankaj K. Garg and Walt Scacchi, "ISHYS Designing an Intelligent Software Hypertext System", University of Southern California, IEEE EXPERT, Fall 1989.

[LANGE] Danny B. Lange, "A formal Approach to Hypertext using Post-Prototype Formal Specification", Department of Computer Science, Technical University of Denmark.

[NEILSON] Jacob Neilson, Hypertext and Hypermedia, Academic press, 1990.

[GARG2] Pankaj K. Garg and Walt Scacchi, "A Hypertext System to manage Software Life-Cycle Documents", IEEE Software, May 1990.

[CONKLIN] Jeff Conklin, "Hypertext - An Introduction and Survey", Microelectronics and Computer Technology Corp., Computer, September 1987.

## 1.5 Overview

The remaining part of the document is organized as given below. Section 2.1 gives a perspective of HyperDoc as defined by it's objects and their organization. Nodes, links, organizations and networks are introduced. However, the concept of how cooperative work can be handled with HyperDoc is also outlined. Security considerations are introduced. The system model followed by a brief description of it is also provided. The main tasks expected to be performed by HyperDoc are delineated in Section 2.2 with a brief description of what is to be provided by each task. Section 2.3 gives the user involvement during the building and utilization of a hyperdocument. Special constraints regarding hardware or HyperDoc itself are brought out in Section 2.4.

The assumptions on which the SRD is based are listed in Section 2.5. This section also includes some presupposed knowledge about the management of rights to users and the availability of tools used by HyperDoc.

Section 3.1 gives specific functional requirements. Each requirement is given in a self-contained form giving the input, action, result, exception handling and remarks. The format will help in test-data preparation for verification of individual requirements. Since the tasks involved have to perform in a coordinated manner, overall performance requirements not conflicting with individual requirements are given in Section 3.2.

Specific attributes and their conformity with HyperDoc are brought out in Section 3.3. The acceptance criteria for HyperDoc, the various stages of review and the scope of each review are given in Section 3.4. References cited in Section 1.5 provide an excellent background to the principles of Hypertext.

## 2. GENERAL DESCRIPTION

### 2.1 Product Perspective

HyperDoc consists of interlinked pieces of information. These pieces are called nodes. A node is the basic unit of information in HyperDoc. Whatever the granularity of these nodes, each may have pointers to other units, and these pointers are called links. Links and nodes form networks in HyperDoc. The entire HyperDoc is the set of all networks. The responsibility of defining the content of nodes or the structure of networks rests with the user. Figure 2.1 shows a HyperDoc system where nodes are illustrated as circles.

*****
put network example here
****

Network A: one organization is the sequence of nodes A,C,D,B

Network B: one organization is the set of nodes E,D,H,F,G

Node D is shared by both networks

Figure 2.1. Simplified view of small HyperDoc structure having two networks

A HyperDoc system can handle multiple users and provide means for cooperative work. HyperDoc will execute on homogeneous workstations suitably networked for data communication. The communication protocols do not concern HyperDoc.

### 2.1.1 Nodes

HyperDoc represents a node as a window on the screen. Multiple nodes may be displayed on the screen. Each window contains a header specifying the identification of its node. Editing within the window as well as scrolling mechanisms are provided by appropriate tools already available which HyperDoc uses (refer to Section 2.1.7). Multiple users can access the same node. However only one user can modify the content of a node at a time, assuming he/she possesses the appropriate rights.

A user can define regions within a node. Regions are marked areas within a node. Regions and nodes can be departure points or destination points of links. Regions are unique within their node. Attributes can be assigned to regions. A node can only handle one type of medium and it can be of any size. Nodes are uniquely named throughout the entire hyperdocument and possess attributes. Versioning is done at the node level.

### 2.1.2 Links

Links provide a way of moving from one departure point to one destination point within a network. We may have several links from one departure point and similarly, several links to one destination point. A user with appropriate rights can follow a link from a source to explore the information within the destination of the link. A link is named and attributed so that HyperDoc applications can associate higher level semantic to the link. This allow further developments such as query base on the semantic of these links. This leads to a high potential for the query which is based on the logic or inference rule to make HyperDoc more attractive. For example, in a HyperDoc application, an enterprise can organize link names such that it supports a cooperative work (refer to Section 2.1.5).

Future requirement may call for other types such as active links in which a link points to a function executing some process.

### 2.1.3 Organizations

Organizations are created by the user to organize the nodes of a given network as a set, a map or a sequence. When HyperDoc is invoked, an organization must be selected afterwards. If the organization is a sequence, nodes may be visited in that logical sequence using the function Move_to_Next_Node. At any time, the links can always be used to navigate through the network as usual and are totally independent of the logical sequence of nodes. The user choose

to navigate through nodes with links or following the sequence defined.

### 2.1.4 Networks

Nodes and links form networks. A network can contain many user defined organizations. When HyperDoc is invoked, a network must be selected afterwards. A node may be shared by several networks. Networks have a unique identity and are attributed.

### 2.1.5 Cooperative Work

As it was stated before, HyperDoc provides a perfect environment for the documentation aspect in Software Development. In such an environment, work is done within the solution space of the problem/task by many people/experts, but each piece of work is uniquely identified by its creator. When the author of a piece of work wishes to receive feedback/comments on his/her work, he/she can post an issue on the work done to a selected audience (i.e a selected set of users, project group(s), etc). The audience can then individually or jointly respond to, question, support, object to, suggest, take a position, argue, etc... on the issue and the positions taken and the arguments put forward. For example, user1 is responsible for a functional requirement, but needs some additional information before proceeding. The user then decides to post an issue, which may be a question, a call for help, etc... to user2 and user3. User2 and user3, in turn, may respond to the issue by agreeing or disagreeing in the form of a position or argument. User1 may then follow the suggestions in the positions and/or argument nodes to complete his work.

To fit this within our definition of HyperDoc, issues (or concepts), positions (or responses to an issue) and arguments (or debates) reside in nodes and the links which connect them carry one of the types mentioned above (i.e respond_to, question, object_to, etc). The author of an issue can see the list of all incoming and outgoing links to the issue node and depending on the feedback/comments he/she receives, can make decisions as to the direction of his/her work. He/she may decide for example to create a new version of the work after consulting and evaluating all feedback/comments by the other users.

This method then, not only supports cooperation during the software engineering process but provides a means for tracing the decision making process throughout the life of a piece of work, from problem formulation to solution. The following diagram outlines the concepts described here.

*******
PUT DIAGRAM FROM CON87 PP.25
**********

### 2.1.6 Security in HyperDoc

HyperDoc is an object made out of nodes, links, organizations and networks. For each of these objects there exists a creator. In the context of software engineering, the creation of each object is done within the scope of one of the software life cycle processes or within the scope of the interaction or relationship(s) of these processes. People creating these objects, the authors, belong to different classes such as class user, class analyst, class designer, class programmer, etc... Documents or work associated with the SLC processes such as problem definition, software requirement specification, design, source code, etc... are the objects which the different classes of users and in effect the users within those classes, manipulate.

In order to protect these documents and the system in general from unauthorized access as well as define the scope of the users in such system, the need for a security manager exists in order to assign rights to users or a class of users. A user inherits the rights (permissions) of the user class where he/she belongs (Refer to Section 2.5 for the rules assumed to govern security).

### 2.1.7 System Model

HyperDoc is similar to an operating system that sits between the user and the operating system of the host machine. The user is required to login to HyperDoc (with a user id and a password) so that appropriate user permissions can be associated with this user. HyperDoc uses facilities that are available in the operating system, such as time and date utilities.

HyperDoc uses a set of existing tools such as text editors, graphics editors, etc..., to manage a hyperdocument. These tools are assumed to have been modified in order to function in the HyperDoc environment.

The following diagram shows the interface of HyperDoc with the relevant external entities.

*********
put spiros graph and
**********

## 2.2 Product Functions

The tasks that comprise HyperDoc can be broadly categorized into the following (these tasks have been brought out from the user's viewpoint):

a. manipulation of node's content;

b. manipulation of HyperDoc's objects (links, nodes, networks, organizations);

c. handling of attributes for HyperDoc's objects;

d. definition of security management functions;

e. navigation through a hyperdocument; and

f. searches for strings through a hyperdocument.

Manipulation of node's content: HyperDoc is expected to provide a set of functions that perform normal editing on any type of medium being part of the node. Those editing capabilities are provided by existing tools which are used by HyperDoc. At this level, we should be able to save the content of a node (possibly under a new version) and retrieve a specific node version.

Manipulation of HyperDoc's objects: the basic objects of HyperDoc are networks, organizations, nodes, links within a network and regions within nodes. Facilities to add/create or delete any of these must be provided. Functions for the user to select a network and an organization to work with are also provided.

Handling of attributes for HyperDoc's objects: HyperDoc should provide facilities to assign attributes to networks, links, nodes, and regions in nodes. Such attributes are date of creation, author, etc (refer to Section 1.3 for definition of attributes) and others which may be defined by the user. It should also be possible to query on any attribute of any object. For example, selective retrieval of information would be:

a. select all links created on a certain date;

b. select all nodes of type graphic; and

c. select all nodes of a certain author.

Each query's response is displayed in a special window called an information window (refer to Section 1.3 for definition). Functions are also provided to inform the user

on which networks and organizations within a network he/she can work with. The information is stored in such a way that in the future, an inference engine can be built to interpret it.

Definition of security management functions: protection at several levels must be provided by HyperDoc. The ability to assign permissions for creation, deletion or viewing of nodes, links, networks, organizations to different type of user classes or to single users must be available.

Navigation through a hyperdocument: HyperDoc should provide the ability to move from one node/region to another via links. The user has the choice of navigating through the information space as he/she wants using the links available or using a predefined sequence of nodes within a network, if any (refer to Section 2.1.3). Backtracking facilities must also be provided. In this way, it serves as a lifeline for the user who can do anything and still be certain to be able to get back to familiar territory. In addition, the user must also be able to move to any node in the network without the use of links, given the user knows the node identifier.

Searches for strings through a hyperdocument: when dealing with a large information space, one need a search mechanism that will retrieve the occurrences of a string of characters throughout the nodes of the network. As well, the search for next occurrence is also provided.

## 2.3 User Characteristics

Three general classes of users are identified in HyperDoc. A super user who can be seen as the manager of the HyperDoc system. He/She is assumed to be the only one who can create/delete users or user classes. He/She can designate one user as a leader of a user class (refer to Section 2.1.6 on security).

Another user class can be seen as the builders who are authorized to create/delete nodes, links, organizations and networks in HyperDoc.

Finally, another user is the consulter who can only browse hyperdocuments and request specific information. The users are assumed to have some basic knowledge about windowing facilities. As stated before, users can be Software Developers and consequently, one assumes that they possess domain knowledge in order to build the documents appropriately.

## 2.4 Special Constraints

HyperDoc is expected to run under Unix environments. The tools for editing must be modified appropriately to interface with HyperDoc. The operating system must handle protection at the file level (secondary storage). At the hyperdocument level, HyperDoc allow only authorized users (defined within HyperDoc) to access objects.

Only one user can modify the content of a node at a time, given he/she possesses the permissions to do so.

## 2.5 Assumptions

a. atomicity is guaranteed for any function occurring at the node level.

b. the tools to edit multimedia must be available under HyperDoc's environment.

c. regions within a node are highlighted in some way when the node is displayed on the screen.

d. a window environment is assumed to be available. This facility will take care of basic window management functions such as move the window in the screen, resize the window, etc. . HyperDoc does not have to take care of such functions.

e. hardware support is assumed to possess enough characteristics to meet the performance requirements as stated in Section 3.2.

f. it is the user responsibility to decide what to put in the node and how to link them.

g. any modifications of administrative information (add/create nodes, links, organizations, networks) in HyperDoc is saved implicitly and permanently when modified.

h. multiple user access to information is provided by the operating system.

i. the following rules are followed to govern the security system:

   (1) it is assumed that the verification of rights for a specific user is done at the single user level as well as at the user class level where he/she belongs to (refer to Section 2.1.6).
   (2) a user belong to at least one user class.
   (3) a user can access the system only when he/she has been granted permission under a certain user class.
   (4) there exists a SUPER user with full access rights at all times.
   (5) a user or user class can only access links, nodes, organizations and networks they have been granted access to in the manner specified by object rights.
   (6) object rights granted to users or groups of users are: Read, Write, Delete, Create.

(7) only SUPER user can create/delete/update users and user classes.
(8) a user can assign/revoke/update object rights to other users or user classes of only objects he/she has created. The class leader can assign/revoke object rights for any user of his/her class.
(9) SUPER User can assign/revoke/update access rights of any object for any users or user classes.
(10) the author of an object can be a user or a user class.

## 3.0 SPECIFIC REQUIREMENTS

### 3.1 Functional Requirements

#### 3.1.1 NAME: Show_Outgoing_Links

PURPOSE: To identify all the links of the current network having the given anchor identification as their departure point.

INPUT: Anchor id
Current network

ACTION: The links having the anchor identifier as their departure point are searched through the current network and are displayed on a new information window.

RESULT: A new information window is created and displayed over all windows (if any) currently appearing on the screen. This window contains the list of links having the anchor as their departure point.

EXCEPTIONS: When the given anchor is not the departure point of any link. In such a case, a message is displayed to the user.

REMARKS: Anchor identifier may be a Node identifier or a Region identifier with a node identifier.

3.1.2 NAME: Show_Incoming_Links

PURPOSE: To identify all the links of the current network having the given destination identification as their destination point.

INPUT: Destination id
Current network

ACTION: The links having the destination identifier as their destination point are searched through the current network and are displayed on a new information window.

RESULT: A new information window is created and displayed over all windows (if any) currently appearing on the screen. This window contains the list of links having the given destination as their destination point.

EXCEPTIONS: When the given destination in not the destination point of any link. In such a case, a message is displayed to the user.

REMARKS: Destination identifier may be a Node identifier or a Region identifier with a Node identifier.

3.1.3 NAME: Follow_Link

PURPOSE: To move around the information space of the hyperdocument using an explicit link by displaying the node visited.

INPUT: Link id with its departure and destination point
Current node
Current navigation path

ACTION:

i) If the link refers to a destination point in another node,
- If the current node contains unsaved changes, confirm with the user to move or not.
- If the users wants to move,
  - include the current node identifier in the current navigation path,
  - display the latest version of the node referred by the link in a new node window with the cursor positioned at the beginning of the region (if specified by the link) or at the beginning of the node otherwise.
- Display the node identifier in the window header.

ii) If the link refers to a destination point in the same node (current node window), make the region pointed by the link visible in the current node window and position the cursor at the beginning of the region.

RESULT:

i) A new node window is displayed on the screen with the user positioned at the destination point specified by the link.

ii) The user position is at the destination point specified by the link (this region is made visible in the current node window).

EXCEPTIONS:
- Unauthorized user permission.
- Link departure point is not part of the current node.

REMARKS:

- If the user has unsaved changes in the node, he/she is requested to confirm whether to move or not. If he/she decide not to move, his functions is disabled.
- Link destination point contains a node identifier and possibly a region identifier within this node.

3.1.4 NAME: Move_To_Unconnected_Node

PURPOSE: To visit a specified node of the current network without using a link.

INPUT: Node id to visit
Current node identifier
Current navigation path

ACTION:
- If the node contains unsaved changes, confirm with the user to move or not.
- If the user wants to move,
  - include the current node in the current navigation path,
  - display the latest version of the node in a new node window on the screen with the cursor positioned at the beginning of the node,
  - the node identifier is displayed in the node window header.

RESULT: A new node window is displayed on the screen with the cursor positioned at the beginning of the node.

EXCEPTIONS:
- Unauthorized user permission.
- Node identifier does not exist in the current network.

REMARKS: If the user has unsaved changes in the node, he/she is requested to confirm whether to move or not. If he/she decide not to move, this functions is disabled.

3.1.5 NAME: Backtrack_One_Step

PURPOSE: To backtrack one step in the navigation path followed by the user through the network.

INPUT: Current navigation path

ACTION:
- The current position of the user in the network is backtracked one step to the previous node of the navigation path.
- The current node window is cleared from the screen and the cursor in now in the previous node window (made visible) at the previous position.
- Remove the current node from the current navigation path.

  e.g.

  1) current navigation path: node1-->node4-->node2-->node9

     current node ---+

  2) backtrack one step

  3) current navigation path: node1-->node4-->node2

     new current node ---+

RESULT: The current node window disappears from the screen and the previous node window is redisplayed at the front of any windows currently shown on the screen.

EXCEPTIONS:
- When the current visited node is the first node in the current navigation path (cannot back track).
- When the current navigation path is not defined (empty).
- When the previous node in the navigation path does not exist anymore.

REMARKS: Nil

3.1.6 NAME: Display_Overview_Diagram

PURPOSE: To inform the user about its relative position (node position) in the current network.

INPUT: Network
Current navigation path

ACTION:
- Display the diagram of the current network on a new information window.
- Mark the nodes visited so far (nodes which are part of the navigation path) differently from the other nodes.
- Mark the current node differently from the nodes in the navigation path as well as all other nodes in the network.

RESULT: A new information window is displayed over all windows currently shown on the screen, with the overview diagram in it.

EXCEPTIONS: There is no current network selected.

REMARKS: Nil

3.1.7 NAME: Display_Network_Attributes

PURPOSE: To inform the user about the attributes and their assigned values in the given network.

INPUT: Network id
Current network

ACTION: Display in an information window, the list of the network's attribute labels as well as their value.

RESULT: A new information window is displayed on the screen and it contains the list of attributes assigned to the network.

EXCEPTIONS: The network does not exist is the HyperDoc system.

REMARKS: Nil

3.1.8 NAME: Display_Organization_Attributes

PURPOSE: To inform the user about the attributes and their assigned values for the given organization in the current network.

INPUT: Organization id
Current network

ACTION: Display in an information window, the list of the structure's attribute labels as well as their value.

RESULT: A new information window is displayed on the screen and it contains the list of attributes assigned to the structure.

EXCEPTIONS: The organization does not exist in the current network.

REMARKS: Nil

3.1.9 NAME: Display_Node_Attributes

PURPOSE: To inform the user on the attributes and their values assigned to the given node in the current network.

INPUT: Node id
Current network

ACTION: Display in an information window, the list of the node's attribute labels as well as their value.

RESULT: A new information window is displayed on the screen and it contains the list of attributes assigned to the node.

EXCEPTIONS: The node does not exist in the current network.

REMARKS: Nil

3.1.10 NAME: Display_Link_Attributes

PURPOSE: To inform the user on the attributes and their values assigned to the given link.

INPUT: Link identifier
Current network

ACTION: Display in an information window, the list of the link's attribute labels as well as their value.

RESULT: A new information window is displayed on the screen and it contains the list of attributes assigned to the link.

EXCEPTIONS: The link does not exist in the current network.

REMARKS: Nil

3.1.11 NAME: Destroy_Information_Window

PURPOSE: To remove an information window from the screen.

INPUT: Information window id

ACTION: Destroy the information window.

RESULT: The given information window disappears from the screen.

EXCEPTIONS: The window identified is not currently displayed on the screen.

REMARKS: Nil

3.1.12 NAME: Change_Node

PURPOSE: To change the content of a node.

INPUT:
Node id
Node attributes
Editing command
User id

ACTION:
- The editing command is interpreted by the appropriate tool editor for the node (refer to Section 2.1.7) and applied on the node's content.
- If the node or the region changed is referenced by a link who's author is not the current user, versioning is forced (refer to Section 1.3.43 save under new version).
- Update the last modified date attribute of the current network.

RESULT: The content of the node has changed appropriately.

EXCEPTIONS:
- Unauthorized user permission
- The current node is not locked by the user wanting to change it.
- The current node is not the latest version that exists.

REMARKS: From the node type attribute (graphic, text, voice), the appropriate tool is used so that the user can utilize the commands available from that tool to change the content of the node.

For example, an editing command could be "add a character". The appropriate tool will take care of the changes implied by the command (ie, adjusting cursor position, updating the display, etc).

3.1.13 NAME: Lock_Node

PURPOSE: To disable the ability of any other user to change the content of the current node.

INPUT: Current node id
User id
Node attribute list

ACTION:
- Lock the current node for the given user.
- Set the lock attribute of the current node to true.

RESULT: The current node is locked by the user id.

EXCEPTIONS:
- Unauthorized user permission.
- The node is already locked by another user.

REMARKS: Nil

3.1.14 NAME: Unlock_Node

PURPOSE: To unlock the node from the user.

INPUT: Current node id
User id
Node attribute list

ACTION:
- Unlock the current node from the given user.
- Set the lock attribute of the current node to false.

RESULT: The node is unlocked and is available for other users to change it.

EXCEPTIONS:
- The node is not currently locked.
- The user under whom the node was actually locked is not the same as the given user wanting to unlock it.

REMARKS: Nil

3.1.15 NAME: Search_For_String

PURPOSE: To search nodes in a network for a given string.

INPUT: A string
A scope id

ACTION:
- For every node in the given scope, search the node contents for a match to the search string.
- If the search string is found, display the node that contains it, otherwise, display a "not found" message to the user.

RESULT:
- The current node becomes the node where the search string is found.

OR

- A "not found" message is displayed.

EXCEPTIONS: The scope id does not exist in the system.

REMARKS: The scope id can be either a network id OR an organization id. The scope id creates a boundary for the number of nodes that are included in the search.

3.1.16 NAME: Search_For_Next_Occurrence_Of_String

PURPOSE: To find the next occurrence of a search string entered by the user, within the previously specified scope.

INPUT: A string
A scope id

ACTION:
- Find the next occurrence of the string.
- If there is no other occurrence of the search string within the boundaries of the scope, display a "not found" message to the user.

RESULT:
- The current node becomes the node where the search string is found.

OR
- A "not found" message is displayed.

EXCEPTIONS:
- A search string and a scope id were not previously entered.
- The scope id does not exist in the hyperdocument.

REMARKS: The scope id can be either a network id OR an organization id. The scope id creates a bound on the number of nodes that are included in the search.

3.1.17 NAME: Search_For_An_Attribute

PURPOSE: To search all attribute lists associated with links, nodes, regions, networks and organizations for a <label,value> pair that matches the given input.

INPUT:
A <label, value> pair
A search object, (ie a link, node, region, network or organization id) whose attribute lists will be searched (or use the default, search all)
A scope id in which to search, (or use the default, search current network)

ACTION:
- Search the attribute list of the given search object for a <label, value> pair that matches the given input.
- Display a list of search object identifiers having matches in their attribute lists.
- Display a "not found" message if no search objects have a match to the user input in their attribute lists.

RESULT:
- A list of object identifiers whose attribute list contains a match to the given attribute.

OR

- A "not found" message is displayed.

EXCEPTIONS: The given scope id does not exist in the system.

REMARKS: The scope id can be either a network id OR an organization id. The scope id creates a boundary for the number of nodes that are included in the search.

3.1.18 NAME: Assign_Link_Attribute

PURPOSE: To assign an <label, value> pair to a link.

INPUT: Link id
A <label, value> pair
A user id

ACTION:
- Assign the value to the label in the attribute list for the given link. If the label already exists, its value is overwritten by the new value.
- The Last_Modified_Date will be set to the system time when the attributes are assigned, and the Created_By attribute will be set to the name of the user who makes the assignment.
- If no value for the Link_Description is supplied by the user, HyperDoc will give the label a value equal to the first part of the Anchor attribute.

RESULT:
- The attribute list associated with the given link id will have either a new <label, value> pair, or a new value for an existing label.
- System maintained attributes will be updated (Created_By, Last_Modified_Date, Link_Description ).

EXCEPTIONS:
- The link id does not exist in the network.
- The user does not have the appropriate permissions to modify the link attributes.

REMARKS: Nil

3.1.19 NAME: Assign_Node_Attribute

PURPOSE: To assign an <label, value> pair to a node.

INPUT: Node id
A <label, value> pair
A user id

ACTION:
- Assign the value to the label in the attribute list for the given node. If the label already exists, its value is overwritten by the new value.
- The Last_Modified_Date will be set to the system time when the attributes are assigned, and the Created_By attribute will be set to the name of the user who makes the assignment.
- If no value for the Node_Description is supplied by the user, HyperDoc will give the Label a value equal to the first part of the content of the node.

RESULT:
- The attribute list associated with the given node id will have either a new <label, value> pair, or a new value for an existing label.
- System maintained attributes will be updated (Created_By, Last_Modified_Date, Node_Description).

EXCEPTIONS:
- The node id does not exist in the network.
- The user does not have the appropriate permissions to modify the node attributes.

REMARKS: Nil

3.1.20 NAME: Assign_Network_Attribute

PURPOSE: To assign an <label, value> pair to a network.

INPUT: Network id
A <label, value> pair
A user id

ACTION:
- Assign the value to the label in the attribute list for the given network. If the label already exists, its value is overwritten by the new value.
- The Last_Modified_Date will be set to the system time when the attributes are assigned, and the Created_By attribute will be set to the name of the user who makes the assignment.
- If no value for the Network_Description is supplied by the user, HyperDoc will give the Network_Description a default value.

RESULT:
- The attribute list associated with the given network id will have either a new <label, value> pair, or a new value for an existing label.
- System maintained attributes will be updated (Created_By, Last_Modified_Date, Network_Description).

EXCEPTIONS:
- The network id does not exist in the system.
- The user does not have the appropriate permissions to modify the network attributes.

REMARKS: Nil

3.1.21 NAME: Assign_Organization_Attribute

PURPOSE: To assign an <label, value> pair to a organization.

INPUT: Organization id
A <label, value> pair
A user id

ACTION:
- Assign the value to the label in the attribute list for the given organization. If the label already exists, its value is overwritten by the new value.
- The Last_Modified_Date will be set to the system time when the attributes are assigned, and the Created_By attribute will be set to the name of the user who makes the assignment.
- If no value for the Organization_Description is supplied by the user, HyperDoc will give the Description a default value.

RESULT:
- The attribute list associated with the given organization id will have either a new <label,value> pair, or a new value for an existing label.
- System maintained attributes will be updated (Created_By, Last_Modified_Date, Organization_Description).

EXCEPTIONS:
- The organization id does not exist.
- The user does not have the appropriate permissions to modify the organization attributes.

REMARKS: Nil

3.1.22 NAME: Assign_Region_Attribute

PURPOSE: To assign an <label, value> pair to a region.

INPUT: Region id
A <label, value> pair
A user id

ACTION:
- Assign the value to the label in the attribute list for the given region. If the label already exists, its value is overwritten by the new value.
- The Last_Modified_Date will be set to the system time when the attributes are assigned, and the Created_By attribute will be set to the name of the user who makes the assignment.
- If no value for the Region_Description is supplied by the user, the hypertext system will give the Region_Description a value equal to the first part of the contents of the region.

RESULT:
- The attribute list associated with the given region id will have either a new <label, value> pair, or a new value for an existing label.
- System maintained attributes will be updated (Created_By, Last_Modified_Date, Region_Description).

EXCEPTIONS:
- The region id does not exist.
- The user does not have the appropriate permissions to modify the region attributes.

REMARKS: Nil

3.1.23 NAME: Create_Node

PURPOSE: To create a new and empty node within the HyperDoc and assign it to a network.

INPUT: Node attributes
Network id

ACTION:
- Create new node in the HyperDoc.
- The HyperDoc assigns default attribute values to the new node; its version number will be assigned to 1.
- Assign node to the specified network.
- The node lock is false.

RESULT:
- New node is created.
- A unique node id is returned.

EXCEPTIONS:
- The user has unauthorized capability to create new node.
- Network id does not exist.

REMARKS: The default values of the node were described in detail in section 2.

3.1.24 NAME: Delete_Node

PURPOSE: To delete the node from the network.

INPUT: Node id
Current network

ACTION:
- If there exist links within the same network associated to the node, the HyperDoc informs the user about these links.
- The HyperDoc prompts the user to confirm his/her decision so that the user can undo the command and manipulate these links manually.
- If the user confirms to delete the node while there are links associated to the node then the HyperDoc deletes all links and logically remove the node from the context of the network.
- If there exist in-coming links which reference the node from the other network context, then the HyperDoc forces versioning.
- Update the modification date of the networks affected by this function.

RESULT: The node is deleted from the context of the specified network.

EXCEPTIONS:
- The user has unauthorized capability to delete a node.
- Node is locked.

REMARKS: Nil

3.1.25 NAME: Split_Node

PURPOSE: To split the node into two.

INPUT: Node id
Position within the node
Current network

ACTION:
- Create new and empty node to the HyperDoc and assign it to the current network.
- The HyperDoc assigns the default attribute values to the new node.
- Copy content from second part starting from the given position of the original node to the newly created node.
- Remove the second half of the original node content.
- If there exist links within the current network associated to the original node, the HyperDoc informs the user about these links.
- The HyperDoc prompts the user to confirm his/her decision so that the user can undo the command and manipulate these links manually.
- If the user confirms to delete the node while there are links associated to the node then the HyperDoc deletes all links and logically removes the node from the context of the network.
- If there exist in-coming links which reference the node from the other network context, then the HyperDoc forces versioning.
- Update the modification date of networks affected by this function.

RESULT:
- Original node is split into two.
- A unique node id of the newly created node is returned.

EXCEPTIONS:
- The user has unauthorized capability to split the node.
- Cursor position is in the area within a region.
- Node is locked.

REMARKS: The user may require to update the links associated to the nodes in order to correct the semantics of those links.

3.1.26 NAME: Insert_Node

PURPOSE: To insert content of node (1) to node (2).

INPUT:
Node id (1)
Node id (2)
Position within node (2)
Current network

ACTION:
- Content of node (1) is inserted after the given position within node (2).
- For links previously associated to the node (1) in the current network context, they will be updated to the corresponding destination in node (2).
- Delete node (1) from the current network.
- If there exist in-coming links which reference the node from the other network context, then the HyperDoc forces versioning.
- Update the modification date of nodes and networks affected by this function.

RESULT:
- Content of node (1) becomes part of node (2).
- Node (1) is deleted from the current network.
- Node is locked.

EXCEPTIONS:
- The user has unauthorized capability to insert the node.
- Node id of node (1) is identical to that of node (2).

REMARKS: The user may require to update links in the current network to keep the links' semantic correct.

3.1.27 NAME: Save_Node_Under_Current_Version

PURPOSE: To save the node to the permanent store under current version.

INPUT: Node id
Current network
Node version

ACTION:
- If region(s) has been changed within the context of the current network and there exist in-coming link(s) which references the region from other network context, then the HyperDoc forces versioning, otherwise the HyperDoc save the node onto the permanent store.
- Updates the modification dated of networks affected by this function.

RESULT: The permanent store of the node is updated under current version or new node version will be created and saved.

EXCEPTIONS:
- The user has unauthorized capability to save the node.
- There is not enough space in the permanent store.
- Node is locked.

REMARKS: Nil

3.1.28 NAME: Assign_Node_To_Network

PURPOSE: To assign the node to a specified network.

INPUT: Node id
Network id

ACTION:
- The node is input into the network.
- Updates the last modified dated to the network.

RESULT: The node is in the context of the network.

EXCEPTIONS:
- The user has unauthorized capability to assign the node to the network.
- Node id or network id is invalid.

REMARKS: Nil

3.1.29 NAME: Create_Region

PURPOSE: To create a region in the specified node.

INPUT: Node id
Selected area within the node
Region attributes

ACTION:
- Create a region within the specified node.
- The HyperDoc assigns the default attribute values to the region.

RESULT:
- The region is created within the specified node.
- Region_Id is returned.

EXCEPTIONS: The user has unauthorized capability to create new region in the specified node.

REMARKS: Regions may overlap partly or completely within the node.

3.1.30 NAME: Delete_Region

PURPOSE: To delete a region from the specified node.

INPUT: Node id
Region id
Current network

ACTION:
- If there exist links within the current network associated to the region, the HyperDoc informs the user about these links.
- The HyperDoc asks the user to confirm his/her decision so that the user can undo the command and manipulate these links manually.
- If the user confirms to delete the region while there are links associated to the region then the HyperDoc deletes all theselinks and removes the region.
- If there exist in-coming links which reference the region from the other network context, then the HyperDoc forces versioning.
- Update the modification date of networks affected by this function.

RESULT: The specified region is removed from the node.

EXCEPTIONS: The user has unauthorized capability to delete region from the specified node within the network.

REMARKS: Nil

3.1.31 NAME: Create_Link

PURPOSE: To create a new link from a departure to a destination within the specified network.

INPUT:
Current network
Departure id
Destination id
Link attributes

ACTION:
- The HyperDoc assigns the default attribute values to the link.
- Connects the link from the departure point to the destination point.
- Assign the link to the current network.
- Update the last modified date of the current network.

RESULT:
- A link is created in the current network.
- A unique link id is returned.

EXCEPTIONS:
- The user has unauthorized capability to create link in the specified network.
- Departure or destination is undefined within the context of the current network.

REMARKS: Departure/destination can be node or regions.

3.1.32 NAME: Delete_Link

PURPOSE: To delete a link within the specified network.

INPUT: Current network
Link id

ACTION:
- Remove the link from the current network.
- Update the last modified date of the network.

RESULT: The link is deleted.

EXCEPTIONS: The user has unauthorized capability to delete the link within the current network.

REMARKS: Nil

3.1.33 NAME: Create_Network

PURPOSE: To create a new and empty network to the hyperdocument.

INPUT: Network attribute

ACTION:
- New and empty network in the HyperDoc.
- The HyperDoc assigns default attributes value to the network.

RESULT:
- New network is created in the HyperDoc.
- A unique network id is returned.

EXCEPTIONS: The user has unauthorized capability to create new network.

REMARKS: Nil

3.1.34 NAME: Delete_Network

PURPOSE: To delete the network from the HyperDoc.

INPUT: Network id

ACTION:
- All links and nodes within the network are deleted.
- Remove the specified network from the HyperDoc.

RESULT: The network is removed from the Hypertext.

EXCEPTIONS:
- The user has the unauthorized capability to delete the network.
- Network id is undefined.

REMARKS: Nil

3.1.35 NAME: Create_Organization

PURPOSE: To create a new/empty organization within a network.

INPUT: Network id
Organization type
Organization attributes

ACTION:
- Create a new and empty organization within the specified network.
- The HyperDoc assigns the default value of attributes to the organization.
- Update last modified date of the specified network.

RESULT:
- The new/empty organization is created.
- A unique organization id is returned.

EXCEPTIONS:
- The user has unauthorized capability to create new organization to the network.
- Network id is undefined.

REMARKS: The user may select one of the Hypertext defined type (SET, SEQUENCE, MAP) or his/her own defined type.

3.1.36 NAME: Delete_Organization

PURPOSE: To delete an organization from the network.

INPUT: Organization id
network id

ACTION:
- Remove the organization description (content).
- Remove the organization from the network.

RESULT: The specified organization is deleted.

EXCEPTION:
- The user has unauthorized capability to delete the organization.
- Network id is undefined.

REMARKS: Nil

3.1.37 NAME: Invoke_HyperDoc

PURPOSE: To invoke the HyperDoc from the host operating system.

INPUT: User id
Password

ACTION: Execute the HyperDoc system.

RESULT: Hyperdocument.

EXCEPTIONS: The user id or password is undefined.

REMARKS: Nil

3.1.38 NAME: Shutdown_HyperDoc

PURPOSE: To quit from the HyperDoc.

INPUT: Current HyperDoc

ACTION: Quit from HyperDoc.

RESULT: The user is out of the HyperDoc and back to the host operating system.

EXCEPTIONS: Nil

REMARKS: Nil

3.1.39 NAME: Show_Network_List

PURPOSE: To display the list of all networks to the user.

INPUT: Nil

ACTION: Display the list of all networks in an information window.

RESULT: A list of all networks in the hyperdocument is displayed on in an information window.

EXCEPTIONS: Nil

REMARKS: Nil

3.1.40 NAME: Show_List_Of_Organizations_Within_A_Network

PURPOSE: To display the list of organizations defined for a given network.

INPUT: A network id
A user id

ACTION: Display a list of organizations defined for the given network in an information window.

RESULT: A list of organizations that the user can activate is displayed on the screen.

EXCEPTIONS: Nil

REMARKS: The user is already within the context of a network for which he has the appropriate access rights.

3.1.41 NAME: Select_Network

PURPOSE: To allow the user to activate a network.

INPUT: A network id
A user id

ACTION: Change the currently active network to the selected network.

RESULT: The currently active network becomes the selected network.

EXCEPTIONS:
- The network id does not exist in the system.
- The user does not have access rights for the given network.

REMARKS: When a network is activated, the user can access only those nodes and organizations that are defined for that network.

3.1.42 NAME: Move_To_Next_Node

PURPOSE: To display the contents of the next node in an organization of type sequence to the user.

INPUT: An organization id
A network id
The current node id
The current navigation path

ACTION:
- If the current node contains unsaved changes, confirm with the user to move or not.
- If the user wants to move,
  - include the current node identifier in the current navigation path,
  - display the latest version of the next node in the sequence in a new node window with the cursor positioned at the beginning of the node, and
  - display the node identifier in the window header.

RESULT: The content of the next node in the sequence is displayed in a new node window.

EXCEPTIONS:
- The organization type is not a sequence type.
- Unauthorized user permission.

REMARKS: The user is already within the context of a network and organization for which he has the appropriate rights.

3.1.43 NAME: Save_Node_Under_New_Version

PURPOSE: To save the contents of the node with a new version number.

INPUT: A node id
A set of node attributes

ACTION:
- Save the content of the node on secondary storage with a new version number.
- Increment the version number in the attribute list of the given node.
- Remove all incoming links to the new node that were created by the author, and keep all incoming and outgoing links to the old node.
- Update the appropriate link attributes that have been affected by this operation.

RESULT:
- The content of the node is written into a new file on the storage device.
- The version attribute of the node is incremented.
- Those attributes that are maintained by HyperDoc are updated.

EXCEPTIONS:
- Insufficient storage space exists on the storage device.
- The user does not have the security permission to save changes to a node.

REMARKS: Nil

3.1.44 NAME: Retrieve_Previous_Version_Of_A_Node

PURPOSE: To retrieve and display the contents of a previous version of a given node.

INPUT: A node id
A version number

ACTION:
- Retrieve the given version of the node from the storage device.
- Display the contents of the node on the screen in a node window.
- Update the window header to display the identification of the node.

RESULT: A node window is displayed that contains the contents of the node.

EXCEPTIONS:
- The version number does not exist for the given node.
- The node id does not exist in the hyperdocument.
- The user does not the appropriate access permissions.

REMARKS: Nil

3.1.45 NAME: Create_User_Class

PURPOSE: To create a new user class and add it to the user class list.

INPUT:
User Class Id
User Class Name
User Class Description
User Class List

ACTION:
- Create the new user class and add it to the user class list.
- Inform the SUPER user of the results of the operation.

RESULT: New user class list.

EXCEPTIONS:
- A user class list does not exist.
- The user class id already exist in the user class list.
- The creator is not the SUPER user.

REMARKS: The user class description attribute captures information pertaining to the role that the user class plays within the context of the software engineering process. User classes should be unique (ie. unique user class id).

3.1.46 NAME: Delete_User_Class

PURPOSE: To remove the definition of a user class from the user class list.

INPUT: User Class Id
User Class List

ACTION:
- Delete the user class from the user class list.
- Inform the SUPER user of the results of the operation.

RESULT: New User class list.

EXCEPTIONS:
- The user class to be deleted is not in the user class list.
- The set of users belonging to that class is not empty.
- The user attempting the deletion is not the SUPER user.

REMARKS: Nil

3.1.47 NAME: Update_User_Class

PURPOSE: To modify the name and/or description of a user class.

INPUT: User Class Id
New Name and/or New Description
User Class List

ACTION:
- Replace the old name and/or description of user class with new values.
- Inform the user SUPER of the results of the operation.

RESULT: New modified user class.

EXCEPTIONS:
- The user class is not in the user class list.
- The new values are not different than the old ones.
- The user attempting to modify any of the attributes of the user class is not SUPER user.

REMARKS: A user class id cannot be modified, only created or deleted.

3.1.48 NAME: Create_User

PURPOSE: To create a new user and add him/her to the user list.

INPUT:
User Id
User Name
User Password
User Description
User List

ACTION:
- Create the new user and add him/her to the user list.
- Inform SUPER user of the results of the operation.

RESULT: New user list.

EXCEPTIONS:
- The user list does not exist.
- The User Id already exist in the user list.
- The creator is not the SUPER user.

REMARKS: The user description attribute captures information pertaining to the role that the user plays within the context of one or more of the software engineering processes. User Id's should be unique.

3.1.49 NAME: Delete_User

PURPOSE: To remove a user from the user list.

INPUT:
User Id
User List
Set of pairs of <object type, object id> of user objects
Set of user classes to which a user belongs
Another user's Id

ACTION:
- Allocate all objects rights of deleted user to another user.
- Delete user from all classes that appear in the set of user classes.
- Delete the user from the user list.
- Inform the SUPER user of the results of the operation.

RESULT: New User list. All objects rights of deleted user transferred to another user.

EXCEPTIONS:
- The user to be deleted is not in the user list.
- The user attempting this operation is not the SUPER user.

REMARKS: Nil

3.1.50 NAME: Update_User

PURPOSE: To modify the name, password and/or description of a user.

INPUT: User Id
New Name and/or New password and/or New Description
User Class List

ACTION:
- Replace the old name and/or password and/or description of user with new values.
- Inform the user of the results of the operation.

RESULT: New modified user attributes.

EXCEPTIONS:
- The user is not in the user class list.
- The new values are different than the old ones.
- User attempting modification is not the SUPER user or the user himself.

REMARKS: A user id cannot be modified, only created or deleted.

3.1.51 NAME: Assign_User_To_Class

PURPOSE: To make a user a member of a user class.

INPUT:
User Id
User List
User Class Id
User Class List
User-Class Relation List

ACTION:
- Add the <user id, user class id> relation in the user-class relation list.
- Inform the SUPER user of the results of the operation.

RESULT: New user-class relation list.

EXCEPTIONS:
- The user is not in the user list.
- The user class is not in the user class list.
- The <user id, user class id> relation already exists in the user-class relation list.
- Another user besides SUPER user attempts to perform this operation.

REMARKS: Nil

3.1.52 NAME: Delete_User_From_Class

PURPOSE: To revoke a user's membership from a user class.

INPUT:
User Id
User List
User Class Id
User Class List
User-Class Relation List

ACTION:
- Remove the <user id, user class id> relation from the user-class relation list.
- Inform the SUPER user of the results of the operation.

RESULT: New user-class relation list.

EXCEPTIONS:
- The user is not in the user list.
- The user class is not in the user class list.
- The <user id, user class id> relation does not already exist in the user-class relation list.
- This is the only class that the user belongs to.
- It is not the SUPER user performing this operation.

REMARKS: Nil

3.1.53 NAME: Assign_Object_And_Rights_To_Manager

PURPOSE: To make a user or a user class eligible to access an object with certain access privileges.

INPUT:
User Id
User List (or User Class Id, User Class List)
Object type Object Id
Manager-Object Relation List
Access rights set (ie read, write, create, delete, etc...)

ACTION:
- Add the <user id, object id, access rights set> or <user class id, object id, access rights set> relation in the manager-object relation list.
- Inform the SUPER user of the results of the operation.

RESULT: New manager-object relation list.

EXCEPTIONS:
- The user or user class is not in the user or user class list.
- The <user id, object id> or <user class id, object id> relation does not already exist in the manager-object relation list.
- It is not the SUPER user performing this operation.

REMARKS: The notation <user id, object id, access rights set> associates a user with an object in the way specified by the access rights set.

3.1.54 NAME: Delete_Object_And_Rights_From_Manager

PURPOSE: To revoke a user's or a user class access to an object which they used to access with certain access privileges.

INPUT:
User Id
User List (or User Class Id, User Class List)
Object type
Object Id
Manager-Object Relation List

ACTION:
- Delete the <user id, object id, access rights set> or <user class id, object id, access rights set> relation from the manager-object relation list.
- Inform the SUPER user or class leader of the results of the operation.

RESULT: New manager-object relation list.

EXCEPTIONS:
- The user or user class is not in the user or user class list.
- The <user id, object id> or <user class id, object id> relation does not already exist in the manager-object relation list.
- It is not the SUPER user or the class leader performing this operation.

REMARKS: Nil

3.1.55 NAME: Update_Object_Rights_Of_Manager

PURPOSE: To update a user's or a user class access to an object which they have access to already with certain access privileges.

INPUT:
User Id
User List (or User Class Id, User Class List)
Object type
Object Id
Manager-Object Relation List
New access privileges set (ie. read, write, etc...)

ACTION:
- Replace the old access rights set with the new ones.
- Inform the SUPER user or class leader of the results of the operation.

RESULT: New manager-object relation list.

EXCEPTIONS:
- The user or user class is not in the user or user class list.
- The <user id, object id> or <user class id, object id> relation already exist in the manager-object relation list.
- The old and new set of privileges are the same.
- It is not the SUPER user or the class leader performing this operation.

REMARKS: Nil

3.1.56 NAME Show_Object_Access_Rights

PURPOSE: To show the access rights that a user has on an object.

INPUT: User Id
Object id

ACTION: Display allocated rights to user.

RESULT: The object rights for the user.

EXCEPTIONS: Nil

REMARKS: Nil

3.1.57 NAME: Select_Organization

PURPOSE: To allow the user to activate an organization within a network.

INPUT: The current network
An organization id
A user id

ACTION: Change the currently active organization to the selected organization.

RESULT: The currently active organization becomes the selected organization.

EXCEPTIONS:
- The organization id does not exist in the current network.
- The user does not have access rights for the given organization.

REMARKS: When a organization is activated, the user can access only those nodes that are defined for that organization.

3.1.58 NAME: Display_Region_Attributes

PURPOSE: To inform the user about the attributes and their values assigned to the given region.

INPUT: Region identifier
Node identifier
Current network

ACTION: Display in an information window, the list of the region's attribute labels as well as their value.

RESULT: A new information window is displayed on the screen and it contains the list of attributes assigned to the region.

EXCEPTIONS: The region does not exist in the current network.

REMARKS: Nil

3.1.59 NAME: Display_Assigned_Object_Ids

PURPOSE: To provide the user with a list of object ids, having the given object attribute, to which he has full access rights.

INPUT: Author id
Object attribute

ACTION: Display in an information window, the list of the object ids of all objects containing the given object attribute in their attribute lists.

RESULT: A new information window is displayed on the screen and it contains the list of object ids selected.

EXCEPTIONS: Nil

REMARKS: An object attribute is an attribute defined for the object. For example, a user can list all the issues that he has been asked to respond to by specifying that the object attribute (in this case, a node attribute) be of type issue.

3.1.60 Future Requirements

a. garbage collection facility to collect objects that are not used in the system;

b. versioning at the network level;

c. importation of information created outside HypderDoc;

d. facilities for printing; and

e. pattern search capabilities.

## 3.2 Overall Performance Requirements

### 3.2.1 Timing Constraints

HyperDoc is an interactive tool. The following constraints are required for the timely execution of user commands:

a. the response time to any user command must be no longer than 30 seconds;

b. the overview diagram must be updated after a reasonable number of modifications have been made to the structure of the hyperdocument;

c. operations performed by the SUPER user must take effect instantaneously; and

d. performance of HyperDoc must not degrade below the above stated timing constraints when the system load increases.

## 3.3 Attributes

### 3.3.1 Availability

HyperDoc should be available at all times to at most 50 users. In case of a failure internal to HyperDoc appropriate recovery procedures can be activated to repair the damage, if possible. In case of a host operating system failure, restart and consistency checking procedures exist.

### 3.3.2 Security

There are three levels of security, host-user level, HypderDoc-user level and HyperDoc-object level. For details on HyperDoc security refer to section 2.1.6

## 3.4 Acceptance Criteria

The product, HypderDoc will be accepted if the following criteria are satisfied:

a. functional Requirements are satisfied;

b. the Timing Constraints are satisfied;

c. an average user can easily learn HyperDoc;

d. HyperDoc should be easily extendible and maintainable;

e. HyperDoc should be reliable;

f. HyperDoc should be user-friendly; and

g. a user manual should be provided, which must be consistent with the functionalities in HyperDoc.

# Appendix D

# The infoMap for the HyperDoc SRD

| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | A | A | 1 | { IDENTIFICATION } | |
| 2 | v | v | v | v | v | v | v | | | 1. infoSchema for the HyperDoc SRD. |
| 3 | A | A | A | A | A | A | A | 8 | { REFERENCE } | |
| 4 | A | A | A | A | A | A | A | 6 | { VIEW } | |
| 5 | v | . | . | . | . | . | . | | | 1. Definitions |
| 6 | . | v | . | . | . | . | . | | | 2. Product Perspective |
| 7 | . | . | v | v | . | . | . | | | 3. Product Functions |
| 8 | . | . | . | . | v | v | . | | | 4. Functional Requirements |
| 9 | . | . | . | . | . | . | v | | | 5. Product Users |
| 10 | . | . | . | . | . | . | . | v | | 6. Set Cardinality |
| 11 | O | . | . | . | . | . | . | 29 | { DEFINITION } | |
| 12 | T | . | . | . | . | . | O | 6 | { USER TYPE } | |
| 13 | . | . | . | . | H | O | . | 12 | { OPERATION } | |
| 14 | . | H | . | . | . | . | . | 1 | { EXTERNAL OBJECT } | |
| 15 | T | H | O | O | . | F | M | 45 | { INTERNAL OBJECT } | |
| 16 | . | . | G | . | . | . | . | 11 | { PRE-GUARD } | |
| 17 | . | . | . | . | . | G | . | 15 | { PRE-CONDITION } | |
| 18 | . | . | . | M | H | . | . | 11 | { TASK } | |
| 19 | . | . | . | . | . | S | . | 19 | { ACTION } | |
| 20 | . | . | . | . | . | G | . | 8 | { POST-CONDITION } | |
| 21 | . | . | G | . | . | G | . | 27 | { POST-GUARD } | |
| 22 | . | . | . | M | . | M | . | 5 | { REMARK } | |

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | AA | AB | AC | BS | BT | SU |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | 1 | | { IDENTIFICATION } |
| 2 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 1 InfoMap for the HyperDoc SRD |
| 3 | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | | 8 | { REFERENCE } |
| 4 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 1 IEEE Std 830-1984 |
| 5 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 2 SRD for a hyperdocument system |
| 6 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 3 V S Alagar, P. Goyal, "CONSLT· A Software Life Cycle Tool", |
| 7 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Concordia University, Montreal, Canada. |
| 8 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 4. Pankaj K. Garg and Walt Scacchi, "ISHYS Designing an |
| 9 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Intelligent Software Hypertext System", University of |
| 10 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Southern California, IEEE Expert, Fall 1989. |
| 11 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 5. Danny B. Lange, "A formal Approach to Hypertext using |
| 12 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Post-Prototype Formal Specification", Department of |
| 13 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Computer Science, Technical University of Denmark. |
| 14 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 6 Jacob Nelson, Hypertext and Hypermedia, Academic Press, 1990. |
| 15 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 7 Pankaj K. Garg and Walt Scacchi, "A Hypertext System to manage |
| 16 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Software Life-Cycle Documents", IEEE Software, May 1990. |
| 17 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 8 Jeff Conklin, "Hypertext - An Introduction and Survey", Computer, |
| 18 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Microelectronics and Computer Technology Corp., Computer, |
| 19 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | September 1987. |
| 20 | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | | 6 | { VIEW } |
| 21 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 1. Definitions |
| 22 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 2 Product Perspective |
| 23 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 3. Product Functions |
| 24 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 4. Functional Requirements |
| 25 | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 5 Product Users |
| 26 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | v | 6. Set Cardinality |
| 27 | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | | 29 | { DEFINITION } |
| 28 | o | . | . | . | . | . | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | | | | 1. Node or region in a node which can be selected by the user in order to |
| 29 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | to use a link. The anchor selected is the departure point of a link. |
| 30 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Anchors are created within a node by authorized users. |
| 31 | | o | . | | | | | . | | | | | | | | . | | . | | | . | | . | | | | | | | | | 2 An attribute is a pair (label, value) existing for nodes, regions, |
| 32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | networks and organizations used to described them. Each of the previous |
| 33 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | has some fixed labels, and can support an unlimited number of |
| 34 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | user-defined attribute pairs The labels must be unique. |
| 35 | | | o | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | . | | | 3 A place marker in the node window. |
| 36 | . | . | . | o | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 4 The source point of a link. It can be a node or a region in a node. |
| 37 | . | . | . | . | o | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | . | | | 5 The destination in a hyperdocument that a link leads to It can be a node |
| 38 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | or a region in a node |
| 39 | . | . | | | . | o | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | | 6 The entire HyperDoc information space (contents of all nodes and networks). |
| 40 | | | | | . | . | o | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | | | 7. Window used to display any information about HyperDoc's objects. |
| 41 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | The user can only browse the information and is not allowed to change it. |
| 42 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | An information window contains a header with its name. |
| 43 | | | | | . | | . | o | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | | | | | | 8 Means of relating two pieces of information. A link is defined by its |
| 44 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | departure point and its destination point. The link provides a way for |
| 45 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | moving from one point to another. |
| 46 | . | . | . | | | | . | | o | . | | | | . | . | . | . | . | . | . | . | . | . | | | | | | | | | 9. The list of attribute labels minimally required for links is· id, departure |
| 47 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | point, destination point, author, last modified date, type, description. |
| 48 | | | | | | | | . | . | o | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | | | 10 An attribute of the node which when set to true by a user disables the |
| 49 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | other users from chnaging the content of the node. When this |
| 50 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | attribute is false no users are modifying the node. |
| 51 | | . | | | | | . | . | . | . | o | . | . | | . | . | . | . | . | . | . | . | . | . | | | | | . | | | 11 Similar to an index, where a list of user defined keywords map into nodes |
| 52 | | | . | | | | . | . | . | . | . | o | . | . | | | | . | . | | . | . | . | . | | | | | | | | 12 The ordered list of nodes visited by the user in the current network. |
| 53 | | | | | | . | . | . | . | . | . | . | o | . | . | | | . | . | . | . | . | . | . | . | | | | | | | 13 Set of nodes and links which have some logical association. A certain |
| 54 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | concept can be captured within a network. A network can have several |
| 55 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | organizations |
| 56 | | | | | | | . | . | | | . | . | | o | . | | | . | . | . | . | . | . | | . | | | | | | | 14 The list of attribute labels minimally required for networks is id, |
| 57 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | author, last modified date, description. |
| 58 | | . | | | . | | | | | . | . | . | | | o | | | | . | . | | | . | . | . | | | | . | | | 15 A piece of information of a specific medium. It is the fundamental unit |
| 59 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | of HyperDoc nodes are represented as node windows on the screen. |
| 60 | | | | | | . | | | | . | . | . | | . | . | o | | . | . | | | . | | . | . | | | | | | | 16. The list of attribute labels minmally required for nodes is. id |
| 61 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | description, author, date, type, last modified date, version number, lock. |
| 62 | . | | | | | | | | . | . | . | . | . | . | . | . | o | | . | . | . | . | | | | . | | | . | | | 17. A means of displaying a node on the screen A node window is provided |
| 63 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | the appropriate tool to edit or browse the content of a node. A node |
| 64 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | contains a header with the node's id. |
| 65 | | | | | | | . | | | | . | | | | . | . | . | o | . | | | | | | . | | | | | | | 18 An arrangement of nodes of a network as a set, a map or a sequence |
| 66 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | An organization can only be one of the above |
| 67 | | | | | | | . | . | . | . | . | | . | . | | | | . | o | | . | . | | | . | | | | . | | | 19 The list of attribute labels minimally required for organizations is: id |
| 68 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | author, creation date, type. |
| 69 | | | | | | | | . | | | . | | | | . | | . | . | . | o | | . | | | | | | | | | | 20 A bird's eye view of the network with the current navigation path |
| 70 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | highlighted |
| 71 | | | | | | | | | | | | | | | | | | | . | | o | . | | | . | | | | . | | | 21. A section in a node of contiguous text, voice, video, grphics, etc... |
| 72 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | defined by the user. A region serves as an endpoint of a link A region |
| 73 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | can be of any size. |
| 74 | | | | | | | | | . | | . | | | | | | | | . | | | o | . | | | | | | | | | 22. The list of attribute labels minimally required for regions is: id |
| 75 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | parent node, description, type. |
| 76 | | | | | | | | | . | | | | | | . | . | | | | | . | | o | . | . | | | | | | | 23. A user defined ordering that is composed on some node in a network |
| 77 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | i.e the nodes can be access in this order |
| 78 | | | | | | | | | | | | | | | | | . | | | | | | | o | | | | | | | | 24. A user defined collection of nodes that may be accessed in any order. |
| 79 | | | | | | | | | | | | | | | | | | | | | | | | | o | | | | | | | 25 A user with full rights at all time in HyperDoc system |
| 80 | | | | | | | | | | | | | | | | | | | | | . | | | . | | o | | | | | | 26 A person belonging to a least one user class |
| 81 | | | | | | | | | | | | | | | | | | | | | | | | | | | o | | | | | 27 Describes the role of people and defines the scope of their work within |

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | AA | AB | AC | DS | BT | SU |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 82 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | the software life cycle context. Contains a list of users assigned to it |
| 83 | . | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | o | | | | 28 A supervisor defined for a user class having the authority to manage |
| 84 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | the rights of users in that user class |
| 85 | | | | | | | | . | . | . | | . | | . | . | | | | . | . | | | | | | . | | | o | | | 29 The set of all networks |
| 86 | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | 6 | | {USER TYPE} |
| 87 | . | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | v | . | . | . | . | | | 1 Super user |
| 88 | . | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 2 Builder |
| 89 | . | . | | . | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 3. Consulter |
| 90 | . | . | . | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | v | | | | 4 User leader |
| 91 | . | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | | | 5 Class leader |
| 92 | . | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | | | 6 Author |
| 100 | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | T | 45 | | {INTERNAL OBJECT} |
| 109 | | | | | | . | | . | . | | | | . | | . | . | | | | . | . | . | . | | | | | . | v | | | 1. HyperDoc |
| 110 | . | | . | | | v | | . | . | . | | | . | . | . | | | | . | | | . | . | . | | | | . | | | | 2 Hyperdocument |
| 111 | | . | | | | | . | . | . | . | . | . | v | . | . | . | . | | | | . | | | | | . | . | . | . | | | 3 Network |
| 112 | . | | | | . | . | . | . | . | . | | | | . | . | . | . | | . | | | . | | | | . | | | | | | 4 Network identity |
| 113 | | | | | | | . | . | | . | . | | . | v | . | . | . | . | | . | . | | . | | | . | . | . | . | | | 5 Network attributes |
| 114 | . | | | | . | | . | . | | | . | | . | | . | | | v | . | | | | | | . | . | . | | | | | 6 Organization |
| 115 | . | | | | . | . | . | v | . | . | | . | | . | . | . | . | . | . | | | . | . | . | | | . | . | . | | | 7 Link |
| 116 | . | . | | | | . | | . | | | . | . | | . | v | . | . | . | . | . | . | | . | . | . | | | | . | | | 8 Node |
| 117 | . | | | | | . | | . | . | . | | . | | . | . | | . | | v | . | . | | | . | | | | . | . | | | 9 Organization attributes |
| 118 | . | . | . | | | . | | . | . | v | | . | | . | . | . | . | . | . | | | . | . | | . | | . | | . | | | 10 Link attributes |
| 119 | . | | | | | . | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | | | 11 Link name |
| 120 | . | . | | | | . | | . | . | . | . | . | | . | . | . | . | . | . | . | v | | . | . | . | . | . | . | . | | | 12 Region |
| 121 | . | . | | | | . | | . | . | . | . | . | | . | | v | . | . | . | . | . | | . | . | . | . | . | . | . | | | 13 Node attributes |
| 122 | . | . | | | | . | | . | . | . | | . | | . | . | . | . | . | . | . | . | v | . | . | . | . | . | | | | | 14 Region attributes |
| 123 | | . | | . | . | . | . | . | . | v | | . | | . | . | . | . | . | . | . | . | | . | . | . | . | . | | | | | 15. Lock attribute |
| 124 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | v | . | . | . | . | . | . | . | . | . | . | | | | | 16 Node window |
| 125 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 17 Node window header |
| 126 | . | . | | v | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | | | 18 Cursor position |
| 127 | . | | | . | . | . | v | . | . | . | . | . | | . | . | . | . | . | . | | | . | . | | | . | . | . | . | | | 19 Information window |
| 128 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | | 20 Information window header |
| 129 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | v | . | | | | 21. User class |
| 130 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 22. User class id |
| 131 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 23 User class name |
| 132 | . | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 24 User class description |
| 133 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 25. User class list |
| 134 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | v | . | | | | | 26 User |
| 135 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 27 User id |
| 136 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 28. User name |
| 137 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 29 User password |
| 138 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | . | | | 30. User description |
| 139 | | . | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | . | | | 31. User list |
| 140 | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | . | | | 32 Object type |
| 141 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 33. Object id |
| 142 | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 34. User-class relation list |
| 143 | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 35. Manager-object relation list |
| 144 | . | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 36. Access rights set (Read, Write, Delete, etc ) |
| 145 | v | . | . | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 37. Anchor |
| 146 | . | v | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 38 Attribute |
| 147 | . | . | . | v | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 39 Departure point |
| 148 | | . | | . | v | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 40. Destination point |
| 149 | . | . | . | . | . | . | . | . | . | . | . | v | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 41. Map |
| 150 | | . | . | | | . | . | . | . | . | . | . | v | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | | | 42. Navigation path |
| 151 | . | . | . | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | v | . | . | . | . | . | . | . | . | . | | | 43 Overview diagram |
| 152 | | | | | | . | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | v | . | . | . | . | . | . | | | 44 Sequence |
| 153 | . | | | | | | | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | v | . | . | . | . | . | | | 45 Set |

| | AC | AE | AF | AG | AH | AI | AJ | AK | AL | AM | AN | AO | AP | AQ | AR | AS | AT | AU | AV | AW | AX | AY | BS | BT | BU |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | 1 | | { IDENTIFICATION } |
| 2 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 1 InfoMap for the HyperDoc SRD |
| 3 | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | 6 | | { REFERENCE } |
| 4 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 1 IEEE Std 830 1984 |
| 5 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 2 SRD for a hyperdocument system |
| 6 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 3 V S Alagar, P Goyal, "CONSLT A Software Life Cycle Tool", Concordia University, Montreal, Canada |
| 7 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 4 Pankaj K Garg and Walt Scacchi, "ISHYS Designing an Intelligent Software Hypertext System", University of Southern California, IEEE Expert, Fall 1989 |
| 9 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 5 Danny B Lange, "A formal Approach to Hypertext using Post Prototype Formal Specification", Department of Computer Science, Technical University of Denmark |
| 12 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 13 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 14 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 6 Jacob Nelson, Hypertext and Hypermedia, Academic Press, 1990 |
| 15 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 7 Pankaj K. Garg and Walt Scacchi, "A Hypertext System to manage Software Life-Cycle Documents", IEEE Software, May 1990 |
| 16 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 17 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 8 Jeff Conklin, "Hypertext - An Introduction and Survey", Computer, Microelectronics and Computer Technology Corp , Computer, September 1987 |
| 18 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 19 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 20 | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | 6 | | { VIEW } |
| 21 | | | | | | | | | | | | | | | | | | | . | | | | | | 1. Definitions |
| 22 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | . | | | | 2 Product Perspective |
| 23 | . | | . | | | | | | | | | | | | | . | | | | v | v | v | | | 3 Product Functions |
| 24 | | | | | | | | | | | | | | . | . | . | | | | | | . | | | 4 Functional Requirements |
| 25 | | . | | | | | | | . | | | | | | | | | | | | | | | | 5 Product Users |
| 26 | | | | | | | | | | | | | | | | | . | | | | | . | | v | 6 Set Cardinality |
| 106 | H | H | H | H | H | H | H | H | H | H | H | H | H | H | H | H | H | H | H | . | | | 1 | | { EXTERNAL OBJECT } |
| 107 | p | . | | | | | | | | . | | | | | | . | . | | | | | | | | 1 Unix operating system |
| 108 | H | H | H | H | H | H | H | H | H | H | H | H | H | H | H | H | H | H | H | O | O | O | 45 | | { INTERNAL OBJECT } |
| 109 | c | . | | | | | | | | | | | | | | . | | | | o | o | o | | | 1 HyperDoc |
| 110 | | p | | | | | | | | | . | | | . | | | . | | | | | | | | 2 Hyperdocument |
| 111 | | c | p | | | | | | | | | . | | | . | . | | . | | | | | | | 3 Network |
| 112 | . | | c | | . | | | | | | | | . | | | | | | | | | | | | 4 Network identity |
| 113 | . | | c | | | | | | . | | | | | . | | | | | | | | . | | | 5 Network attributes |
| 114 | . | | o | p | | | | | | | | | | | . | | . | | . | | | . | | | 6 Organization |
| 115 | . | . | c | | p | . | . | | | . | . | | . | | | | | . | | | | | | | 7 Link |
| 116 | . | . | c | o | | p | | | | . | | | | . | | | | | | . | . | . | | | 8 Node |
| 117 | | | . | c | . | | | | . | | | | | | . | | | | | | | . | | | 9 Organization attributes |
| 118 | | | . | . | c | | . | . | . | | | | | | | . | . | | . | | . | | | | 10 Link attributes |
| 119 | . | | | | c | | | | . | . | | | . | | | | | | . | | . | . | | | 11 Link name |
| 120 | | | | | . | c | p | | | | | . | | | . | | . | | | | | | | | 12 Region |
| 121 | | | . | | . | c | | p | | | . | . | | | . | | | | | . | . | | | | 13 Node attributes |
| 122 | . | | | | | | c | | | | | | | . | | | | | | . | . | | | | 14 Region attributes |
| 123 | | | | | | | | c | | | | | | . | | | | | . | | | | | | 15 Lock attribute |
| 124 | | | | | | | | | p | | | | . | | | | . | | | . | | | | | 16 Node window |
| 125 | . | | | | . | | . | | c | | | . | | . | | | | | | . | . | . | | | 17 Node window header |
| 126 | . | | | | . | . | | | c | | | | | | | | | | . | | . | | | | 18 Cursor position |
| 127 | | | | | | | | | | p | | | | | | | | | | | . | . | | | 19 Information window |
| 128 | . | | | | | | . | | | c | . | | . | . | . | . | . | . | . | . | . | . | | | 20 Information window header |
| 129 | | | | . | | . | . | | | | p | . | | . | . | | | | | | | | | | 21 User class |
| 130 | | | | | | . | | | | | o | c | | | c | +c | | . | | | | . | | | 22 User class id |
| 131 | . | | | | | | | | | | c | | | . | | . | | | | | | | | | 23. User class name |
| 132 | | | | | | | | | | | c | | | | . | | | | | | | . | | | 24 User class description |
| 133 | . | | . | | | | | | | | | p | | | | | | | | | | | | | 25. User class list |
| 134 | | | | | | | | | | | . | | p | | . | | | | | | | | | | 26 User |
| 135 | | | | | | | | | | | | | c | o | c | +o | | . | | | | | | | 27. User id |
| 136 | . | . | | | | | | | | | | | c | . | | . | | | | | . | | | | 28 User name |
| 137 | | | | | | | | | | | | | c | . | . | | | | | . | . | . | | | 29 User password |
| 138 | | | | | | | | | | . | | | c | . | . | . | | | | | | | | | 30 User description |
| 139 | | | | | | | | | . | | | . | | p | | . | . | . | | | | | | | 31 User list |
| 140 | . | | | | | | | | | . | | | . | | . | | . | | | | . | | | | 32. Object type |
| 141 | | | | | | | | | | | . | | | | . | c | | | . | | | | | | 33 Object id |
| 142 | | | | | | | | | | | | | | | p | . | . | | | | . | | | | 34 User class relation list |
| 143 | | | | | | | | | | | . | . | | | | p | | . | . | | | | | | 35 Manager-object relation list |
| 144 | | | | | | | | | | | | | . | . | | c | | | | | | | | | 36 Access rights set (Read, Write, Delete, etc ) |
| 145 | | | | | . | | | | | | | | | | . | | | | | | | | | | 37 Anchor |
| 146 | | | | | | | | | | | | | | . | | . | . | | | | | . | | | 38 Attribute |
| 147 | | | | | | | | | | | | | | | . | | . | | | | . | | | | 39 Departure point |
| 148 | | | | | | | | | | | | | | | . | | | | | | | | | | 40 Destination point |
| 149 | | | | | | | | | | | | | | | | | | | | | | | | | 41 Map |
| 150 | | | | | | | | | . | | | . | | | . | . | | | | | | | | | 42 Navigation path |
| 151 | | | | | | | | | | | | | . | | | | | | | | | | | | 43 Overview diagram |
| 152 | | | | | | | | | | . | . | | | | | | . | | | | | | | | 44 Sequence |
| 153 | | | | | . | | | | | | | | | | . | | | . | | | | | | | 45 Set |
| 154 | | | | | | | | | | | | | | | . | . | . | | | O | | | 11 | | { PRE-GUARD } |
| 155 | | | | | | | | | | | . | | | | | | | | | t | . | | | | 1. Atomicity is guaranteed for any function occuring at the node level |
| 156 | | | | | | | | | | | | | | | . | | | | | t | | | | | 2 The tools to edit multimedia must be available under HyperDoc's environment. |
| 157 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 158 | | | | | | | | | | | | | | | | | | | | t | | | | | 3 Regions within a node are highlighted in some way when the node is displayed on the screen |
| 159 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 160 | | | | | | | | | | | | | | | | | | | | t | | | | | 4 A window environment is available It takes care of basic window |

| | AD | AE | AF | AG | AH | AI | AJ | AK | AL | AM | AN | AO | AP | AQ | AR | AS | AT | AU | AV | AW | AX | AY | BS | BT | SU |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 161 | | | | | | | | | | | | | | | | | | | | | | | | | management functions |
| 162 | | | | | | | | | | | | | | | | | | | | t | | | | | 5 Hardware support possesses enough characteristics to meet the |
| 163 | | | | | | | | | | | | | | | | | | | | | | | | | performance requirements |
| 164 | | | | | | | | | | | | | | | | | | | | t | | | | | 6 It is the user responsibility to decide what to put in the node and how |
| 165 | | | | | | | | | | | | | | | | | | | | | | | | | to link them |
| 166 | | | | | | | | | | | | | | | | | | | | t | | | | | 7 Any modifications of administrative information (add/create nodes |
| 167 | | | | | | | | | | | | | | | | | | | | | | | | | links, organizations, networks) in HyperDoc is saved implicitly and |
| 168 | | | | | | | | | | | | | | | | | | | | | | | | | permanently when modified |
| 169 | | | | | | | | | | | | | | | | | | | | t | | | | | 8 Multiple user access to information is provided by the operating system |
| 170 | | | | | | | | | | | | | | | | | | | | t | | | | | 9 Rules in Annex 7 govern the security system |
| 171 | | | | | | | | | | | | | | | | | | | | t | | | | | 10 The tools for editing must be modified appropriately to interface |
| 172 | | | | | | | | | | | | | | | | | | | | | | | | | with HyperDoc |
| 173 | | | | | | | | | | | | | | | | | | | | t | | | | | 11 The operating system must handle protection at the file level |
| 174 | | | | | | | | | | | | | | | | | | | | | | | | | (secondary storage) |
| 194 | | | | | | | | | | | | | | | | | | | M | M | 11 | | | | {TASK} |
| 195 | | | | | | | | | | | | | | | | | | | v | | | | | | 1 Manipulation of node's content. |
| 196 | | | | | | | | | | | | | | | | | | | v | | | | | | 2 Manipulation of HyperDoc's objects. |
| 197 | | | | | | | | | | | | | | | | | | | v | | | | | | 3 Handling of attributes for HyperDoc's objects |
| 198 | | | | | | | | | | | | | | | | | | | v | | | | | | 4 Definition of security management functions |
| 199 | | | | | | | | | | | | | | | | | | | v | | | | | | 5 Navigation through a hyperdocument. |
| 200 | | | | | | | | | | | | | | | | | | | v | | | | | | 6 Searches for strings through a hyperdocument |
| 201 | | | | | | | | | | | | | | | | | | | | | v | | | | 7 Garbage collection facility to collect objects that are not used in the system |
| 202 | | | | | | | | | | | | | | | | | | | | | v | | | | 8 Versioning at the network level |
| 203 | | | | | | | | | | | | | | | | | | | | | v | | | | 9 Importation of information created outside HyperDoc |
| 204 | | | | | | | | | | | | | | | | | | | | | v | | | | 10 Facilities for printing |
| 205 | | | | | | | | | | | | | | | | | | | | | v | | | | 11 Pattern search capabilities |
| 241 | | | | | | | | | | | | | | | | | | | | G | | | 2.7 | | {POST-GUARD} |
| 242 | | | | | | | | | | | | | | | | | | | | t | | | | | 1 To provide a support environment for the software life cycle |
| 243 | | | | | | | | | | | | | | | | | | | | t | | | | | 2 To provide capabilities to allow for the documentation and the |
| 244 | | | | | | | | | | | | | | | | | | | | | | | | | management of the software development process |
| 245 | | | | | | | | | | | | | | | | | | | | t | | | | | 3 To make the users feel they can move freely through the information |
| 246 | | | | | | | | | | | | | | | | | | | | | | | | | according to their own needs |
| 247 | | | | | | | | | | | | | | | | | | | | t | | | | | 4 To provide manipulation facilities for the document's organization and |
| 248 | | | | | | | | | | | | | | | | | | | | | | | | | the document's content. |
| 249 | | | | | | | | | | | | | | | | | | | | t | | | | | 5 To run under Unix environments |
| 250 | | | | | | | | | | | | | | | | | | | | t | | | | | 6 Only one user can modify the content of a node at a time |
| 251 | | | | | | | | | | | | | | | | | | | | | | | | | 7 To create a new user class and add it to the user class list. |
| 252 | | | | | | | | | | | | | | | | | | | | | | | | | 8 To remove the definition of a user class from the user class list |
| 253 | | | | | | | | | | | | | | | | | | | | | | | | | 9 To modify the name and/or description of a user class |
| 254 | | | | | | | | | | | | | | | | | | | | | | | | | 10 To create a new user and add him/her to the user list. |
| 255 | | | | | | | | | | | | | | | | | | | | | | | | | 11 To remove a user from the user list. |
| 256 | | | | | | | | | | | | | | | | | | | | | | | | | 12 To modify the name password and/or description of a user |
| 257 | | | | | | | | | | | | | | | | | | | | | | | | | 13 To make a user a member of a user class |
| 258 | | | | | | | | | | | | | | | | | | | | | | | | | 14 To revoke a user's membership from a user class |
| 259 | | | | | | | | | | | | | | | | | | | | | | | | | 15 To make a user eligible to access an object with certain access privileges |
| 260 | | | | | | | | | | | | | | | | | | | | | | | | | 16 To make a user class eligible to access an object with certain access |
| 261 | | | | | | | | | | | | | | | | | | | | | | | | | privileges |
| 262 | | | | | | | | | | | | | | | | | | | | | | | | | 17 To revoke a user access to an object which he/she used to access with |
| 263 | | | | | | | | | | | | | | | | | | | | | | | | | certain access privileges |
| 264 | | | | | | | | | | | | | | | | | | | | | | | | | 18 To revoke a user class access to an object which they used to access with |
| 265 | | | | | | | | | | | | | | | | | | | | | | | | | certain access privileges |
| 266 | | | | | | | | | | | | | | | | | | | | | | | | | 19 To update a user access to an object which he/she has access to already |
| 267 | | | | | | | | | | | | | | | | | | | | | | | | | with certain access privileges |
| 268 | | | | | | | | | | | | | | | | | | | | | | | | | 20 To update a user class access to an object which they have access to |
| 269 | | | | | | | | | | | | | | | | | | | | | | | | | already with certain access privileges |
| 270 | | | | | | | | | | | | | | | | | | | | | | | | | 21 To show the access rights that a user has on an object |
| 271 | | | | | | | | | | | | | | | | | | | | t | | | | | 22 The response time to any user command must be no longer than 30 seconds |
| 272 | | | | | | | | | | | | | | | | | | | | t | | | | | 23 The overview diagram must be updated after a reasonable number of |
| 273 | | | | | | | | | | | | | | | | | | | | | | | | | modifications have been made to the structure of the hyperdocument |
| 274 | | | | | | | | | | | | | | | | | | | | t | | | | | 24 Operations performed by the SUPER user must take effect instantaneously |
| 275 | | | | | | | | | | | | | | | | | | | | t | | | | | 25 Performance of HyperDoc must not degrade below the stated timing |
| 276 | | | | | | | | | | | | | | | | | | | | | | | | | constraints when the system load increases |
| 277 | | | | | | | | | | | | | | | | | | | | t | | | | | 26 Availability available at all times to at most 50 users |
| 278 | | | | | | | | | | | | | | | | | | | | t | | | | | 27 Security at three levels host-user level, HyperDoc-user level, and |
| 279 | | | | | | | | | | | | | | | | | | | | | | | | | and HyperDoc-object level See Annex 7 for security rules |
| 280 | | | | | | | | | | | | | | | | | | | M | M | 5 | | | | {REMARK} |
| 281 | | | | | | | | | | | | | | | | | | | | | | | | | 1 The user class description attribute captures information pertaining to |
| 282 | | | | | | | | | | | | | | | | | | | | | | | | | the role that the user class plays within the context of the software |
| 283 | | | | | | | | | | | | | | | | | | | | | | | | | engineering process User classes should be unique (i.e. unique user |
| 284 | | | | | | | | | | | | | | | | | | | | | | | | | class id) |
| 285 | | | | | | | | | | | | | | | | | | | | | | | | | 2 A user class id cannot be modified, only created or deleted |
| 286 | | | | | | | | | | | | | | | | | | | | | | | | | 3 The notation <user id, object id, access rights set> associates a user |
| 287 | | | | | | | | | | | | | | | | | | | | | | | | | with an object in the way specified by the access rights set. |
| 288 | | | | | | | | | | | | | | | | | | | v | | | | | | 4 Current requirements |
| 289 | | | | | | | | | | | | | | | | | | | | | | v | | | 5 Future requirements |

| | AZ | BA | BB | BC | BD | BE | BF | BG | BH | BI | BJ | BK | BL | BM | BN | BO | BP | BQ | BR | BS | BT | BU |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | 1 | | { IDENTIFICATION } |
| 2 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 1 intellMap for the HyperDoc SRD |
| 3 | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | 6 | | { REFERENCE } |
| 4 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 1 IEEE Std 830-1984 |
| 5 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 2 SRD for a hyperdocument system |
| 6 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 3 V S Alagar, P Goyal, "CONSLT A Software Life Cycle Tool", |
| 7 | | | | | | | | | | | | | | | | | | | | | | Concordia University, Montreal, Canada |
| 8 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 4 Pankaj K. Garg and Walt Scacchi "S.P.IVS Designing an |
| 9 | | | | | | | | | | | | | | | | | | | | | | Intelligent Software Hypertext System", University of |
| 10 | | | | | | | | | | | | | | | | | | | | | | Southern California IEEE Expert, Fall 1989 |
| 11 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 5 Danny B. Lange, "A formal Approach to Hypertext using |
| 12 | | | | | | | | | | | | | | | | | | | | | | Post Prototype Formal Specification", Department of |
| 13 | | | | | | | | | | | | | | | | | | | | | | Computer Science, Technical University of Denmark. |
| 14 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 6 Jacob Nielsen, Hypertext and Hypermedia, Academic Press, 1990 |
| 15 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 7 Pankaj K. Garg and Walt Scacchi, "A Hypertext System to manage |
| 16 | | | | | | | | | | | | | | | | | | | | | | Software Life-Cycle Documents", IEEE Software May 1990 |
| 17 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | 8 Jeff Conklin, "Hypertext - An Introduction and Survey", Computer |
| 18 | | | | | | | | | | | | | | | | | | | | | | Microelectronics and Computer Technology Corp., Computer, |
| 19 | | | | | | | | | | | | | | | | | | | | | | September 1987 |
| 20 | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | A | 6 | | { VIEW } |
| 21 | | | | | | | | | | | | | | | | | | | | | | 1 Definitions |
| 22 | | | | | | | | | | | | | | | | | | | | | | 2 Product Perspective |
| 23 | | | | | | | | | | | | | | | | | | | | | | 3 Product Functions |
| 24 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | | | | | 4. Functional Requirements |
| 25 | | | | | | | | | | | | | . | | | | v | v | v | | | 5 Product Users |
| 26 | | | | | | | | | | | | | | | | | | | | | v | 6 Set Cardinality |
| 86 | | | | | | | | | | | | | | | | | O | O | O | 6 | | { USER TYPE } |
| 87 | | | | | | | | | | | | | | | | | o | | | | | 1 Super user |
| 88 | | | | | | | | | | | | | | | | | | o | | | | 2. Builder |
| 89 | | | | | | | | | | | | | | . | | | | | o | | | 3 Consulter |
| 90 | | | | | | | | | | | | | | | | | | | | | | 4 User leader |
| 91 | | | | | | | | | | | | | | | | | | | | | | 5 Class leader |
| 92 | | | | | | | | | | | | | | | | | | | | | | 6 Author |
| 93 | H | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | | | | 12 | | { OPERATION } |
| 94 | c | o | | | | | | | | | | | | . | . | | | | | | | 1 Create_User_Class |
| 95 | c | | o | | | | | | | | | | | . | . | . | | | | | | 2 Delete_User_Class |
| 96 | c | | | o | | . | | | | | | | . | | | | | | | | | 3. Update_User_Class |
| 97 | c | | . | | o | | | | | | | | | | | | | | | | | 4 Create_User |
| 98 | c | | | | | o | . | | | | | | | . | . | | | | | | | 5 Delete_User |
| 99 | c | | | | | | o | | | | | | | | . | . | | | | | | 6 Update_User |
| 100 | c | | | | | | | o | | | | | | | | | | | | | | 7 Assign_User_To_Class |
| 101 | c | | | | | | | | o | | | | | | | | | | | | | 8. Delete_User_From_Class |
| 102 | c | | | | | | | | . | o | o | | | | | | | | | | | 9 Assign_Object_And_Rights_To_Manager |
| 103 | c | | | | | | | | | | | o | o | | . | | | | | | | 10 Delete_Object_And_Rights_From_Manager |
| 104 | c | | | | | | | | | | | | | o | o | | | | | | | 11 Update_Object_Rights_Of_Manager |
| 105 | c | | | | | | | | | | | | | . | | o | . | | | | | 12. Show_Object_Access_Rights |
| 106 | | F | F | F | F | F | F | F | F | F | F | F | F | F | F | F | M | M | M | 45 | | { INTERNAL OBJECT } |
| 109 | | | | | | | | | | | | | | | | | v | v | v | | | 1 HyperDoc |
| 110 | | | | | | | | | | | | | | | | | | | | | | 2 Hyperdocument |
| 111 | | | | | | | | | | | | | | | | | | | | | | 3 Network |
| 112 | | | | | | | | | | | | | | | | | | | | | | 4 Network identity |
| 113 | | | | | . | | | | | | | | . | | | | | | | | | 5 Network attributes |
| 114 | | | | | | | . | | . | | | | | | | | | | | | | 6 Organization |
| 115 | | | | | | | | | | | | | | | . | | | | | | | 7 Link |
| 116 | | | | | | | | | | | | | | | | | | | | | | 8. Node |
| 117 | | | | | | | | | | | | | | | | | | | | | | 9 Organization attributes |
| 118 | | | | | | | | | | | | | | | | | | | | | | 10 Link attributes |
| 119 | | | | | | | | | | | | | | | | | | | | | | 11 Link name |
| 120 | | | | . | | | | | | | | | | | | | | | | | | 12. Region |
| 121 | | | | | | | | | | | | | | | . | | | | | | | 13 Node attributes |
| 122 | | | . | | | | | | | | | | . | | | | | . | | | | 14 Region attributes |
| 123 | | | | | | . | | | | | | | | | | | . | . | | | | 15 Lock attribute |
| 124 | | | | | | | | | | | | | | | | | | | | | | 16 Node window |
| 125 | | | | | . | | | | . | | | | . | | | | | | | | | 17 Node window header |
| 126 | | | | | | | | | | | | | | . | | | | | | | | 18 Cursor position |
| 127 | | | | | | | | | | | | | | | | | | | | | | 19 Information window |
| 128 | | | | | | | | | | | | | | | | | | | | | | 20 Information window header |
| 129 | | | | o | | | | | | | . | | | | | | | | | | | 21 User class |
| 130 | | i | i | i | | xi | | i | i | | | i | | i | | i | | | | | | 22 User class id |
| 131 | | i | | ·i | | | | | | | | | | | | | . | | | | | 23 User class name |
| 132 | | i | | ·i | | | | | | | | | | | | | | | | | | 24 User class description |
| 133 | | b | b | i | | | i | i | i | | i | | | i | | i | | | | | | 25 User class list |
| 134 | | | | | | | | | | . | | | | | | | | . | | | | 26 User |
| 135 | | | | | i | zi | i | i | i | i | | i | | | i | | i | | | | | 27 User id |
| 136 | | | | | i | | ·b | | | | | | | | | | | | | | | 28 User name |
| 137 | | | | | i | | ·b | . | | | | | | | . | | . | | | | | 29 User password |
| 138 | | | | | i | | ·b | | | | | | | | | | | | | | | 30 User description |
| 139 | | | | | b | b | | i | i | i | | i | | | i | | | | | | | 31 User list |
| 140 | | | | | | xi | | | | . | | i | i | i | i | | | | | | | 32 Object type |
| 141 | | | | | | xi | | | | | | i | i | i | i | i | | | | | | 33 Object id |
| 142 | | | | | | | | b | b | | | | | | | | | | | | | 34 User-class relation list |

| | AZ | BA | BB | BC | BD | BE | BF | BG | BH | BI | BJ | BK | BL | BM | BN | BO | BP | BQ | BR | BS | BT | | BU |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 143 | | | | | | | | | | b | b | b | b | b | b | | | | | | | 35 | Manager object relation list |
| 144 | | | | | | | | | | i | i | | | i | i | e | | | | | | 36 | Access rights set (Read, Write, Delete, etc ) |
| 145 | | . | | | | | | | | | | | | | | | | | | | | 37 | Anchor |
| 146 | | | | | | | | | | | | | | | | | | | | | | 38 | Attribute |
| 147 | | | | | | | | | | | | | | | | | | | | | | 39 | Departure point |
| 148 | | | | | | | | | | | | | | | | | | | | | | 40 | Destination point |
| 149 | | | | | | | | | | | | | | | | | | | | | | 41 | Map |
| 150 | | | | | | | | | | | | | | | | | | | | | | 42 | Navigation path |
| 151 | | | | | | | | | | | | | | | | | | | | | | 43 | Overview diagram |
| 152 | | | | | | | | | | | | | | | | | | | | | | 44 | Sequence |
| 153 | | | | | | | | | | | | | | | | | | | | | | 45 | Set |
| 175 | | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | | | | | | 15 | {PRE-CONDITION} |
| 176 | | t | | | | | | | | | | | | | | | | | | | | 1 | A user class list exists |
| 177 | | t | t | t | t | t | +t | t | t | t | t | +t | t | +t | t | | | | | | | 2 | The user performing the operation is the SUPER user |
| 178 | | f | t | t | | | | t | t | | t | | t | | t | | | | | | | 3 | The user class id is in the user class list |
| 179 | | | t | | | | | | | | | | | | | | | | | | | 4 | The set of user ids belonging to that class is empty |
| 180 | | | | t | | | f | | | | | | | | | | | | | | | 5 | The new values are different than the old ones |
| 181 | | | | | t | | | | | | | | | | | | | | | | | 6 | A user list exists |
| 182 | | | | | f | t | | t | t | t | | t | | t | | | | | | | | 7 | The user id is in the user list |
| 183 | | | | | | | +t | | | | | | | | | | | | | | | 8 | The user performing the operation is the User himself |
| 184 | | | | | | | | t | t | | | | | | | | | | | | | 9 | The <user id, user class id> relation already exists in the user class relation list |
| 185 | | | | | | | | | | | | | | | | | | | | | | | |
| 186 | | | | | | | | | t | | | | | | | | | | | | | 10 | This is the only class that the user belongs to |
| 187 | | | | | | | | | | t | | t | | t | | | | | | | | 11 | The <user id object id> relation exists in the manager-object relation list |
| 188 | | | | | | | | | | | | | | | | | | | | | | | |
| 189 | | | | | | | | | | | f | | t | | t | | | | | | | 12 | The <user class id, object id> relation exists in the manager-object relation list |
| 190 | | | | | | | | | | | | | | | | | | | | | | | |
| 191 | | | | | | | | | | | | +t | | +t | | | | | | | | 13 | The user performing the operation is the class leader |
| 192 | | | | | | | | | . | | | | | f | f | | | | | | | 14 | The old and new access rights sets are the same |
| 193 | | | | | | | t | | | | | | | | | | | | | | | 15 | The user id is in the user class list |
| 194 | H | | | | | | | | | | | | | | | | | | | | | 11 | {TASK} |
| 195 | | | | | | | | | | | | | | | | | | | | | | 1 | Manipulation of node's content |
| 196 | | | | | | | | | | | | | | | | | | | | | | 2 | Manipulation of HyperDoc's objects |
| 197 | | | | | | | | | | | | | | | | | | | | | | 3 | Handling of attributes for HyperDoc's objects |
| 198 | P | | | | | | | | | | | | | | | | | | | | | 4 | Definition of security management functions |
| 199 | | | | | | | . | | | | . | | | | | | | | | | | 5 | Navigation through a hyperdocument |
| 200 | | | | . | | | | | | | | | | | | | | | | | | 6 | Searches for strings through a hyperdocument |
| 201 | . | | | | | | | | | | | | | | . | | | | | | | 7 | Garbage collection facility to collect objects that are not used in the system |
| 202 | | | | | | | | | | | | | | | | | | | | | | 8 | Versioning at the network level |
| 203 | | | | | | | | | | | | | | | | . | | | | | | 9 | Importation of information created outside HyperDoc |
| 204 | | | | | | | | | | | | | | | | | | | | | | 10 | Facilities for printing |
| 205 | . | | | | | | | | | | | | | | | | | | | | | 11 | Pattern search capabilities |
| 206 | | S | S | S | S | S | S | S | S | S | S | S | S | S | S | S | | | | | | 19 | {ACTION} |
| 207 | | 1 | | | | | | | | | | | | | | | | | | | | 1 | Create the new user class and add it to the user class list |
| 208 | | 2 | 2 | 2 | 2 | 4 | | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | | | | | | | 2 | Inform the SUPER user of the results of the operation |
| 209 | | | 1 | | . | | | | | | | | | | | | | | | | | 3 | Delete the user class from the user class list |
| 210 | | | | 1 | | | | | | | | | | | | | | | | | | 4 | Replace the old name and/or description of user class with new values |
| 211 | | | | | 1 | | | | | | | | | | | | | | | | | 5 | Create the new user and add him/her to the user list |
| 212 | | | | | | 1 | | | | | | | | | | | | | | | | 6 | Allocate all objects rights of deleted user to another user |
| 213 | | | | | | 2 | | | | | | | | | | | | | | | | 7 | Delete user from all classes that appear in the set of user classes |
| 214 | | | . | | . | 3 | . | | | | | | | | | | | | | | | 8 | Delete the user from the user list |
| 215 | | | | | | | 2 | | | | | | | | | | | | | | | 9 | Inform the user of the results of the operation |
| 216 | | | | . | | | 1 | | | | | | | | | | | | | | | 10 | Replace the old name and/or password and/or description of user with new values |
| 217 | | | | | | | | | | | | | | | | | | | | | | | |
| 218 | | | | | | | | 1 | | | | | | | | | | | | | | 11 | Add the <user id, user class id> relation in the user-class relation list |
| 219 | | | | | | | | | 1 | | | | | | | | | | | | | 12 | Remove the <user id user class id> relation from the user-class relation list |
| 220 | | | | | | | | | | | | | | | | | | | | | | | |
| 221 | | | | | | | | | | 1 | | | | | | | | | | | | 13 | Add the <user id, object id, access rights set> relation to the manager-object relation list |
| 222 | | | | | | | | | | | | | | | | | | | | | | | |
| 223 | | . | | | | | | | | | 1 | | | | | | | | | | | 14 | Add the <user class id, object id, access rights set> relation to the manager-object relation list |
| 224 | | | | | | | | | | | | | | | | | | | | | | | |
| 225 | | | | | | | | . | | | . | 2 | | 2 | | | | | | | | 15 | Inform the class leader of the results of the operation |
| 226 | | | | . | | | . | | | | | 1 | | | | | | | | | . | 16 | Delete the <user id object id, access rights set> relation from the manager-object relation list |
| 227 | | | | | | | | | | | | | | | | | | | | | | | |
| 228 | | | | | | | | | | | | | 1 | | | | | | | | | 17 | Delete the <user class id, object id, access rights set> relation from the manager-object relation list |
| 229 | | | | | | | | | | | | | | | | | | | | | | | |
| 230 | | | | | | | | | | | | | | | t | t | | | | | | 18 | Replace the old access rights set with the new ones |
| 231 | | | | | | | | | | | | | | | | | 1 | | | | | 19 | Display allocated rights for the user |
| 232 | | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | | | | | | 8 | {POST-CONDITION} |
| 233 | | t | t | | | | | | | | | | | | | | | | | | | 1 | New user class list |
| 234 | | | | t | | | | | | . | | | | | | | | | | | | 2 | New modified user class |
| 235 | | | | | t | t | | | | | . | | | | | | | | | | | 3 | New user list |
| 236 | | | | | | t | | | | . | | | | | | | | | | | | 4 | All objects rights of deleted user transfered to another user |
| 237 | | | | | | | t | | | | | | | | | | | | | | . | 5 | New modified user attributes |
| 238 | | | | | | | | t | t | | | | | | . | | | | | | | 6 | New user-class relation list |
| 239 | . | | | | | | | | | t | t | t | t | t | t | t | | | | | | 7 | New manager object relation list |
| 240 | . | | | | | | | | | | | | | | | | t | | | | | 8 | The object rights for the user |
| 241 | | G | G | G | G | G | G | G | G | G | G | G | G | G | G | G | | | | | | 27 | {POST-GUARD} |
| 242 | | | | | | | | | . | | | | | | | | | | | | | 1 | To provide a support environment for the software life cycle |
| 243 | | | | | | | | | | | | | | | | | | | | | | 2 | To provide capabilities to allow for the documentation and the management of the software development process |
| 244 | | | | | | | | | | | | | | | | | | | | | | | |

| | AZ | BA | BB | BC | BD | BE | BF | BG | BH | BI | BJ | BK | BL | BM | BN | BO | BP | BQ | BR | BS | BT | BU |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 246 | | | | | | | | | | | | | | | | | | | | | | 3 To make the users feel they can move freely through the information |
| 246 | | | | | | | | | | | | | | | | | | | | | | according to their own needs |
| 247 | | | | | | | | | | | | | | | | | | | | | | 4 To provide manipulation facilities for the document's organization and |
| 248 | | | | | | | | | | | | | | | | | | | | | | the document's content |
| 249 | | | | | | | | | | | | | | | | | | | | | | 5 To run under Unix environments |
| 250 | | | | | | | | | | | | | | | | | | | | | | 6 Only one user can modify the content of a node at a time |
| 251 | | | | | | | | | | | | | | | | | | | | | | 7 To create a new user class and add it to the user class list |
| 252 | | | | | | | | | | | | | | | | | | | | | | 8 To remove the definition of a user class from the user class list |
| 253 | | | | | | | | | | | | | | | | | | | | | | 9 To modify the name and/or description of a user class |
| 254 | | | | | | | | | | | | | | | | | | | | | | 10 To create a new user and add him/her to the user list |
| 255 | | | | | | | | | | | | | | | | | | | | | | 11 To remove a user from the user list |
| 256 | | | | | | | | | | | | | | | | | | | | | | 12 To modify the name, password and/or description of a user |
| 257 | | | | | | | | | | | | | | | | | | | | | | 13 To make a user a member of a user class |
| 258 | | | | | | | | | | | | | | | | | | | | | | 14. To revoke a user's membership from a user class |
| 259 | | | | | | | | | | | | | | | | | | | | | | 15 To make a user eligible to access an object with certain access privileges |
| 260 | | | | | | | | | | | | | | | | | | | | | | 16 To make a user class eligible to access an object with certain access |
| 261 | | | | | | | | | | | | | | | | | | | | | | privileges |
| 262 | | | | | | | | | | | | | | | | | | | | | | 17 To revoke a user access to an object which he/she used to access with |
| 263 | | | | | | | | | | | | | | | | | | | | | | certain access privileges |
| 264 | | | | | | | | | | | | | | | | | | | | | | 18 To revoke a user class access to an object which they used to access with |
| 265 | | | | | | | | | | | | | | | | | | | | | | certain access privileges |
| 266 | | | | | | | | | | | | | | | | | | | | | | 19 To update a user access to an object which he/she has access to already |
| 267 | | | | | | | | | | | | | | | | | | | | | | with certain access privileges |
| 268 | | | | | | | | | | | | | | | | | | | | | | 20 To update a user class access to an object which they have access to |
| 269 | | | | | | | | | | | | | | | | | | | | | | already with certain access privileges |
| 270 | | | | | | | | | | | | | | | | | | | | | | 21 To show the access rights that a user has on an object |
| 271 | | | | | | | | | | | | | | | | | | | | | | 22 The response time to any user command must be no longer than 30 seconds |
| 272 | | | | | | | | | | | | | | | | | | | | | | 23 The overview diagram must be updated after a reasonable number of |
| 273 | | | | | | | | | | | | | | | | | | | | | | modifications have been made to the structure of the hyperdocument |
| 274 | | | | | | | | | | | | | | | | | | | | | | 24 Operations performed by the SUPER user must take effect instantaneously |
| 275 | | | | | | | | | | | | | | | | | | | | | | 25 Performance of HyperDoc must not degrade below the stated timing |
| 276 | | | | | | | | | | | | | | | | | | | | | | constraints when the system load increases |
| 277 | | | | | | | | | | | | | | | | | | | | | | 26 Availability - available at all times to at most 50 users. |
| 278 | | | | | | | | | | | | | | | | | | | | | | 27 Security at three levels host-user level, HyperDoc-user level, and |
| 279 | | | | | | | | | | | | | | | | | | | | | | and HyperDoc-object level. See Annex 7 for security rules |
| 280 | M | M | M | M | M | M | M | M | M | M | M | M | M | M | M | | | . | | | 5 | (REMARK) |
| 281 | v | | | v | | | | | . | | | | | | | | | . | . | | | 1 The user class description attribute captures information pertaining to |
| 282 | | | | | | | | | | | | | | | | | | | | | | the role that the user class plays within the context of the software |
| 283 | | | | | | | | | | | | | | | | | | | | | | engineering process. User classes should be unique (i.e. unique user |
| 284 | | | | | | | | | | | | | | | | | | | | | | class id) |
| 285 | | | | v | | | v | | | | | | | | | | | | | | | 2 A user class id cannot be modified, only created or deleted |
| 286 | | | | | | | | | | v | v | | | | | | | | | | | 3 The notation <user id, object id, access rights set> associates a user |
| 287 | | | | | | | | | | | | | | | | | | | | | | with an object in the way specified by the access rights set. |
| 288 | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | v | | . | | | | 4. Current requirements |
| 289 | | | | | | | | | | | | | | | | | | | | | | 5 Future requirements |

# Appendix E

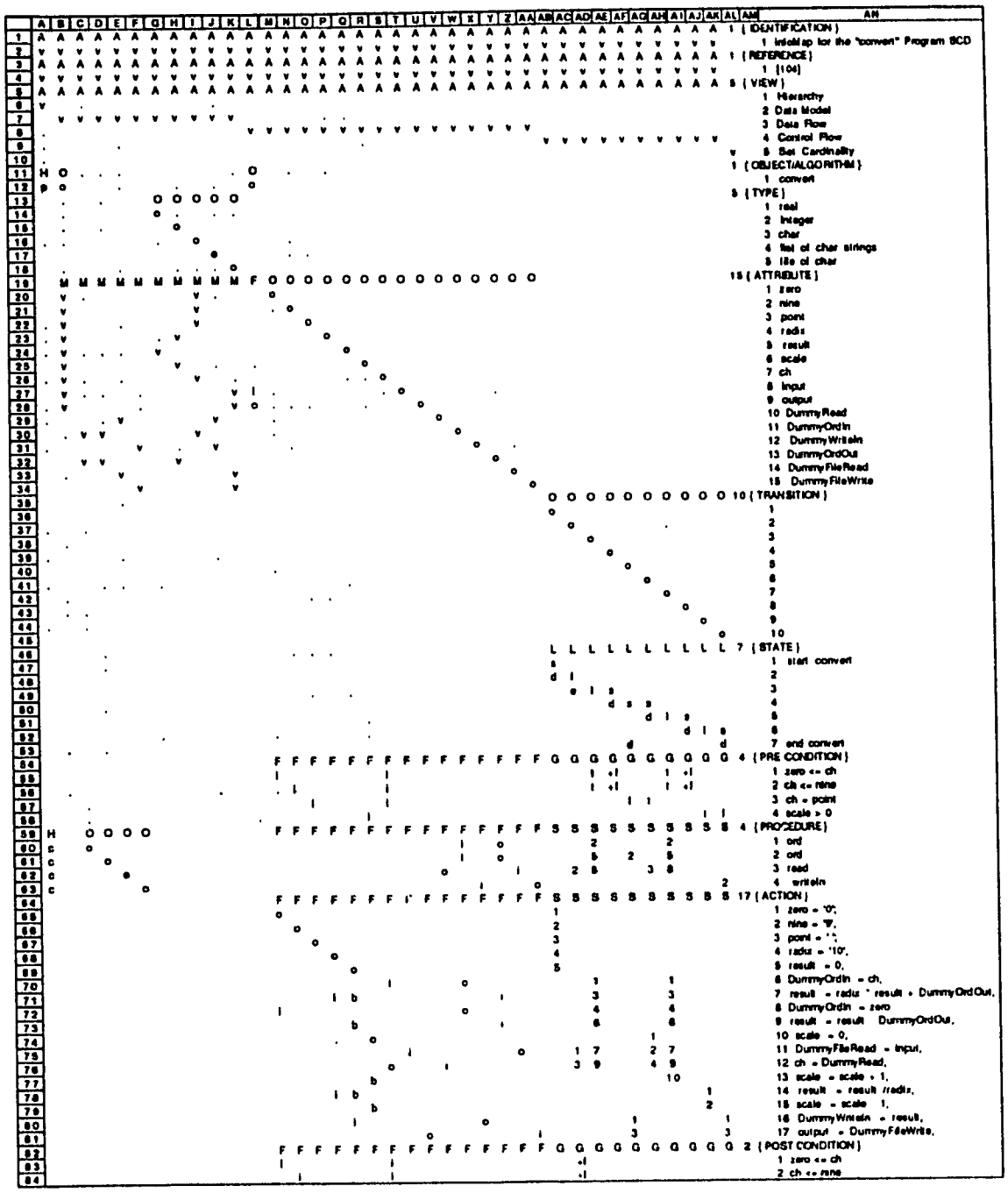
## The convert Program SCD [51]

```
PROGRAM convert (input, output)
   CONST
      zero = '0';
      nine = '9';
      point = '.';
      radix = 10;
   VAR
      result : real;
      scale : integer;
      ch : char;
   BEGIN
      result := 0;
      REPEAT
         read(ch)
      UNTIL (zero <= ch) AND (ch <= nine);
      WHILE (zero <= ch) AND (ch <= nine) DO
      BEGIN
         result := radix * result + ord(ch) - ord(zero);
         read(ch)
      END; { while }
      IF ch = point
      THEN
      BEGIN
         scale := 0;
         read(ch);
         WHILE (zero <= ch) AND (ch <= nine) DO
         BEGIN
            result := radix * result + ord(ch) - ord(zero);
            read(ch);
            scale := scale + 1
         END; { while }
         WHILE scale > 0 DO
         BEGIN
            result := result / radix;
            scale := scale - 1
         END { while }
      END;
      writeln(result)
   END. { convert }
```

# Appendix F

## The infoMap for the convert Program SCD

| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | A | A | 1 | { IDENTIFICATION } | |
| 2 | v | v | v | v | v | v | v | | | 1. infoSchema for the "convert" Program SCD. |
| 3 | A | A | A | A | A | A | A | 1 | { REFERENCE } | |
| 4 | A | A | A | A | A | A | A | 5 | { VIEW } | |
| 5 | v | . | . | . | . | . | . | | | 1. Hierarchy |
| 6 | . | v | v | v | . | . | . | | | 2. Data Model |
| 7 | . | . | . | . | v | v | . | | | 3. Data Flow |
| 8 | . | . | . | . | . | . | v | | | 4. Control Flow |
| 9 | . | . | . | . | . | . | . | v | | 5. Set Cardinality |
| 10 | H | O | . | . | O | . | . | 1 | { OBJECT/ALGORITHM } | |
| 11 | . | . | . | O | . | . | . | 5 | { TYPE } | |
| 12 | . | M | M | M | F | O | . | 15 | { ATTRIBUTE | |
| 13 | . | . | . | . | . | . | O | 10 | { TRANSITIO | } |
| 14 | . | . | . | . | . | . | L | 7 | { STATE } | |
| 15 | . | . | . | . | . | F | G | 4 | { PRE-COND ION } | |
| 16 | H | . | O | . | . | F | S | 4 | { PROCEDUR | } |
| 17 | . | . | . | . | . | F | S | 17 | { ACTION } | |
| 18 | . | . | . | . | . | F | G | 2 | { POST-CONL ITION } | |


AN
1 { IDENTIFICATION }
1 Intellap for the "convert" Program BCD
1 { REFERENCE }
1 [104]
5 { VIEW }
1 Hierarchy
2 Data Model
3 Data Flow
4 Control Flow
5 Set Cardinality
1 { OBJECT/ALGORITHM }
1 convert
5 { TYPE }
1 real
2 Integer
3 char
4 list of char strings
5 file of char
15 { ATTRIBUTE }
1 zero
2 nine
3 point
4 radix
5 result
6 scale
7 ch
8 input
9 output
10 DummyRead
11 DummyOrdIn
12 DummyWriteln
13 DummyOrdOut
14 DummyFileRead
15 DummyFileWrite
10 { TRANSITION }
1
2
3
4
5
6
7
8
9
10
7 { STATE }
1 start convert
2
3
4
5
6
7 end convert
4 { PRE CONDITION }
1 zero <= ch
2 ch <= nine
3 ch = point
4 scale > 0
4 { PROCEDURE }
1 ord
2 ord
3 read
4 writeln
17 { ACTION }
1 zero = '0',
2 nine = '9',
3 point = '.',
4 radix = '10',
5 result = 0,
6 DummyOrdIn = ch,
7 result = radix * result + DummyOrdOut,
8 DummyOrdIn = zero
9 result = result DummyOrdOut,
10 scale = 0,
11 DummyFileRead = input,
12 ch = DummyRead,
13 scale = scale + 1,
14 result = result /radix,
15 scale = scale 1,
16 DummyWriteln = result,
17 output = DummyFileWrite,
2 { POST CONDITION }
1 zero <= ch
2 ch <= nine

# Appendix G

# Software Requirement Document
# for
# New Editor (NED)

Prepared by
Group 1

Version 2.0

submitted in partial fulfilment
of the requirements of
COMP 354
Concordia University
October 1990

TABLE OF CONTENTS


1 INTRODUCTION ..... G-4

1.1 Purpose ..... G-4

1.2 Scope ..... G-5

1.3 Definitions ..... G-6

1.4 Notations ..... G-8

1.5 References ..... G-8

1.6 Overview ..... G-9

2 GENERAL DESCRIPTION ..... G-10

2.1 Product Perspective ..... G-10

2.1.1 What is Ned ..... G-10
2.1.2 Buffers, files and documents ..... G-10
2.1.3 Modes of Operations ..... G-10
2.1.3.1 Command Mode ..... G-10
2.1.3.2 Input Mode ..... G-10
2.1.4 What is expected from the User ..... G-10
2.1.5 Contextual Aspect ..... G-10
2.1.6 System Model ..... G-11
2.1.6.1 Model ..... G-11
2.1.6.2 Description of the Model ..... G-12
2.1.7 Future Requirements ..... G-12

2.2 Product Functions ..... G-13

2.2.1 Main Tasks ..... G-13
2.2.1.1 Load a file ..... G-13
2.2.1.2 Create a document ..... G-13
2.2.1.3 Insert line(s) ..... G-13
2.2.1.4 Append line(s) ..... G-13
2.2.1.5 Delete line(s) ..... G-13
2.2.1.6 Display line(s) ..... G-13
2.2.1.7 Save a document ..... G-13

2.3 Assumptions ..... G-14

2.4 Constraints ..... G-15

3 SPECIFIC REQUIREMENTS..... G-16

3.1 Functional Requirements..... G-16

3.1.1 Append_line..... G-16
3.1.2 Change_current-line..... G-19
3.1.3 Change_mode..... G-20
3.1.4 Create_document..... G-21
3.1.5 Delete_line..... G-22
3.1.6 Display_line..... G-23
3.1.7 Insert_line..... G-24
3.1.8 Load_document_from_file..... G-26
3.1.9 Quit..... G-27
3.1.10 Write_lines_from_buffer_to_file..... G-28

3.2 Overall Performance Requirements..... G-30

3.2.1 Accuracy..... G-30
3.2.2 Timing Constraints..... G-30

3.3 Attributes and Documents to be delivered..... G-31

3.3.1 Attributes..... G-31
3.3.2 Documents to be delivered..... G-31

3.4 Acceptance Criteria..... G-32

# 1. INTRODUCTION

Line editors appeared in the early ages of computer software. The needs for creating text and programs easily was imperative. The first line editors were designed so that each line edited corresponded to an image of a punched card, used before inputting commands, because they would be easily interpreted by the computer (but not by the users). But their limitations, especially the limited line length of 80 columns, were too restrictive. The big step toward modern editors was to be able to abstract from a card line image to a line of text without having to worry how the computer will interpret it. Since then, editors have become an invaluable tool for creating texts and programs, and their long future makes no doubts.

## 1.1. Purpose

The purpose of this SRD is primarily to state the functional and non-functional requirements of NED (see sec2.3, sec2.4, sec3.1, sec3.2), along with a description of its capabilities. In addition to this, it provides users with a reference manual for consultation. It is also intended to provide more experienced users and maintenance people with the specifics of the software so that upgrade, enhancements and maintenance are possible.

This SRD has been written by the member of group 1 of the software engineering class coordinated by Alain Sarraf and assisted by Laverne Morisson. The document will be submitted to Dr. Alagar for verification and evaluation.

## 1.2. Scope

The purpose of ned is to provide the user with an easy way to edit files, by using simple one character commands, and consistent expressions. Since ned is a line editor, the commands will work on a complete line or a range of complete lines. Ned will work in UNIX environment and networked SUN workstations.

## 1.3. Definitions

- access : a file may be accessed for reading, writing or appending.

- access time : The time required to access an item of data on a storage device.

- address : specifies a particular line position in the document.

- append : to add information to the end of a document, or line without altering the information already present.

- blank : a space or tab character.

- buffer : A storage area used by ned to hold information temporarily.

- character : represent symbols people use for written communication; any keystroke from the keyboard is represented by an ASCII code that includes all the possible symbols in our scope.

- command : an instruction given to the editor by the user to perform a task

- create : to bring into existence (file or document).

- current line : the presently selected line of text (implicitly indicated).

- current line position : refers to position/address of the current line in the document.

- cursor : An indicator on the screen which shows where the next character will be displayed.

- delete : remove, thereby freeing the space a line(s) occupies in the document.

- document: the current contents of the buffer is collectively known as the document.

- escape : a temporary exit from a command from one mode of operation to another.

- file : a mechanism for the long-term storage of information in a computer system.

- filename : a symbolic name used to reference a file (see section 1.4).

- hardware : The collective name given to the electronic and electro-mechanical devices that go together to make a computer.

- insert : to place lines(s) at a line position in the document without altering the information already present.

- interactive system : A system designed to carry out a dialogue with a user; reads commands from a terminal and then executes them immediately.

- keyboard : an input device which generates a signal representing a character corresponding to the key that has been pressed by the user.

- line : a sequence of characters terminated by a newline which is generated by the RETURN key (carriage return).

- line editor : a type of editor that allows the user to locate lines in a file so that the text in the file may be changed.

- newline : a control character used to mark the end of a line of text; usually generated by striking the RETURN key (see section 1.4).

- operator : a symbol defining the action to be performed on one or more operands to obtain a result.

- process : an executing program.

- runtime error : errors which occur when a program is executing; logical or sematic errors.

- screen : video display unit

- sematics : the meaning of symbols and groups of symbols used in a programming language.

- string : a sequence of characters usually terminated by a delimiter.

- syntax : the set of rules defining the grammar of a programming language.

- text : one or more lines composed of printable characters.

- word : a sequence of characters in a line of text delimited in some way.

- write : the output of data from a buffer to an external peripheral device.

- quit : exiting from a process.

## 1.4. Notations

[document_name] : the name given to a document being edited.

[filename] : name given to a file.

[newline] : the control character used to mark the end of a line of text.

## 1.5. References

Bourne, S.R. The UNIX System. London, Addison-Wesley, 1983.

Kernigham, B.W. and Pike, R. The UNIX Programming Environment. New Jersey, Prentice_hall, 1984. pp. 319-328.

Poole, P.C. and Poole, N. Using UNIX by example. Sydney, Addison_Wesley, 1986.

SUN Microsystem using the help function ($man command).

SUN OS Reference Manual. Part number 800-1751-10, revision A, May 9 1988.

## 1.6. Overview

The following pages will provide further information on the editor system to be designed.

In Chapter 2, section 2.1 discusses the editor as a software that allows the user to create a document and effect changes to this file, one line at a time. The functions provided by the editor in order to achieve document creation and updating are identified in Section 2.2. The following section, Section 2.3, explicitely denotes the underlying assumption made about the environment. Any hardware constraints are listed in section 2.4 to describe the environment in which the editor works and the interface between command mode and the editor.

Chapter 3 deals with the specific requirements of the editor. Section 3.1 describes the functional requirements based on a functional characteristic. Section 3.1 contains a brief description of each functionality along with their attributes and restrictions. Each function is specified using a standard format in order to present and enforce consistency in representation. This section not only describes the necessary functions of the system but it also provides preliminary guidelines by which the functions may be verified during implementation. Although Section 3.1 provides some minimal data about limitations and timing requirements, Section 3.2 further elaborates on the non-functional characteristics of the system. Attributes concerning the integrated components of the system are discussed in Section 3.3. Finally, Section 3.4 provides a set of guidelines by which the system will be tested for acceptance.

## 2. GENERAL DESCRIPTION

### 2.1. Product Perspective

#### 2.1.1. What is ned

Ned is a general purpose line editor which is provided as a utility in the UNIX operating system environment. "ed" is a product currently available in this environment which can be compared to ned in scope. Ned is actually a scaled down version of ed with some of ed's functionalities left out. It is a line oriented tool rather than a full screen display editor and thus provides functions which use the line of text as the basic unit of manipulation. Ned allows users in this environment to create new text files and/or modify already existing ones.

#### 2.1.2. Buffers, files, documents

The contents of the buffer is called a document. Once the document is transferred to secondary storage it is known as a file. A file that is transferred to the editor's buffer becomes a document.

#### 2.1.3. Modes of Operation

The editor operates in two modes and can only be in one of these modes at a time. These are the command mode and the input mode. Note that ned does not indicate to the user which mode he/she is in (see section 3.1.3).

##### 2.1.3.1. Command mode

When in command mode, the editor provides line oriented editing commands that allow the user to manipulate lines of text in a document (see section 3.1). The commands may require 0,1, or 2 line position addresses within the document. These include commands to insert, append, delete, and display lines. The command mode of the editor also provides commands for storing or retrieving a file, and leaving the editor. When in this mode, the user can only enter commands.

##### 2.1.3.2. Input mode

When the editor is in input mode , the user can write lines of text into the document. Ned commands cannot be issued in the input mode. Just as the command mode is strictly for command input, the input mode is strictly used for inputting lines of text into the document.

#### 2.1.4. What is expected of user

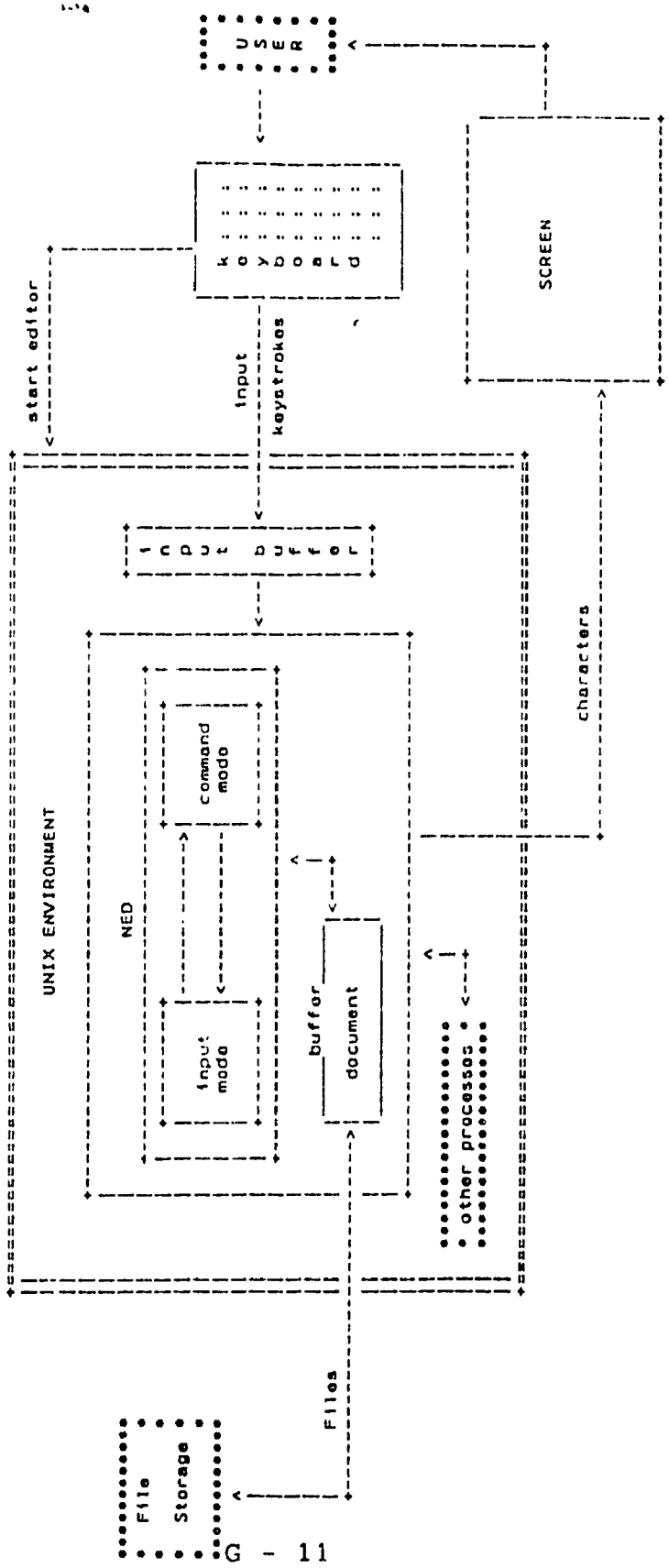
It is assumed that the user is familiar with the UNIX operating system environment and the SUN station. Since ned operates in two modes, the user will be expected to enter ned commands when in command mode or a line of text when in input mode.

#### 2.1.5. Contextual Aspect

Ned will mainly be used by users who would like to create or make changes to a text file. The editor could also be made available to any process running under the UNIX environment.

### 2.1.6. System model

#### 2.1.6.1. Model



#### 2.1.6.2. Description of the model

The user invokes ned from the UNIX prompt. When the editor is entered , it is in command mode by default .The keyboard is the input device which allows the user to input keystrokes to the editor. These keystrokes show up on the user's screen as characters from the standard ASCII character set. Before reaching the editor, these keystrokes are stored in the input buffer (provided by the Unix environment) which acts as the interface to ned. The editor can switch back and forth from command mode to input mode. All operations act on the document in the ned buffer which is either loaded in to the buffer from file storage or stored to file storage. Since UNIX allows multiple processes to be executing at the same time (see section 2.4), while the editor is executing there may be other processes running in the system. These other processes may be another ned process or maybe even a process which relies on the use of ned.

### 2.1.7. Future Requirements

Future requirements may call for improvement of the command syntax, error reporting, and the on line help which is currently provided by ed. Ned currently does not provide any on-line help. The editor may be required to show line numbers which indicate relative line positions in the document. The showing of invisibles such as blank spaces, tab characters and newline characters may also be required in the future. Ned may be required to support the editing of multiple documents and although it is currently a line oriented tool, it may be upgraded to a full screen editor. Ned is currently supported by the UNIX environment running on SUN workstations however, it may be desirable to port this editor to another system. Currently, ned does not provide the user with information concerning the current mode of the editor. This is another requirement which may be added in the future.

## 2.2. Product functions

The ned editor performs file manipulation tasks. It allows the user to create or modify a file, by loading it into the buffer as a document (see section 3.1.8). It is then possible to manipulate the content of the buffer by inserting, appending, deleting and displaying one or more line(s) of text. In order to keep those changes it is possible to save the content of the buffer to a secondary storage via a ned service.

### 2.2.1. Main tasks

In order to edit a file, the editor should allow the following main tasks:

- LOAD a file.
- CREATE a document.
- INSERT line(s).
- APPEND line(s).
- DELETE line(s).
- DISPLAY line(s).
- SAVE a document.

#### 2.2.1.1. Load a file:

Takes a file from secondary storage and puts it into memory. The file can be loaded when you invoke the editor. It can also be loaded but once your in it, the editor will clear the buffer before loading the new file.

#### 2.2.1.2. Create a document:

Ned reserves space in the buffer for a new document.

#### 2.2.1.3. Insert line(s):

Allows user to add one or more line of text before a specified position in the document. The new line(s) will push down the rest of the document.

#### 2.2.1.4. Append line(s):

Allows user to add one or more line of text after a specified position in the document. The new line(s) will push down the rest of the document.

#### 2.2.1.5. Delete line(s):

Allows user to remove one or more line of text from a specified position in the document. The other line positions will be updated.

#### 2.2.1.6. Display line(s):

Shows onto the screen the requested line(s) from the document. The user specifies the range of line(s) to display.

#### 2.2.1.7. Save a document:

Writes the content of the memory buffer to secondary storage. The user can specify to which file the buffer should be written to.

### 2.3. Assumptions

In order for ned to function in a known environment, the following assumptions have been made:

- The environment will be supplied by the UNIX operating system.

- The keyboard will supply the basic typewriter character set.

- The TAB and DELETE keys will provide the same action they do in the UNIX environment.

- Only the basic "ed" command outlined in the handout given in class will be described.

- The user should know the purpose of the editor and be familiar with the UNIX operating system.

- Screen handling facilities will be controlled by the UNIX operating system.

## 2.4. Constraints

-Ned will only work on one document at a time.

-The maximum size of the buffer that ned can work on is limited by the memory available on the workstation.

-The maximum number of files that ned can create is limited by the user's disk quota.

-The editor will not occupy more than 400 Kbytes of main memory.

-The user cannot edit a file for which he does not have write access.

-A line may contain up to 256 characters (including the NEWLINE character).

-The maximum number of editors per user is limited by the operating system.

-Ned will use the ASCII standard for character representation. It will conform to all relevant standards used by the UNIX operating system and Sun systems.

-It is possible for users to connect remote terminals to the SUNs through the PACX. network at Concordia University. The user must have a modem set up according to the recommendations of the Concordia University Computer Center to ensure proper operation of Ned.

## 3. SPECIFIC REQUIREMENTS

All functional requirements in section 3.1 will have the following format:

NAME : ( name of the function ).

PURPOSE: ( the aim of the function. ie: what the function does ? )

INPUT: ( information required to execute the function. )

ACTION: ( the action performed after the command is executed. )

RESULT: ( result of the action. )

EXCEPTIONS: ( cases where the required action is not performed .)

PERFORMANCE CONSTRAINTS: ( conditions under which the function will have to perform. )

TIMING CONSTRAINTS: (time limits imposed on the function)

PERIODICITY: ( frequency of use. )

REMARKS: ( other relevant information. )

Note: (a) The notations "(i)","(ii)"... will be used to differentiate different INPUT options available for the function. For each of the "(i)" ... there are corresponding (i)""(ii)"... in ACTION and RESULT.

(b) "ln1" and "ln2" are values that give the address of the line in the document.

(c) "string1" is a string of text(see section 1.4).

### 3.1. Functional Requirements

In these requirements we assume that the current mode and current line position is always known.

#### 3.1.1. Append_line

NAME : Append_line

PURPOSE : To append a line into the document after the addressed line.

INPUT : i) a) ln1 [document]
b) a line of text
ii) a) string1 [document]
b) a line of text.

ACTION : i) The line of text is inserted into the document after the line currently at position ln1.
ii) The line of text is inserted after the first line in the document containing string1.

RESULTS : i) a) The document contains an additional line which is the new line of text at position (ln1+1).

b) The current line position is (ln1 + 1).

c) All lines in the document after and including the line that was at position (ln1 + 1) before the appending (if any), are moved down by one line position. Nothing is moved down if ln1 was the last line of the document before the appending took place.

Example: Text in the document

<ln1> this is full
<ln1+1> use for the
<ln1+2> testing.....
<ln1+3> help me God.

After the following text is appended at ln1:
ABCDEFGHIJ

<ln1> this is full
<ln1+1> ABCDEFGHIJ
<ln1+2> use for the
<ln1+3> testing.....
<ln1+4> help me God.

ii) a) Let the position of the first line in the document containing string1 be denoted by first_with_string. The document contains the new line of text at position first_with_string + 1.

b) The current line position is first_with_string + 1.

c). All lines in the document after and including the line that was at position first_with_string before the appending (if any), are moved down by one line position in the document. Nothing is moved down if first_with_string was the last line of the document before the appending took place.

EXCEPTIONS : i) a) ln1 <= 0

b) ln1 > (number of lines in the document)

c) ln1 is not a numerical value.

ii) string1 does not exist in the document.

PERFORMANCE CONSTRAINTS : Nil

TIMING CONSTRAINTS : Nil

PERIODICITY : Upon user request

REMARK : If lnl is not given then the current line position is used by default. Lines of text can only be appended in input mode.

### 3.1.2. Change_current_line

NAME : Change_current_line

PURPOSE : To change the current line position in the document.

INPUT : ln1 [document]

ACTION : The current line is reset to point to the line specified by the line number, ln1 in the document.

RESULT : The current line is ln1.

EXCEPTIONS : a) ln1 <= 0
b) ln1 > (number of lines in document)
c) ln1 is not numeric.

PERFOMANCE CONSTRAINTS : Nil

TIMING CONSTRAINTS : Nil

PERIODICITY : Upon user request

REMARKS : Can only be done in command mode.

### 3.1.3. Change_mode

NAME : Change_mode

PURPOSE : To toggle between command mode and input mode.

INPUT : Nil

ACTION : i) If the current mode is in input mode then the editor is switched to command mode.
ii) If the current mode is in command mode the editor is switched to insert mode.

RESULT : i) The editor is in command mode.
ii) The editor is in input mode.

EXCEPTIONS : Nil

PERFORMANCE CONSTRAINTS : Nil

TIMING CONSTRAINTS : Nil

PERIODICITY : Upon user request

REMARKS : Nil

### 3.1.4. Create_document

NAME : Create_document

PURPOSE : To initialize the buffer (in memory) for a new document

INPUT : [document_name]

ACTION : a) The buffer is cleared for a new document with specified name [document_name].
b) If no document name is given then the buffer is cleared for a new document with no name.

RESULT : An empty initial buffer ready to read in the new document.The editor is in command mode.The current line position is 0.

EXCEPTIONS : Nil

PERFORMANCE CONSTRAINTS : Nil

TIMING CONSTRAINTS : Nil

PERIODICITY : Upon user request

REMARKS : If no document name is given it can optionally be given a name upon writing the document into a file (see section 3.1.13).

### 3.1.5. Delete_line(s)

NAME : Delete_line(s)

PURPOSE : To delete a line or a set of consecutive lines from the document.

INPUT : ln1, ln2 [document]

ACTION : The set of lines starting from ln1 to ln2 inclusive, are deleted from the document.

RESULT : a) The set of lines between ln1 and ln2 inclusive are removed from the document.
b) The line that was at position (ln2+1) becomes the current line if ln2 was not the last line in the document. Otherwise, the current line position becomes ln1 - 1.
c) All lines (if any) after the line that was at position ln2 are moved up by w positions where

w = no. of lines deleted

Example: Text in the document:

<ln1> this is full
<ln2> use for the
<ln3> testing.....
<ln4> help me God.

After delete ln1, ln2:

Text in the document:

<ln1> testing.....
<ln2> help me God.

EXCEPTIONS : a) ln1 > ln2
b) ln1 <= 0
c)ln2 > (number of lines in document)

PERFORMANCE CONSTRAINTS : Nil

TIMING CONSTRAINTS : Nil

PERIODICITY : Upon user request

REMARKS : a) Input is not specified then the current line is deleted from the document.
b) This can only be used in command mode.

### 3.1.6. Display_line(s)

NAME : Display_line(s)

PURPOSE : To display selected line(s) from the document onto the screen.

INPUT : i) ln1, ln2 [document]
ii) string1, ln2 [document]

ACTION: i) Lines ln1 to ln2 inclusive, in the document are displayed on the screen.
ii) Lines between the first line in the document containing string1 and ln2 inclusive, are displayed on the screen.

RESULT: The specified lines are displayed on the screen. The current line position will be ln2.

EXCEPTIONS: a) ln1 > ln2
b) ln1 <= 0
c) ln2 > (number of lines in document)
d) string1 does not exist in the document.

PERFORMANCE CONSTRAINTS: Nil

TIMING CONSTRAINTS: Nil

PERIODICITY: Upon user request

REMARKS: This can only be used in command mode.

### 3.1.7. Insert_line

NAME : Insert_line

PURPOSE : To insert a line into the document before the addressed line.

INPUT : i) a) ln1 [document]
b) a line of text.
ii) a) string1 [document]
b) a line of text.

ACTION : i) The line of text is inserted into the document before the line currently at position ln1.
ii) The line of text is inserted before the first line in the document containing string1.

RESULTS : i) a) The document contains an additional line which is the new line of text at position ln1.
b) The current line position is ln1.
c) All lines in the document after and including the line that was at position ln1 before the insertion , are moved down by one line.

Example: Text in the document:

<ln1 > this is full.
<ln2 > use for the
<ln3 > testing.....
<ln4 > help me God.

After insert the following text at ln1:
ABCDEFGHIJ

Text in the document:

<ln1 > ABCDEFGHIJ
<ln2 > this is full.
<ln3 > use for the
<ln4 > testing.....
<ln5 > help me God.

ii) a) let the position of the first line in the document containing string1 be denoted by first_with_string. The document contains the new line of text at position first_with_string.
b) The current line position is first_with_string.
c) All lines in the document after and including the line that was at position first_with_string before the insertion,

are moved down by one line.

EXCEPTIONS : i) a) ln1 <= 0
b) ln1 > (number of lines in the document)
c) ln1 is not a numerical value.
ii) a) string1 does not exist in the document.
b) The document is empty.

PERFORMANCE CONSTRAINTS : Nil

TIMING CONSTRAINTS : Nil

PERIODICITY : Upon user request

REMARKS : If ln1 is not given then the current line is used by default. Lines of text can only be inserted in insert mode.

3.1.8. Load_document_from_file

NAME : Load_document_from_file

PURPOSE : To load a file in secondary storage into the buffer.

INPUT : [filename]

ACTION : Fetches the entire contents of the file [filename] into the buffer. The content of the buffer becomes the new document and the [document_name] is given the [filename] by default.

RESULT : The contents of the file is loaded into the buffer and a numeric value stating the number of characters in the file is displayed on the screen. The current line pointer is the last line in the document. The editor is in command mode.

EXCEPTIONS : File does not exist.

PERFORMANCE CONSTRAINTS: Cannot be edited if sufficient memory is not available.

TIMING CONSTRAINTS : nil

PERIODICITY : Upon user request

REMARKS : The previous contents of the buffer is cleared before loading of the file. The editor must be in command mode.

### 3.1.9. Quit

**NAME : Quit**

**PURPOSE : To exit the editor.**

**INPUT : Nil**

**ACTION : Clears the buffer and leaves the editor.**

**RESULT : Returns to shell prompt.**

**EXCEPTION : If changes made to the file are not saved.**

**PERFORMANCE CONSTRAINTS : Nil**

**TIMING CONSTRAINTS : Nil**

**PERIODICITY : Upon user request**

**REMARKS : Warns you if you have not saved the changes..**

3.1.10. Write_lines_from_buffer_to_file

NAME : Write_lines_from_buffer_to_file

PURPOSE : To write the addressed lines from the buffer into a file on secondary storage.

INPUT : i) ln1, ln2 [document_name]
ii) ln1, ln2 [filename]
iii) [filename]
iv) [document_name]

ACTION : For cases i) through iv) if the file does not exist, it is created (see 3.1.4) otherwise it is overwritten.

i) Starting at ln1, all lines between ln1 and ln2 inclusive are written to secondary storage under the name [document_name].
ii) Starting at ln1, all lines between ln1 and ln2 inclusive are written to the file named [filename].
iii) If no line numbers are included, then the entire contents of the buffer is written to the file [filename].
iv) If no line numbers, or filename are given then the entire contents of the buffer are written to the file with the current document's name which is [document_name].

RESULT : i) The file with name [document_name] contains only the lines between ln1 and ln2 inclusive and stored in secondary storage.
ii) The file named [filename] contains only the lines between ln1 and ln2 inclusive, and stored in secondary storage.
iii) The file named [filename] contains an exact copy of the current buffer contents, and is stored in secondary storage.
iv) The file with name [document_name] contains an exact copy of the current buffer contents and is stored in secondary storage.

(for all cases) The line pointer somethimes called the current line position will remain at the same position as before the invocation. Then it will return to the command mode.

EXCEPTIONS : a) ln1 > ln2
b) ln1 <= 0
c) ln2 > (number of lines in document)

PERFORMANCE CONSTRAINTS : Cannot be written if insufficient space

in secondary storage.

TIMING CONSTRAINTS : Nil

PERIODICITY : Upon user request

REMARKS : Nil

### 3.2. Overall Performance Requirements.

#### 3.2.1. Accuracy

The system being implemented is not overly concerned with accuracy as far as numeric values are concerned. Accuracy in our system means being able to produce the desired result in an predefined manner. In other words, it should execute commands as specified in the requirements. This of course will be taken care of in the verification and validation phase.

#### 3.2.2. Timing Constraints.

The editor being implemented works in an interactive environment. Because of this nature, the response time is an important factor. Although, the time requirement is considered it is not critical since the speed of response is not directly related to any safety measures that would have to be implemented in the case of emergency. We expect the commands to be performed within 30 seconds.

### 3.3. Attributes and documents to be delivered

#### 3.3.1. Attributes

Extendibility

So that future requirements may be correctly and easily implemented, the system must be developped in such a way as to allow modification.

Flexibility

Ned is very flexible because all commands are on user request. It allows the user to make almost any changes to a document as required.

Portability

Ned should be able to be easily ported to either a DOS environment or another UNIX environment other than SUN.

Reliability

Ned must perform the tasks specified properly. Not only does it gives the user the ability to perform the desired tasks, but it takes into consideration abnormal conditions.

Safety

Ned does not result in writing out the contents of the buffer into a file in the wrong order.

Security

The integrity of the code should be ensured by the operating system software already installed in the targeted system.

Training Time

The average user will be able to learn and understand all the operations of the editor in less than one hour.S

#### 3.3.2. Documents to be delivered

Review of ned requirements, specifications, preliminary design and critical design would be conducted in four stages in section 3.4 prior to the schedule of the review:

a) Software requirement document

b) Design document

c) Implc.- .ıtation and test results

d) User's manual for ned

### 3.4. Acceptance Criteria

Ned must be tested by the user, the problem analyst and the developers before the system can be accepted. The reason for having these checks are to avoid outrageous maintenance cost in the future. Procedural steps in analyzing the system are as follows:

Stage I : System Concept Review (SCR)

This phase of the testing is carried out by the problem analyst and a professinal in formal language. The SCR would result in:

(i) unambiguous and consistent requirements

(ii) formal specifications

The SCR is derived from the SRD

SRD ---> Formal Specifications

Stage II : Preliminary Design Review (PDR)

In order to be able to evaluate the system, ned will be implemented in a high level language such as C or Pascal on the SUN workstations. From the simulation of our editior, feedback that is obtained can be used to cross check against benchmark values. The performance evaluation of this phase can be obtained from the formal specifications, the executable ned design on the SUN and the benchmark values.

Formal --> Executable ned --> Performance Specifications Evaluation

Stage III : Critical Design Review (CDR)

This phase is a straight implementation of the interface between our software and the SUN workstation. Details about the SUN environment are researched and incorporated. Note, at this point, consideration to portability, compatability and data flow are required.

The detailed design is derived from the formal specifications

Formal Specifications -----> Detailed Design of ned
on target system

Stage IV: Line Editor Review

In order to review our system, the developers test each functionality through a process known as exhaustive proofs. Every possible scenario that can occur will be enacted. Comparisons will be made between the results and the expected outcomes produced.

ned design on the -----> Performance Evaluation
SUN workstation

Stage V : Acceptance Review

The Acceptance Review consists of a hands-on demonstration that walks through the steps of creating a file with edit, updating the file using append, delete and insert and finally saving the file using write. At this point, all the features will be presented for final approval.

# Appendix H

# The infoMap for the NED SRD

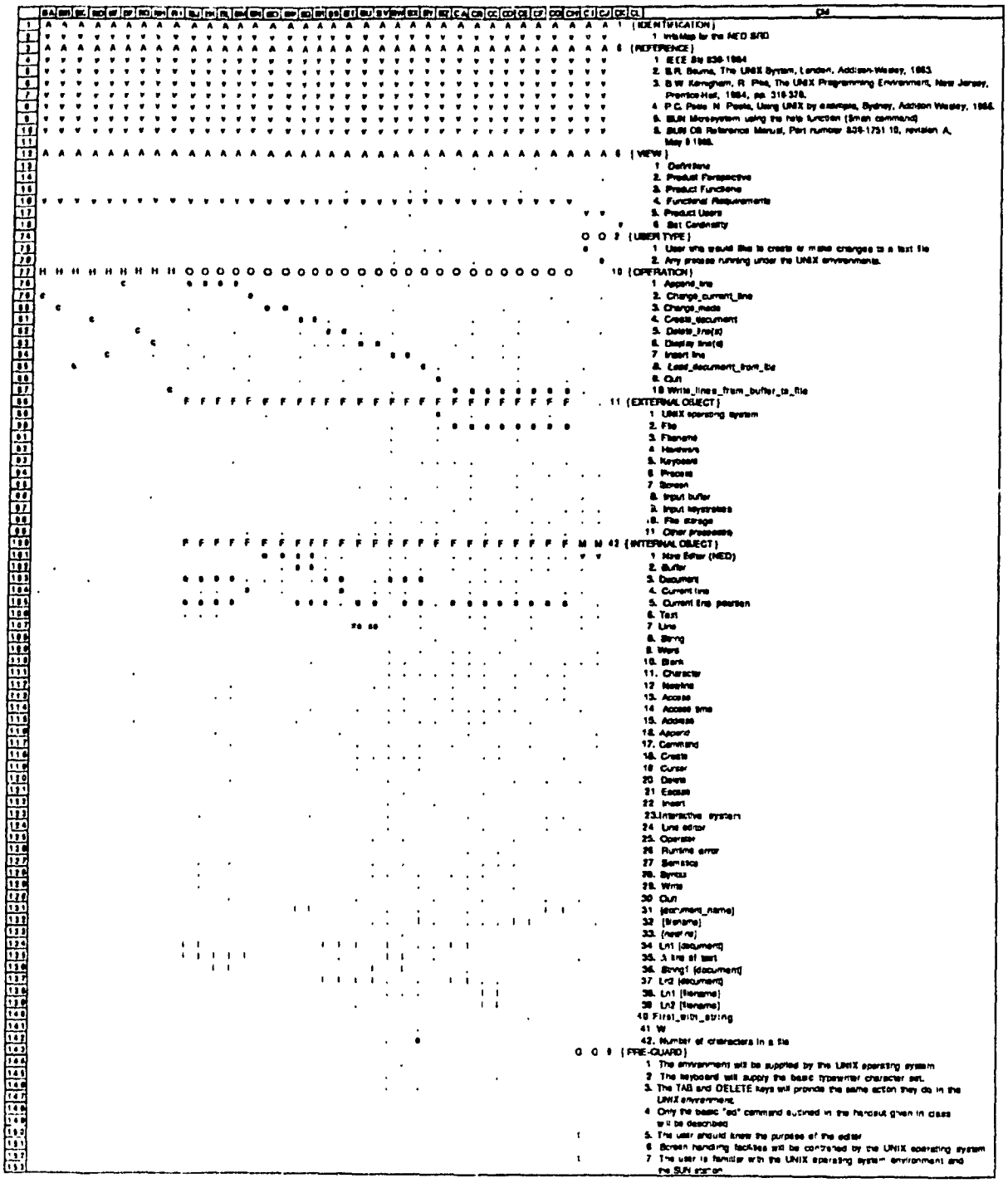
| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | A | A | 1 | { IDENTIFICATION } | |
| 2 | v | v | v | v | v | v | v | | | 1. infoSchema for the NED SRD. |
| 3 | A | A | A | A | A | A | A | 6 | { REFERENCE } | |
| 4 | A | A | A | A | A | A | A | 6 | { VIEW } | |
| 5 | v | . | . | . | . | . | . | | | 1. Definitions |
| 6 | . | v | . | . | . | . | . | | | 2. Product Perspective |
| 7 | . | . | v | v | . | . | . | | | 3. Product Functions |
| 8 | . | . | . | . | v | v | . | | | 4. Functional Requirements |
| 9 | . | . | . | . | . | . | v | | | 5. Product Users |
| 10 | . | . | . | . | . | . | . | v | | 6. Set Cardinality |
| 11 | O | . | . | . | . | . | . | 41 | { DEFINITION } | |
| 12 | . | . | . | . | . | . | O | 2 | { USER TYPE } | |
| 13 | . | . | . | . | H | O | . | 10 | { OPERATION } | |
| 14 | T | H | . | . | . | F | . | 11 | { EXTERNAL OBJECT } | |
| 15 | T | H | O | O | . | F | M | 42 | { INTERNAL OBJECT } | |
| 16 | . | . | G | . | . | . | G | 9 | { PRE-GUARD } | |
| 17 | . | . | . | . | . | G | . | 18 | { PRE-CONDITION } | |
| 18 | . | . | . | . | . | G | . | 1 | { EVENT } | |
| 19 | . | . | . | M | H | . | . | 18 | { TASK } | |
| 20 | . | . | . | . | . | S | . | 24 | { ACTION } | |
| 21 | . | . | . | . | . | G | . | 30 | { POST-CONDITION } | |
| 22 | . | . | G | . | . | G | . | 31 | { POST-GUARD } | |
| 23 | . | . | . | M | . | M | . | 6 | { REMARK } | |

| | | |
|---|---|---|
| 1 | 1 | {IDENTIFICATION} |
| 2 | | 1 Initials for the NED BRID |
| 3 | 6 | {REFERENCE} |
| 4 | | 1 IEEE Std 830-1984 |
| 5 | | 2. B.R. Bourne, The UNIX System, London, Addison-Wesley, 1983. |
| 6 | | 3. B.W Kernighan, R. Pike, The UNIX Programming Environment, New Jersey, Prentice-Hall, 1984, pp. 318-326. |
| 7 | | 4 P.C. Poole, M. Poole, Using UNIX by examples, Sydney, Addison-Wesley, 1986. |
| 8 | | 5. SUN Microsystem using the help function (Sman command). |
| 9 | | 6. SUN OS Reference Manual, Part number 800-1751-10, revision A, May 9 1988. |
| 10 | | |
| 11 | 6 | {VIEW} |
| 12 | | 1 Definitions |
| 13 | | 2. Product Perspective |
| 14 | | 3. Product Functions |
| 15 | | 4. Functional Requirements |
| 16 | | 5. Product Users |
| 17 | | 6 Set Cardinality |
| 18 | 41 | {DEFINITION} |
| 19 | | 1. A file may be accessed for reading, writing or appending. |
| 20 | | 2. The time required to access an item of data on a storage device |
| 21 | | 3. Specifies a particular line position in the document. |
| 22 | | 4. To add information to the end of a document, or line without altering the information already present. |
| 23 | | 5. A space or tab character |
| 24 | | 6. A storage area used by ned to hold information temporarily. |
| 25 | | 7 Represent symbols people use for written communication; any keystroke from the keyboard is represented by an ASCII code that includes all the possible symbols in our scope. |
| 26 | | 8. An instruction given to the editor by the user to perform a task. |
| 27 | | 9 To bring into existence (file or document). |
| 28 | | 10. The presently selected line of text (implicitly indicated). |
| 29 | | 11 Refers to position/address of the current line in the document. |
| 30 | | 12. An indicator on the screen which shows where the next character will be displayed. |
| 31 | | 13. Remove, thereby freeing the space a line(s) occupies in the document. |
| 32 | | 14. The current contents of the buffer is collectively known as the document. |
| 33 | | 15. A temporary exit from a command from one mode of operation to another |
| 34 | | 16 A mechanism for the long-term storage of information in a computer system |
| 35 | | 17 A symbolic name used to reference a file |
| 36 | | 18. The collective name given to the electronic and electro-mechanical devices that go together to make a computer |
| 37 | | 19. To place line(s) at a line position in the document without altering the information already present. |
| 38 | | 20. A system designed to carry out a dialogue with a user, reads commands from a terminal and then executes them immediately |
| 39 | | 21. An input device which generates a signal representing a character corresponding to the key that has been pressed by the user |
| 40 | | 22. A sequence of characters terminated by a newline which is generated by the RETURN key (carriage return). |
| 41 | | 23. A type of editor that allows the user to locate lines in a file so that the text in the file may be changed. |
| 42 | | 24 A control character used to mark the end of a line of text; usually generated by striking the RETURN key |
| 43 | | 25. A symbol defining the action to be performed on one or more [illegible] inputs to obtain a result. |
| 44 | | 26. An executing program |
| 45 | | 27 Errors which occur when a program is executing, logical or semantic errors |
| 46 | | 28. Video display unit. |
| 47 | | 29. The meaning of symbols and groups of symbols uses in a programming language. |
| 48 | | 30. A sequence of characters usually terminated by a delimiter |
| 49 | | 31. The set of rules defining the grammar of a programming language |
| 50 | | 32 One or more lines composed of printable characters. |
| 51 | | 33. A sequence of characters in a line of text delimited in some way |
| 52 | | 34 The output of data from a buffer to an external peripheral device |
| 53 | | 35 Exiting from a process. |
| 54 | | 36. The name given to a document being edited |
| 55 | | 37. Name given to a file. |
| 56 | | 38. The control character used to mark the end of a line of text. |
| 57 | | 39. Value that gives the address of the line in the document. |
| 58 | | 40 The position of the first line in the document containing string1. |
| 59 | | 41 Number of lines deleted. |
| 60 | 11 | {EXTERNAL OBJECT} |
| 61 | | 1 UNIX operating system. |
| 62 | | 2. File |
| 63 | | 3. Filename |
| 64 | | 4. Hardware |
| 65 | | 5. Keyboard |
| 66 | | 6. Process |
| 67 | | 7. Screen |
| 68 | | 8. Input buffer |
| 69 | | 9. Input keystrokes |
| 70 | | 10. File storage |
| 71 | | 11 Other processes |
| 100 | 42 | {INTERNAL OBJECT} |
| 101 | | 1 New Editor (NED) |
| 102 | | 2. Buffer |
| 103 | | 3. Document |
| 104 | | 4. Current line |
| 105 | | 5. Current line position |
| 106 | | 6. Text |
| 107 | | 7 Line |
| 108 | | 8. String |
| 109 | | 9. Word |
| 110 | | 10. Blank |
| 111 | | 11 Character |
| 112 | | 12. Newline |

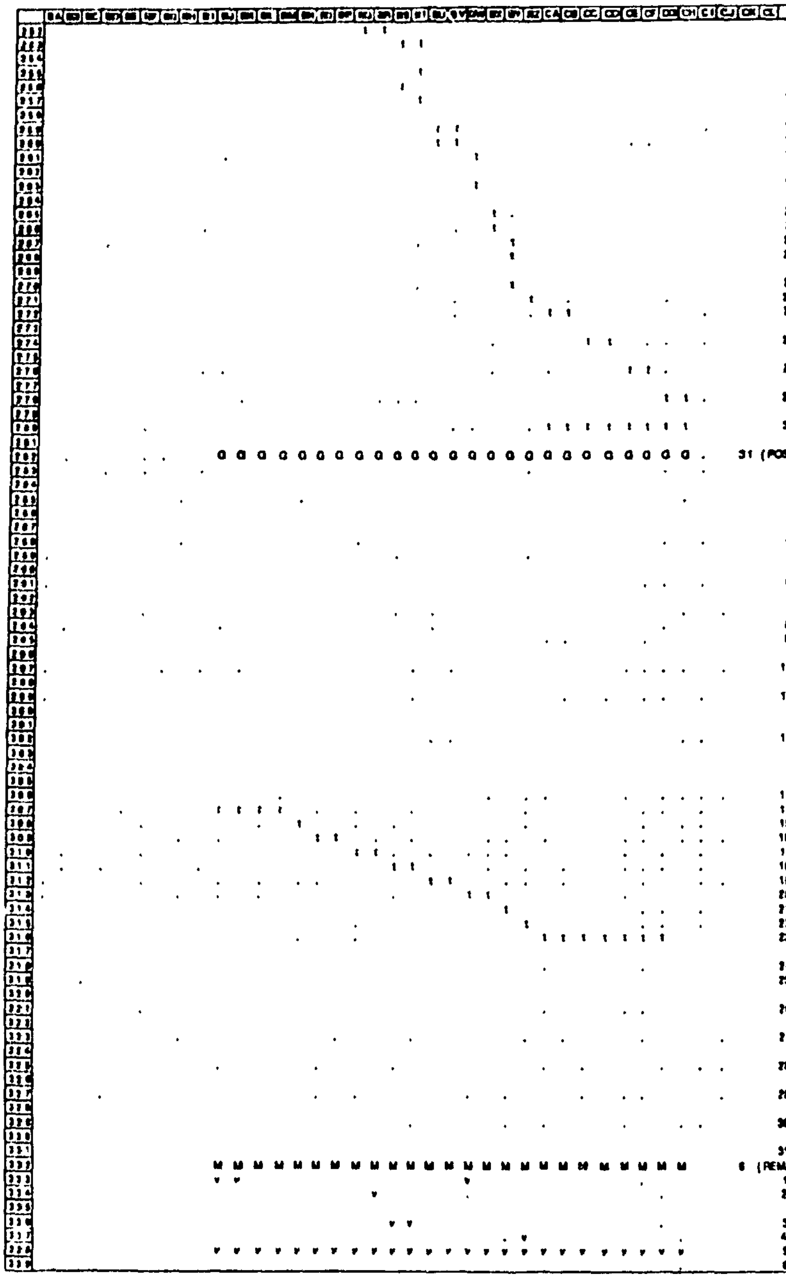
| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | AA | AB | AC | AD | AE | AF | AG | AH | AI | AJ | AK | AL | AM | AN | AO | OR | CL | Ok |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 113 | ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 13. Access |
| 114 | | ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 14. Access time |
| 115 | | | ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 15. Address |
| 116 | | | | ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 16. Append |
| 117 | | | | | | | | ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 17. Command |
| 118 | | | | | | | | | ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 18. Create |
| 119 | | | | | | | | | | | | ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 19. Cursor |
| 120 | | | | | | | | | | | | | ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 20. Delete |
| 121 | | | | | | | | | | | | | | ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 21. Escape |
| 122 | | | | | | | | | | | | | | | | | | | ▼ | | | | | | | | | | | | | | | | | | | | | | | | | 22. Insert |
| 123 | | | | | | | | | | | | | | | | | | | | ▼ | | | | | | | | | | | | | | | | | | | | | | | | 23. Interactive system |
| 124 | | | | | | | | | | | | | | | | | | | | | | | ▼ | | | | | | | | | | | | | | | | | | | | | 24. Line editor |
| 125 | | | | | | | | | | | | | | | | | | | | | | | | | ▼ | | | | | | | | | | | | | | | | | | | 25. Operator |
| 126 | | | | | | | | | | | | | | | | | | | | | | | | | | ▼ | | | | | | | | | | | | | | | | | | 26. Runtime error |
| 127 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ▼ | | | | | | | | | | | | | | | 27. Semantics |
| 128 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ▼ | | | | | | | | | | | | | | 28. Syntax |
| 129 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ▼ | | | | | | | | | | | | | 29. Write |
| 130 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ▼ | | | | | | | | | | | | 30. Quit |
| 131 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ▼ | | | | | | | | | | 31. [document_name] |
| 132 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ▼ | | | | | | | | | 32. [filename] |
| 133 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ▼ | | | | | | | | 33. [message] |
| 134 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ▼ | | | | | | | 34. Ln1 [document] |
| 135 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 35. A line of text |
| 136 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 36. String1 [document] |
| 137 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ▼ | | | | | | 37. Ln2 [document] |
| 138 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ▼ | | | | | | 38. Ln1 [filename] |
| 139 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ▼ | | | | | | 39. Ln2 [filename] |
| 140 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ▼ | | | | | 40. First_with_string |
| 141 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ▼ | | | | 41. W |
| 142 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 42. Number of characters in a file |

| | AP | AQ | AR | AS | AT | AU | AV | AW | AX | AY | AZ | CN | CL | CM |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | A | A | A | A | A | A | A | A | A | A | 1 | | {IDENTIFICATION} |
| 2 | v | v | v | v | v | v | v | v | v | v | v | | | 1 Intended for the NED BRD |
| 3 | A | A | A | A | A | A | A | A | A | A | A | 6 | | {REFERENCE} |
| 4 | v | v | v | v | v | v | v | v | v | v | v | | | 1 IEEE Std 830-1984 |
| 5 | v | v | v | v | v | v | v | v | v | v | v | | | 2 S.R. Bourne, The UNIX System, London, Addison-Wesley, 1983. |
| 6 | v | v | v | v | v | v | v | v | v | v | v | | | 3. B.W. Kernighan, R. Pike, The UNIX Programming Environment, New Jersey, |
| 7 | v | v | v | v | v | v | v | v | v | v | v | | | Prentice-Hall, 1984, pp. 318 326. |
| 8 | v | v | v | v | v | v | v | v | v | v | v | | | 4 P.C. Poole, N Pooe, Using UNIX by example, Sydney, Addison Wesley, 1984. |
| 9 | v | v | v | v | v | v | v | v | v | v | v | | | 5. SUN Microsystem using the new function (Smart command) |
| 10 | v | v | v | v | v | v | v | v | v | v | v | | | 6. SUN OS Reference Manual, Part number 800-1751 18, revision A, |
| 11 | | | | | | | | | | | | | | May 9 1988. |
| 12 | A | A | A | A | A | A | A | A | A | A | A | 8 | | {VIEW} |
| 13 | | | . | . | | | | | | | | | | 1 Definitions |
| 14 | v | v | v | v | v | v | v | v | | | . | | | 2. Product Perspective |
| 15 | . | . | | . | | | | . | v | v | v | | | 3. Product Functions |
| 16 | | . | . | | | . | . | | . | | | | | 4. Functional Requirements |
| 17 | . | . | . | . | . | | . | | | | | | | 5. Product Users |
| 18 | . | . | | . | | | | | | . | . | v | | 6 Bat Constraints |
| 86 | H | H | H | H | H | H | H | H | | | | 11 | | {EXTERNAL OBJECT} |
| 89 | p | | . | | . | | . | | . | | | | | 1 UNIX operating system |
| 90 | | . | . | . | . | | | . | | | | | | 2. File |
| 91 | . | | . | . | | | | | | | | | | 3. Filename |
| 92 | | . | | | | | . | | | | . | | | 4 Hardware |
| 93 | . | | | . | . | . | | . | | | | | | 5. Keyboard |
| 94 | . | | . | . | . | | | | . | . | | | | 6 Process |
| 95 | . | | | . | | . | . | . | . | . | | | | 7 Screen |
| 96 | . | | | | . | | . | | | . | | | | 8. Input buffer |
| 97 | . | | | . | . | | | . | . | . | | | | 9. Input keystroke |
| 98 | . | . | . | | . | . | . | . | . | . | . | | | 10 File storage |
| 99 | . | . | . | . | | . | | . | . | . | . | | | 11 Other processes |
| 100 | H | H | H | H | H | H | H | H | O | O | O | 42 | | {INTERNAL OBJECT} |
| 101 | e | p | . | . | . | | . | | e | e | e | | | 1 New Editor (NED) |
| 102 | . | c | p | . | | | | | . | . | | | | 2. Buffer |
| 103 | . | . | c | p | . | . | . | | . | . | . | | | 3. Document |
| 104 | . | | . | e | . | . | | | . | | . | | | 4. Current line |
| 105 | . | | | e | | | | . | | | . | | | 5.Current line position |
| 106 | . | . | . | c | p | | | | . | | | | | 6. Text |
| 107 | . | . | | | c | p | | | | | . | | | 7 Line |
| 108 | . | . | . | . | | | p | | . | | | | | 8. String |
| 109 | . | . | . | | | | | p | | | | | | 9. Word |
| 110 | | . | . | . | | c | c | c | | | | | | 10. Blank |
| 111 | . | . | . | | | c | c | c | | | . | | | 11 Character |
| 112 | . | . | . | . | . | c | . | | . | | . | | | 12 Newline |
| 113 | | | | | | | . | | . | . | . | | | 13. Access |
| 114 | . | . | | . | | | . | | . | . | . | | | 14 Access time |
| 115 | . | . | | . | | | | | . | | . | | | 15. Address |
| 116 | . | . | . | | | | . | | . | . | | | | 16. Append |
| 117 | . | . | . | . | . | . | | . | . | . | . | | | 17. Commands |
| 118 | . | . | . | . | . | | . | | . | | . | | | 18. Create |
| 119 | . | | . | . | . | . | | . | . | | . | | | 19 Cursor |
| 120 | . | . | | | | | . | | . | | . | | | 20 Delete |
| 121 | . | . | | | | | . | . | . | | . | | | 21 Escape |
| 122 | . | | | | | | | | . | | . | | | 22 Insert |
| 123 | . | . | . | | | | | | | | | | | 23.Interactive system |
| 124 | | . | . | . | | . | | | . | | | | | 24. Line editor |
| 125 | | | . | | | | | . | . | . | | | | 25. Operator |
| 126 | | . | . | . | . | | . | | . | | . | | | 26 Runtime error |
| 127 | . | . | . | . | . | . | | | . | | . | | | 27 Semantics |
| 128 | . | . | . | . | . | . | | | . | | . | | | 28. Syntax |
| 129 | . | | . | . | . | | | | . | | . | | | 29. Write |
| 130 | . | . | . | . | | . | | . | . | | | | | 30 Quit |
| 131 | | . | | | . | . | | | . | . | . | | | 31 [document_name] |
| 132 | . | . | . | | . | | | | . | | | | | 32 [filename] |
| 133 | | | | . | | | . | | | | | | | 33. [newline] |
| 134 | | | | . | . | . | | | . | | . | | | 34 Ln1 [document] |
| 135 | . | | | | . | | | | . | | | | | 35. A line of text |
| 136 | | . | | | | | . | | | | | | | 36. String1 [document] |
| 137 | | | . | | . | | | | | | | | | 37 Ln2 [document] |
| 138 | . | | . | | | | | | | | . | | | 38. Ln1 [filename] |
| 139 | . | . | | . | . | | | | | | . | | | 39 Ln2 [filename] |
| 140 | . | | . | . | . | | | | | | . | | | 40 First_with_string |
| 141 | | . | . | . | | | . | | | | | | | 41 W |
| 142 | | . | | | | | | | | | | | | 42 Number of characters in a file |
| 143 | . | . | . | . | . | . | . | | O | | . | 9 | | {PRE-GUARD} |
| 144 | . | . | . | | | | | | t | | | | | 1 The environment will be supplied by the UNIX operating system. |
| 145 | | . | . | | | | | | t | | | | | 2 The keyboard will supply the basic typewriter character set. |
| 146 | . | . | . | . | . | . | . | | t | . | | | | 3. The TAB and DELETE keys will provide the same action they do in the |
| 147 | | | | | | | | | | | | | | UNIX environment. |
| 148 | | | | | | . | | | t | | . | | | 4 Only the basic "ed" command outlined in the handout given in class |
| 149 | | | | | | | | | | | | | | will be supported |
| 150 | | | | | | | | | | | | | | 5 The user should know the purpose of the editor |
| 151 | | | | | | | | | t | | | | | 6 Screen handling facilities will be controlled by the UNIX operating system |
| 152 | . | | | | | | | | | | | | | 7 The user is familiar with the UNIX operating system environment and |
| 153 | | | | | | | | | | | | | | the SUN system. |
| 154 | | | | | | | | | | | | | | 8. The user is expected to enter NED commands when in command mode |
| 155 | | | | | | | . | | | | | | | 9. The user is expected to enter a line of text in input mode |
| 170 | | | | | . | | | | M | M | 18 | | | {TASK} |
| 179 | | | | | | | | | v | | | | | 1 Provides functions which use the line of text as the basic unit of |
| 180 | | | | | | | | | | | | | | manipulation |
| 181 | . | | | | | | | | v | | | | | 2 Allows users to create new text files and/or modify already existing ones. |
| 182 | | | | | | | | | v | | | | | 3. Load a file |
| 183 | | | | | | | | | v | | | | | 4 Create a document |
| 184 | | | | | | | | | v | | | | | 5. Insert line(s) |
| 185 | | | | | | | | | v | | | | | 6. Append line(s) |
| 186 | | | | | | | | | v | | | | | 7 Delete line(s) |
| 187 | | | | | | | | | v | | | | | 8. Display line(s) |
| 188 | | | | | | | | | v | | | | | 9 Save a document |
| 189 | | | | | | | | | | v | | | | 10 Improvement of the command prompt |

| | AP | AQ | AR | AS | AT | AU | AV | AW | AX | AY | AZ | CK | CL | CM |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 100 | | | | | | | | | | | v | | | 11. Improvement of the error reporting |
| 101 | | | | | | | | | | | v | | | 12. Improvement of the online help. |
| 102 | | | | | | | | | | | v | | | 13. To show line numbers which indicate relative line positions in the |
| 103 | | | | | | | | | | | | | | document. |
| 104 | | | | | | | | | | | v | | | 14. To show invisible characters such as blank spaces, tab characters, |
| 105 | | | | | | | | | | | | | | and newline characters. |
| 106 | | | | | | | | | | | v | | | 15. To support the editing of multiple documents. |
| 107 | | | | | | | | | | | v | | | 16. To upgrade to a full screen editor |
| 108 | | | | | | | | | | | v | | | 17. To port to another system |
| 109 | | | | | | | | | | | v | | | 18. To provide the user with information concerning the current mode of the ed |
| 110 | | | | | | | | | | G | | 31 | | (POST-GUARD) |
| 111 | | | | | | | | | | t | | | | 1. To provide the user with an easy or y to edit files by using simple one |
| 112 | | | | | | | | | | | | | | character commands and abundant expressions. |
| 113 | | | | | | | | | | t | | | | 2. NED is a line editor which commences and ends on a complete line or a |
| 114 | | | | | | | | | | | | | | a range of complete range |
| 115 | | | | | | | | | | t | | | | 3. NED will work in UNIX environment on networked SUN workstations. |
| 116 | | | | | | | | | | t | | | | 4. NED will only work on one document at a time |
| 117 | | | | | | | | | | t | | | | 5. The maximum size of the buffer that NED can work on is limited by the |
| 118 | | | | | | | | | | | | | | memory available on the workstation |
| 119 | | | | | | | | | | t | | | | 6. The maximum number of files that NED can create is limited by the |
| 120 | | | | | | | | | | | | | | user's disk quota. |
| 121 | | | | | | | | | | t | | | | 7. The editor will not occupy more than 400 Kbytes of main memory |
| 122 | | | | | | | | | | t | | | | 8. The user cannot edit a file for which he does not have write access. |
| 123 | | | | | | | | | | t | | | | 9. A line may contain up to 256 characters (including the NEWLINE |
| 124 | | | | | | | | | | | | | | character) |
| 125 | | | | | | | | | | t | | | | 10. The maximum number of editors per user is limited by the operating |
| 126 | | | | | | | | | | | | | | system |
| 127 | | | | | | | | | | t | | | | 11. NED will use the ASCII standard for character representation. It will |
| 128 | | | | | | | | | | | | | | conform to all relevant standards used by the UNIX operating system |
| 129 | | | | | | | | | | | | | | and SUN systems. |
| 130 | | | | | | | | | | t | | | | 12. It is possible for users to connect remote terminals to the SUNs |
| 131 | | | | | | | | | | | | | | through the PACX network at Concordia University. The user must |
| 132 | | | | | | | | | | | | | | have a modem set up according to the recommendations of the |
| 133 | | | | | | | | | | | | | | Concordia University Computer Center to ensure proper operation of NED |
| 134 | | | | | | | | | | t | | | | 13. Commands are expected to be performed within 30 seconds. |
| 135 | | | | | | | | | | | | | | 14. To append a line into the document after the addressed line |
| 136 | | | | | | | | | | | | | | 15. To change the current line position in the document. |
| 137 | | | | | | | | | | | | | | 16. To toggle between command mode and input mode. |
| 138 | | | | | | | | | | | | | | 17. To initialize the buffer (in memory) for a new document. |
| 139 | | | | | | | | | | | | | | 18. To delete a line or a set of consecutive lines from the document. |
| 140 | | | | | | | | | | | | | | 19. To display selected line(s) from the document onto the screen. |
| 141 | | | | | | | | | | | | | | 20. To insert a line into the document before the addressed line |
| 142 | | | | | | | | | | | | | | 21. To load a file in secondary storage into the buffer |
| 143 | | | | | | | | | | | | | | 22. To exit the editor |
| 144 | | | | | | | | | | | | | | 23. To write the addressed lines from the buffer into a file on secondary |
| 145 | | | | | | | | | | | | | | storage |
| 146 | | | | | | | | | | t | | | | 24. Extensibility developed in such a way as to allow modification |
| 147 | | | | | | | | | | t | | | | 25. Flexibility allows the user to make almost any changes to a document |
| 148 | | | | | | | | | | | | | | as required. |
| 149 | | | | | | | | | | t | | | | 26. Portability easily ported to either a DOS environment or another |
| 150 | | | | | | | | | | | | | | UNIX environment other than SUN. |
| 151 | | | | | | | | | | t | | | | 27. Reliability performs the tasks specified properly and takes into |
| 152 | | | | | | | | | | | | | | consideration abnormal conditions. |
| 153 | | | | | | | | | | t | | | | 28. Safety does not result in writing out the contents of the buffer into |
| 154 | | | | | | | | | | | | | | a file in the wrong order |
| 155 | | | | | | | | | | t | | | | 29. Security integrity of the data is ensured by the operating system |
| 156 | | | | | | | | | | | | | | software |
| 157 | | | | | | | | | | t | | | | 30. Training Time the average user is able to learn and understand all |
| 158 | | | | | | | | | | | | | | the operations in less than one hour |
| 159 | | | | | | | | | | t | | | | 31. Accuracy executes commands as specified in the requirements |
| 160 | | | | | | | | | | | M | M | B | (REMARK) |
| 161 | | | | | | | | | | | | | | 1. If int is not given then the current line position is used by default. |
| 162 | | | | | | | | | | | | | | 2. The document can optionally be given a name upon writing the |
| 163 | | | | | | | | | | | | | | document into a file. |
| 164 | | | | | | | | | | | | | | 3. Input is not specified then the current line is deleted from the document. |
| 165 | | | | | | | | | | | | | | 4. Warns you if you have not saved the changes. |
| 166 | | | | | | | | | | | v | | | 5. Current requirements |
| 167 | | | | | | | | | | | | v | | 6. Future requirements |


{IDENTIFICATION}
1 IntoMap for the NED SRD
{REFERENCE}
1 IEEE Std 830-1984
2. B.R. Bourne, The UNIX System, London, Addison-Wesley, 1983
3. B.W. Kernighan, R. Pike, The UNIX Programming Environment, New Jersey, Prentice-Hall, 1984, pp. 318-328.
4 P.C. Poole, N. Poole, Using UNIX by examples, Sydney, Addison Wesley, 1986.
5. SUN Microsystem using the help function (Sman command)
6. SUN OS Reference Manual, Part number 800-1751 10, revision A, May 8 1986.
{VIEW}
1 Definitions
2. Product Perspective
3. Product Functions
4. Functional Requirements
5. Product Users
6 Bit Cardinality
{USER TYPE}
1 User who would like to create or make changes to a text file
2. Any process running under the UNIX environments.
10 {OPERATION}
1 Append_line
2. Change_current_line
3. Change_mode
4. Create_document
5. Delete_line(s)
6. Display line(s)
7 Insert line
8. Load_document_from_file
9. Quit
10 Write_lines_from_buffer_to_file
11 {EXTERNAL OBJECT}
1 UNIX operating system
2. File
3. Filename
4 Hardware
5. Keyboard
6 Process
7 Screen
8. Input buffer
9. Input keystrokes
10. File storage
11 Other processes
42 {INTERNAL OBJECT}
1 Non Editor (NED)
2. Buffer
3. Document
4. Current line
5. Current line position
6. Text
7 Line
8. String
9. Word
10. Blank
11. Character
12 Newline
13. Access
14 Access time
15. Address
16. Append
17. Command
18. Create
19 Cursor
20 Delete
21 Escape
22 Insert
23.Interactive system
24 Line editor
25. Operator
26 Runtime error
27 Semantics
28. Syntax
29. Write
30 Quit
31 (document_name)
32 (filename)
33. (newline)
34 Ln1 (document)
35. A line of text
36. String1 (document)
37 Ln2 (document)
38. Ln1 (filename)
39 Ln2 (filename)
40 First_with_string
41 W
42. Number of characters in a file
{PRE-GUARD}
1 The environment will be supplied by the UNIX operating system
2 The keyboard will supply the basic typewriter character set.
3. The TAB and DELETE keys will provide the same action they do in the UNIX environment.
4 Only the basic "ed" command outlined in the handout given in class will be described
5. The user should know the purpose of the editor
6 Screen handling facilities will be contained by the UNIX operating system
7 The user is familiar with the UNIX operating system environment and the SUN station

| | | CM |
|---|---|---|
| 114 | | 8. The user is expected to enter NEO commands which in command mode |
| 115 | | 9. The user is expected to enter a line of text in input mode |
| 116 | | 18 (PRE CONDITION) |
| 117 | | 1 The current mode and the current line position is known |
| 118 | | 2 The current mode is in input mode |
| 119 | | 3. The current mode is in command mode. |
| 120 | | 4 A [document name] is given |
| 121 | | 5. The file exists |
| 122 | | 6. Ln1 was the last line of the document before the appending task place. |
| 123 | | 7 First_with_string was the last line of the document before the appending task place. |
| 124 | | 8. Ln2 was the last line in the document. |
| 125 | | 9. Ln1 <= 0. |
| 126 | | 10. Ln1 > (number of lines in the document) |
| 127 | | 11 Ln1 is a numerated value |
| 128 | | 12 String1 exists in the document. |
| 129 | | 13. Ln1 > ln2 |
| 130 | | 14 Ln2 > (number of lines in the document) |
| 131 | | 15. The document is empty |
| 132 | | 16. Sufficient memory is available |
| 133 | | 17 Changes made to the files are saved |
| 134 | | 18. Sufficient space in secondary storage |
| 135 | | 1 (EVENT) |
| 136 | | 1 Upon user request. |
| 137 | | 19 (TASK) |
| 138 | | 1 Provides functions which use the line of text as the basic unit of manipulation. |
| 139 | | 2 Allows users to create new text files and/or modify already existing ones. |
| 140 | | 3. Load a file. |
| 141 | | 4 Create a document |
| 142 | | 5. Insert line(s) |
| 143 | | 6. Append line(s) |
| 144 | | 7 Delete line(s) |
| 145 | | 8. Display line(s) |
| 146 | | 9 Save a document. |
| 147 | | 10. Improvement of the command syntax |
| 148 | | 11 Improvement of the error reporting |
| 149 | | 12 Improvement of the online help. |
| 150 | | 13. To show line numbers which indicate relative line positions in the document. |
| 151 | | 14 To show invisible characters such as blank spaces, tab characters, and newline characters. |
| 152 | | 15. To support the editing of multiple documents. |
| 153 | | 16. To upgrade to a full screen editor |
| 154 | | 17 To port to another system |
| 155 | | 18. To provide the user with information concerning the current mode of the e |
| 156 | | 24 (ACTION) |
| 157 | | 1 The line of text is inserted into the document after the line currently at position ln1 |
| 158 | | 2 The line of text is inserted after the first line in the document containing string1 |
| 159 | | 3. The current line is reset to point to the line specified by the line number ln1 in the document |
| 160 | | 4. The editor is switched to command mode |
| 161 | | 5. The editor is switched to insert mode. |
| 162 | | 6. The buffer is cleared for a new document with specified name [document_name]. |
| 163 | | 7 The buffer is cleared for a new document with no name. |
| 164 | | 8. The set of lines starting from ln1 to ln2 inclusive are deleted from the document. |
| 165 | | 9 Lines ln1 to ln2 inclusive in the document are displayed on the screen. |
| 166 | | 10. Lines between the first line in the document containing string1 and ln2 inclusive are displayed on the screen. |
| 167 | | 11 The line of text is inserted into the document before the line currently at position ln1 |
| 168 | | 12 The line of text is inserted before the first line in the document containing string1 |
| 169 | | 13. The previous contents of the buffer is cleared |
| 170 | | 14 Fetches the entire contents of the file [filename] into the buffer |
| 171 | | 15. The content of the buffer becomes the new document. |
| 172 | | 16 The [document_name] is given the [filename] by default |
| 173 | | 17 Closes the buffer |
| 174 | | 18. Leaves the editor |
| 175 | | 19 The file is created. |
| 176 | | 20. The file is overwritten. |
| 177 | | 21 Starting at ln1, all lines between ln1 and ln2 inclusive are written to secondary storage under the name [document name] |
| 178 | | 22 Starting at ln1, all lines between ln1 and ln2 inclusive are written to the file named [filename]. |
| 179 | | 23. The entire contents of the buffer is written to the file [filename] |
| 180 | | 24 The entire contents of the buffer is written to the file with the current document's name which is [document name] |
| 181 | | 30 (POST CONDITION) |
| 182 | | 1 The document contains an additional line which is the new line of text at position (ln+1) |
| 183 | | 2 The current line position is (ln1+1) |
| 184 | | 3. All lines in the document after and including the line that was at position (ln1+1) before the appending, are moved down by one line position |
| 185 | | 4. The document contains the new line of text at position first_with_string + 1 |
| 186 | | 5. The current line position is first_with_string + 1 |
| 187 | | 6. All lines in the document after and including the line that was at position first_with_string before the append/insertion, are moved down by one line position in the document. |
| 188 | | 7 The current line is ln1 |
| 189 | | 8 The editor is in command mode |
| 190 | | 9 The editor is in input mode |
| 191 | | 10. An empty, null buffer is ready to read in the new document |



CM

11 The current line position is 0.
12 The set of lines between ln1 and ln2 inclusive are removed from the document.
13. The line that was at position (ln2+1) becomes the current line.
14. The current line position becomes (ln1 1)
15. All lines after the line that was at position ln2 are moved up by n positions.
16. The specified lines are displayed on the screen
17 The current line position will be ln2.
18. The document contains an additional line which is the new line of text at position ln1
19 All lines in the document after and including the line that was at position ln1 before the insertion, are moved down by one line
20. The document contains the new line of text at position line_with_string.
21 The current line position is line_with_string.
22 The contents of the file is loaded into the buffer
23. A numeric value stating the number of characters in the file is displayed on the screen.
24 The current line pointer is the last line in the document.
25. Returns to shell prompt.
26 The file with name [document_name] contains only the lines between ln1 and ln2 inclusive, and is stored in secondary storage.
27 The file named [filename] contains only the lines between ln1 and ln2 inclusive, and is stored in secondary storage.
28. The file named [filename] contains an exact copy of the current buffer contents, and is stored in secondary storage
29 The file with name [document_name] contains an exact copy of the current buffer contents, and is stored in secondary storage
30. The current line position will remain at the same position as before the invocation.

31 ( POST-GUARD )
1. To provide the user with an easy way to edit files by using simple one character commands and consistent expressions.
2. NED is a line editor which commands will work on a complete line or a a range of complete range
3. NED will work in UNIX environment on networked SUN workstations.
4. NED will only work on one document at a time
5. The maximum size of the buffer that NED can work on is limited by the memory available on the workstation
6 The maximum number of files that NED can create is limited by the user's disc quota.
7 The editor will not occupy more than 400 Kbytes of main memory
8. The user cannot edit a file for which he does not have write access.
9. A line may contain up to 256 characters (including the NEWLINE character).
10. The maximum number of editors per user is limited by the operating system.
11 NED will use the ASCII standard for character representation. It will conform to all relevant standards used by the UNIX operating system and SUN systems.
12 It is possible for users to connect remote terminals to the SUNs through the PACX network at Concordia University The user must have a modem set up according to the recommendations of the Concordia University Computer Center to ensure proper operation of NED
13. Commands are expected to be performed within 30 seconds.
14. To append a line into the document after the addressed line
15. To change the current line position in the document.
16. To toggle between command mode and input mode.
17 To initialize the buffer (in memory) for a new document.
18. To delete a line or a set of consecutive lines from the document.
19 To display selected line(s) from the document onto the screen.
20 To insert a line into the document before the addressed line
21 To load a file in secondary storage into the buffer
22. To exit the editor
23. To write the addressed lines from the buffer into a file on secondary storage.
24. Extendibility developed in such a way as to allow modification
25. Flexibility allows the user to make almost any changes to a document as required.
26. Portability - easily ported to either a DOS environment or another UNIX environment other than SUN.
27. Reliability performs the tasks specified properly and takes into consideration abnormal conditions.
28. Safety - does not result in writing out the contents of the buffer into a file in the wrong order
29. Security integrity of the code is ensured by the operating system software.
30. Training Time - the average user is able to learn and understand all the operations in less than one hour
31. Accuracy - executes commands as specified in the requirements.

6 ( REMARK )
1 If ln1 is not given then the current line position is used by default.
2 The document can optionally be given a name upon writing the document into a file.
3. Input is not specified then the current line is deleted from the document.
4. Warns you if you have not saved the changes.
5. Current requirements
6 Future requirements