

AD-A172 486

DESIGN OF A DATA DICTIONARY EDITOR IN A DISTRIBUTED
SOFTWARE DEVELOPMENT ENVIRONMENT(U) AIR FORCE INST OF
TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI..

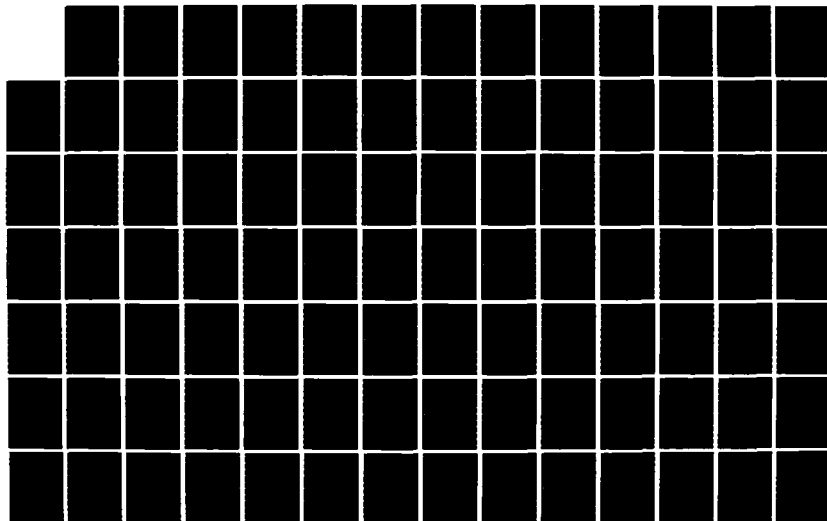
1/2

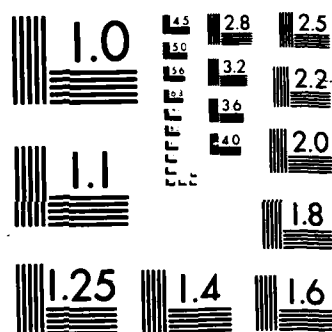
UNCLASSIFIED

J W FOLEY JUN 86 AFIT/GCS/ENG/86J-5

F/G 9/2

NL





AD-A172 406

MM: FILE COPY



DESIGN OF A DATA DICTIONARY
EDITOR IN A DISTRIBUTED SOFTWARE
DEVELOPMENT ENVIRONMENT

THESIS

Jeffrey W. Foley, B.S.
Captain, USA

AFIT/GCS/ENG/86J-5

This document has been approved
for publication and its
distribution is unlimited.

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

TIC
OCT 2 1986

A

86 10 01 203

(1)

AFIT/GCS/ENG/86J-5

DESIGN OF A DATA DICTIONARY
EDITOR IN A DISTRIBUTED SOFTWARE
DEVELOPMENT ENVIRONMENT

THESIS

Jeffrey W. Foley, B.S.
Captain, USA

AFIT/GCS/ENG/86J-5

101 2 1986

Approved for public release; distribution unlimited

AFIT/GCS/ENG/86J-5

DESIGN OF A DATA DICTIONARY EDITOR IN A
DISTRIBUTED SOFTWARE DEVELOPMENT ENVIRONMENT

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University

In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Systems

Jeffrey W. Foley, B.S.

Captain, USA

June 1986



A-1

Approved for public release; distribution unlimited

Preface

Data dictionaries enjoy considerable attention in software documentation requirements in the AFIT research environment. Previous research efforts have provided some insight into the requirements of data dictionary systems but have focused on single computer environments. The purpose of this study was to expand the data dictionary system to the distributed development environment. The key to this expansion was the design and development of a special data dictionary editor that creates and updates definitions at a workstation. Definitions are then transferred to and from the central database as needed for project development.

In preparing this thesis I have received considerable help and support from others. First, I am very grateful to my thesis advisor Dr. Thomas C. Hartrum for his guidance and support throughout the entire effort. I also wish to thank my committee members, Dr. Gary B. Lamont and MAJ Duard S. Woffinden, for their valued assistance. Two classmates were particularly helpful to me: CAPT Thomas Zuzack in educating me on the workings of the Z-100 microcomputer, and CAPT Charles Hamberger for his enhancements of the previous database interface software. Finally, I wish to thank my dear wife Beth for her continuous, loving support throughout our entire graduate school experience.

Jeffrey W. Foley

Table of Contents

	Page
Preface	ii
List of Figures	vi
Abstract	vii
I. Introduction	1
Background	1
Problem Statement	2
Scope	3
Assumptions	4
General Approach	4
Sequence of Presentation	5
II. Review of the Literature	6
Data Dictionary Systems	7
Human Interface Issues	10
Distributed Development Environments	15
Problems Associated With Distributed Interaction	16
Framework for a Distributed Database Interface	21
Summary	25
III. Data Dictionary System Requirements	26
Overall System Analysis	26
Data Dictionary Editor Requirements	40
Central Computer Database Interface	42
Database Interface Facilities	42
Security of the Database	43
Summary	44

	Page
IV. Data Dictionary System Design	45
Data Dictionary Editor Design	45
User-Machine Interface	46
Screen Display	52
Data Structures	55
Windowing Scheme	59
Data File Input and Output	63
Editor Commands	66
Keyboard Layout	67
Error Handling	67
Communications Interface	69
Database Interface Design	70
Summary	72
V. Implementation and Test	73
Portability of the Code	73
Generic Editor Implementation Issues	75
Operation of the Tool	76
Testing	78
Summary	81
VI. Evaluation of the Data Dictionary Editor	82
Measuring User Satisfaction	82
Evaluation of the Data Dictionary Editor	87
Conclusions	91
VII. Conclusion and Recommendations	93
Conclusions	93
Recommendations for Further Study	96
Appendix A: Evaluation of an Automated/Interactive Software Engineering Tool to Generate Data Dictionaries	A-1
Appendix B: Data Dictionary Editor Users' Guide	B-1
Appendix C: Evaluation Questionnaire	C-1
Appendix D: Evaluation Handout	D-1
Appendix E: Editor SADT Diagrams	E-1
Appendix F: Design Structure Charts for the Editor.	F-1

	Page
Appendix G: Summary Paper	G-1
Bibliography	BIB-1
Vita	.VITA-1

The following additional thesis volumes are maintained at the Air Force Institute of Technology, Department of Electrical and Computer Engineering:

Volume II: Workstation Editor Code and Data Dictionary

Volume III: Database Interface Code

List of Figures

Figure	Page
1. Framework for a Distributed Database Interface	22
2. The Distributed Development Environment	27
3. Data Dictionary Format for Structure Chart - Process	30
4. Data Dictionary Format for Structure Chart - Parameter	31
5. Typical Structure Chart	32
6. Example of Parameter Passing	33
7. Third Normal Form Relations for a Design Structure Chart Process	36
8. Third Normal Form Relations for a Design Structure Chart Parameter	37
9. Opening Menu on the Editor	47
10. Menu #2 for Create	48
11. Menu #3 for Create	49
12. Sample Screen Display	53
13. Linked Structure Used in the Editor	57
14. Screen Editor "Windowing Scheme"	60
15. Format of Flat File	65
16. Sample Survey Question	84
17. Score Boundaries for Normalized User Satisfaction	86
18. Normalized Values for Overall User Satisfaction	88
19. Mean Scores for Each Factor	89

Abstract

The project involved the design and implementation of a data dictionary system in a distributed development environment. The distributed environment consists of a central computer that hosts a database management system, a conglomerate of workstations, and the communications links between the workstations and central computer.

The emphasis of the research was placed on the design of a user-friendly data dictionary editor that was implemented on a prototype workstation. Data dictionary definitions are created and updated at the workstation and transferred between the workstation and central computer database.

Background information is provided on data dictionary systems, aspects of human-computer interfaces, and distributed environment interface issues. The design and development of the special editor and the database interface software are described in detail.

Evaluation of the special editor was performed by a subset of the target user group. This evaluation was based on a tool designed to measure user satisfaction. The tool is described and the results of the evaluation are provided.

DESIGN OF A DATA DICTIONARY
EDITOR IN A DISTRIBUTED SOFTWARE
DEVELOPMENT ENVIRONMENT

I. Introduction

Background

The Department of Electrical and Computer Engineering at the Air Force Institute of Technology (AFIT) sponsors a large amount of research in the area of software development. In conjunction with this research, the department has established documentation standards that include data dictionaries to support the requirements, design, and coding phases of the software lifecycle (37). Originally these data dictionaries were created and managed by hand. As the dictionaries grew in size, the effective control and management of their contents became increasingly difficult (16). Because the characteristics of data dictionary systems are so similar to those of database management systems (23:10), the task to automate the data dictionary system was established.

Several efforts at AFIT led to the development of a limited data dictionary system implemented on a Vax-11/780 computer under the Unix operating system, and using the Berkeley version of the relational database management system INGRES. Thomas (39) consolidated these efforts to produce a limited working system, which was later extended

and improved by Hamberger (16).

There are several deficiencies with the current implementation of the software, the most critical being the lengthy response time experienced while using the system (a complete review, by this author, of the current implementation can be found in Appendix A).

Initial performance evaluations of the system indicated that the lengthy response times could be attributed to three factors (16, Appendix A). The first factor, believed to be the biggest contributor, was the time required to edit data dictionary definitions. The time required to interact with the database management system was the second factor. The third factor was the time attributed to the system load, or number of users on the computer.

Because data dictionary systems can be so valuable in documenting software development projects (22) there was a requirement for such a system to support these research efforts.

Problem

Requirements for a complete data dictionary system for the software development lifecycle needed to be reviewed and further developed at AFIT. The existing documentation requirements and the limited implementation of Thomas' data dictionary generator provided a starting point. The overall objective of this thesis research was to expand and enhance the current data dictionary system to support a distributed

development environment. Inherent in this objective was the evaluation of the current system and the goal to design and implement a user-friendly interface on a microcomputer workstation that would generate data dictionary definitions that could be transferred to and from a central database. Subsequent interface software was also required at other interface locations in this distributed development environment.

Scope

This research effort was limited to the development of a data dictionary system for use at AFIT in support of software development sponsored by the Department of Electrical and Computer Engineering. It was implemented on the Vax-11/780 computer, using the Unix operating system, with the prototype microcomputer workstation being the Zenith Z-100 using the MS-DOS operating system.

Interface software was designed and implemented on the microcomputer workstation for the purpose of editing data dictionary definitions, and on the main computer for interfacing with the database. Current available communication software packages were used for the transfer of text files between the workstation and main computer. Efforts were made to keep the code as generic as possible, especially at the microcomputer level, to facilitate the portability of the software to other workstations.

Assumptions

Thorough discussions on the software lifecycle and the graphical representations that support various phases of the lifecycle were not included in this thesis. It is assumed that the reader has a basic knowledge of these subjects, or has the resources available from which to obtain the knowledge.

The relational database management system INGRES is assumed to operate and perform according to specifications and would provide all the necessary requirements for a database.

The current edition of the standards and guidelines published by the Electrical and Computer Engineering department (37) are assumed to represent a valid subset of documentation requirements for software development.

General Approach

Initial research focused on data dictionary systems: their purpose, their information content, and their design considerations. Researching the issues of human-computer interfaces and distributed development environments followed. Together, the results of these investigations provided a foundation from which to analyze and evaluate the current implementation of the data dictionary system employed in the AFIT software development research environment. Based on the results of this evaluation, the design of upgrades to the system to operate in a distributed

environment were developed. Upon completion of the implementation effort, the new portions of the system were presented to a subset of the target user group for evaluation.

Sequence of Presentation

This thesis consists of six sections. A review of current literature on data dictionary systems, human-computer interface issues, and distributed development environments is presented in Chapter II. The data dictionary system requirements are provided in Chapter III, with the design of the data dictionary system following in Chapter IV. Chapter V discusses the implementation and testing of the software. The results of the evaluation performed on the software are presented in Chapter VI. Conclusions and recommendations for further study are discussed in the final chapter.

II. Review of the Literature

A firm understanding of a data dictionary's purpose, its design methodology and its implementation aspects peculiar to the target environment is required. To this end a search of the literature was conducted, the results of which are described in the first section of this chapter.

User-friendly interfaces was also a key subject that pertained to this thesis, particularly the proposed editor for the workstation. Consequently, it was imperative to search the literature for an understanding of how user-friendly is defined and how user-friendly systems are designed for applications similar to this thesis. The second section of this chapter highlights the results of this search.

Distributed development environments provide a more diverse and flexible atmosphere for project development but their design and implementation requires considerable more time and resources than single computer environments. The issues associated with distributed environments required identification to gain an understanding and appreciation for the complexities involved, and to determine the reasonable objectives that could be achieved within the scope of this thesis. The results of this investigation are discussed in the third section of this chapter.

Data Dictionary Systems

Data dictionary systems (DDS) have experienced a growing degree of usage throughout the dataprocessing field. Data is a corporate resource, similar to personnel, money, and raw materials, that, until the late 1970's, was never managed like a corporate resource (23:2). Because of its great tendencies to be erroneous, unavailable, and out-of-date, this data resource had to be carefully managed and controlled. Data dictionary systems provide a facility that supports this objective (22).

The importance of DDS becomes apparent when it is realized how their information is used. Potential users of the information system must know the names and meanings of the information to exercise the system effectively. Programmers and system administrators must know the characteristics of the data to perform their duties effectively. System designers and modifiers must know the naming conventions employed and the structure of the data to perform their duties effectively (22). The data dictionary is, then, the central source for this information.

It is an attempt to capture all definitions of data within an enterprise for the purpose of controlling how data is used and created and to improve the documentation of the total collection of data on which an enterprise depends (22:1-1).

Data dictionary systems can be considered information systems in their own right (22:1-2). An information system essentially consists of four elements:

1. A database containing the information required by the information system.
2. A processing system which interacts with the database and satisfies the functional requirements of the system.
3. The users of the system, together with all the procedures explicit and implicit, for the system.
4. The hardware/software environment required to perform the above (22:1-3).

In the AFIT environment, the database is stored using the database management system INGRES, while the processing system consists of the previous software and the software generated as a result of this thesis. The users are the students at AFIT pursuing a masters degree in computer engineering or computer systems. The hardware is provided by AFIT and the software is provided as just stated.

The effectiveness of a DDS depends on the degree of commitment by the organization. If implemented and managed successfully, it will ensure correctness and consistency of the data and will provide an invaluable asset in managing changes within the current system. Lefkovitz outlined a number of benefits derived from the use of a DDS:

1. DDS reduces unintentional redundancies in data. Redundancies waste storage space and create data update problems. It can also document intentional redundancies where necessary.
2. Systems development costs and time can be reduced by ensuring consistency among the data, and eliminating potential misunderstandings of the data definitions.
3. System maintainability is enhanced through complete documentation of the system. Communication between the users and maintenance programmers is improved.

4. DDS can reduce the impact of changes to the system by aiding in identifying all locations where changes or modifications are affected.

5. DDS can establish and enforce standards of definitions and usages of data.

6. The creation of a database provides an efficient location to store information and make it available to all necessary personnel. This aids in communicating concise information and reduces the potential of semantic misunderstandings.

7. A properly implemented and well-managed DDS provides a system trusted by all users (22:1-8 thru 1-11).

A DDS provides a useful tool in supporting the documentation of a large software system. "Documentation on large software systems is usually poor, outdated, incomplete, inaccurate...sometimes nonexistent" (23: 49). This is generally the case since documentation is traditionally the last task to be completed. A data dictionary generator can aid immensely in providing a timely, complete, and accurate documentation package for any system.

Professor Edgar Sibley, of the University of Maryland, summarizes his strong beliefs on the importance of data dictionary system:

1. Anyone who designs and implements an information system, whether it is data based or not, should use a dictionary system...Human memory is not enough.

2. It is essentially impossible to exert management control over the design, implementation, and use of large information systems without some automated means. Dictionary systems are the best available commercial aid to these important phases of the system lifecycle (22).

For a data dictionary system to be useful it must be easy to use. Information must be easy to input and extract from the system in a usable form. This requires an effectively designed interface between the user and the computer, more commonly known as the human-computer interface.

Human-Computer Interfaces

Human-computer interfaces have been the subject of considerable research. Twenty-five years ago computer systems were designed for the technician. Today, such is not the case. According to James Martin:

...man must become the prime focus of system design. The computer is there to serve him, to obtain information for him, and to help him do his job. The ease with which he communicates with it will determine the extent to which he uses it (27:3).

How effectively an individual can use a computer is based on how that individual was trained to use it and how effectively the computer was designed to work with the individual. These characteristics must be balanced, in a sense, to provide a good working environment. For example, an individual with little or no experience with computers must depend on a well designed interface for the system to provide a useful service. This is further exemplified by Department of the Army sponsored research (10:2-2) which resulted in the following observations:

1. If people do not like the system it will not work.
2. Personnel with negative attitudes commit more

errors and take longer to learn a system than those with neutral or positive attitudes.

The concept of "user-friendly" denotes a variety of meanings depending on the source. The American Heritage Dictionary defines friendly as "...not antagonistic" (38:527). Crenshaw states that friendly environments reinforce the fundamental human needs psychologists long ago identified. These include:

1. Need to have our expectations met.
2. Need for clear information.
3. Need to succeed.
4. Need for a right to fail (without extreme repercussion).
5. Need for individuality (6:530).

The "ease-of-use" concept is often discussed with user-friendliness and provides a more measurable aspect of interfaces (13, 24). "Ease-of-use refers to the physical and mental workload necessary to first learn and then use an interface" (24:246). Lindquist discusses usability of an interface in terms of efficiency where the number of keystrokes, commands, and time can be used for evaluation. Ease-of-use has also been defined as the time required to reach a designated level of efficiency, errors per unit time (or number of operations), and the user's attitude toward the system (10:2-3).

User-friendly systems have been characterized by an almost infinite number of other qualities, such as providing

graceful recovery from errors, displaying colorful and stimulating screens of information, reducing user-defensiveness, anticipating the user's perceptions, keeping the user motivated, and providing on-line help facilities (10, 18, 27). Despite the elusive definitions that these terms have, they still provide helpful guidance to interface designers. Designing an interface that meets these objectives is not easy, and, in fact, some claim has yet to be done (6, 13). "Friendliness must be actively sought, planned for, and designed in. Even when it is actively sought there is no guarantee that it will be found" (6:527).

In his master's thesis Interactive Environment for a Computer-Aided Design System (44) CPT D. S. Woffinden analyzed the criteria, by a variety of authors, for a user-friendly interface. He found that the majority of design guidelines were similar in many respects, yet each set of criteria alone fell short of providing complete guidance for the design of an entire interactive system for computer aided design applications (44:17-19). Designing user-friendly interface software is not an exact science as speed and efficiency tradeoffs abound with any design consideration. By understanding Woffinden's principles the designer will be better prepared to produce the best product for the application. An applicable subset of Woffinden's design principles (those that impact upon data dictionary

systems) are identified below and include brief descriptions of each where appropriate.

1. Determine the Purpose of the System. The purpose of the system must be known to fully understand what the system is expected to do.

2. Know the User. The target user group may have a dramatic impact on how the system is designed. Their level of knowledge on computers and their desires of the system will dictate many aspects of the design.

3. Identify Resources. The resources (primarily hardware and software) available for the research must be known to identify the capabilities and limitations of the targeted environment.

4. Human Factors. Human factors are divided into two categories: physical and psychological. Those aspects of the physical environment, such as the working conditions, that can and cannot be changed must be known so that the interface could be designed accordingly. Psychological factors also will affect the design. Five key areas in the psychological category are:

- a. Keep the user motivated -- do not frustrate or bore him.
- b. Break the lengthy input process into parts to permit the user to achieve "psychological closure." This provides positive feedback to the user through a feeling of accomplishment and success (27:324-6).
- c. Minimize the memorization required by the user.
- d. Provide visually pleasing displays on the screen. This includes minimizing the scrolling and other

distracting movements of text, the highlighting of instructions to the user, and the making effective use of margins and white space.

e. Keep response time to a minimum. Display status messages to keep the user constantly informed of what is happening inside the machine.

5. Design for Evolution. The system must be designed with the ability to accommodate future changes.

6. Optimize Training and Accommodate Levels of Experience. All users must be able to perform meaningful work without assistance. Whether users' manuals or on-line help facilities are provided is based on the circumstances surrounding the system.

7. Use Menus vs Text Entry. Use menus rather than requiring the user to enter text information when a limited set of options exist for input. This method is faster, more efficient, eliminates potential input errors, and reduces memorization.

8. Be Consistent. Be consistent in the design of the interface. Use the same procedures for entering input, the same formats for displaying information on the screen, and the same commands for exiting, saving, and traversing through the various levels of the structure.

9. Anticipate Errors. Anticipate and provide a means for correcting errors where possible. Embed the syntax of the language in the system, transparent to the user. Errors in the text should be correctable on the spot. Users should be given the opportunity to review their input

and make corrections prior to processing the information. The system design must protect the user from both himself and the system. When errors occur, succinct error messages are essential.

There are basically three categories of human-computer dialogue (1:547-48). The first category is where the user requests information from the machine based on the data stored in the machine. The primary communication of information is from the computer to the user. The query language used with data bases is a common example.

The second category is where the user selects an option from a fixed set provided by the computer. Communication of information flows equally in both directions between the user and computer. Menu structures are examples of this category.

The third category is where the user responds to requests by the computer. Information flows primarily from the user to the computer. Form-filling is an example of this. Each type of dialogue provides a method of communication suited for specific situations, but not all situations. Most interfaces will contain a combination of these types of dialogues.

Distributed Development Environments

This section focuses on the significant problems and issues that must be addressed in the design of an interface in a distributed development environment. The term

interface, in this context, refers to more than the user-computer interface. It also includes the hardware and software interfaces between groups of workstations (or nodes), between the workstations and the central system, and between the workstation terminal equipment and the communications equipment.

An overview of the major problem areas and design considerations of the system interfaces is necessary to gain an understanding of the complexity of the design problem. Understanding the key issues is essential for progress even though they may not all be solvable (32:385).

Problems Associated with Distributed Interaction.

Database Semantics. Knowing something about how the information is stored inside the database can be beneficial to the user. Users often have a need to find certain information in the database but do not know exactly where to look (42:382). Some method of data directory management will usually assist in solving this problem. A facility to "browse through" the database may be available, but it may also be a very time consuming process if the database is large. Data dictionaries provide some of the insight into how information is stored within the database and should be available to all users. However, it is difficult for data dictionaries to provide information on the relationships between data. These relationships, and other semantic knowledge of the database, could help the

users overcome this common problem.

Alternatives in Presentation of Information. The requirement exists to present different types of information in different forms (5, 8, 36, 42). Text is not the only available type of information stored in databases. Drawings, photographs, sound, and other graphic representations have a place in the workstation environment and must be supported as the application demands. The alternatives available to present information depend on the hardware and software capabilities of the workstation, how the information is stored in the database, and the costs associated with the transmission of the information.

One long term objective in the AFIT data dictionary system environment is to be able to present data dictionary information in text format or in graphic form, such as structure charts or SADT diagrams. Ideally, the information would be stored in the database in the most efficient manner, extracted and placed in the most efficient structure for transmission purposes, and transmitted to the workstation by the most efficient and economical means available. The interface at the workstation should be able to present that information in whatever form the user requests.

Cost. Cost is a difficult item to control in the distributed database environment. Cost is commonly measured in actual transmission costs (dollars), time, or space, or a

combination of the three (42:382). How and where the information is stored in the database has a major impact on the cost associated with the system. The capabilities and limitations of the hardware within the system can affect cost dramatically. The software used to access the data (including the storage structures used, how the query processing is distributed) can affect the cost dramatically as well. The determination of transmission paths is also an important variable in the overall cost equation. Designing a system to analyze and evaluate these criteria is a difficult and very complex task.

How information is temporarily stored as it is retrieved from the database is an important design consideration. The format of the file, such as binary or ASCII, may impact the transmission capability. What file structure is most efficient in terms of space and transmission time is also important. The actual size of the file will affect transmission.

Some applications for workstations require all information pertaining to a particular project be immediately available at the workstation. This enables the designer to work on the project as an entire entity, rather than piecemeal. In view of this environment, it would be desirable to transmit all definitions of data items (whether required for modifications or not) pertaining to a particular project to the workstation. The most economical

means to transfer files over the transmission path should be employed. With respect to project development, exactly how much of the project data is required at the workstation should be determined by the user.

The decision process in determining optimum storage structures and the means of transmission is logically based on three criteria. These criteria include the database itself, the hardware capabilities and limitations of the equipment involved, and the existing software. Each of these items contribute greatly to the complexity of the cost problem.

Generally, the system itself would analyze the overall cost of alternatives and make a decision. If, however, the system had the option to query the user for assistance in making such decisions, then the user must be qualified to provide such guidance. This introduces an additional requirement that users be qualified to input guidance to the system.

Supporting Software Tools. A variety of tools are used in conjunction with database management systems (42:382). Report generation, statistic collection, or any other tailored query are all possible "tools" that could be used to obtain or manipulate the data in the database.

There are basically three capabilities that must be provided to adequately support any desired tool (42). First, the interface must provide the user access to the

tools that are available for use. Second, the tool must support the specific data desired by the user and must control the execution of the other supporting tools, if required, to properly access the data. In a distributed database environment, different tools will reside at different nodes within the network. These tools must communicate with each other when circumstances arise that require combinations of tools to perform database actions. Third, the system must know the semantics of the database and tools to provide correct facilities for presenting the data in its proper form.

Because the tools and the data can be stored at a variety of locations, the distributed database environment adds to the complexity of the interface design. Decisions have to be made by the system as to the best method of procuring the required information for the user. Where system decisions are not possible or practical, the alternatives must be provided to the user for guidance.

Particular tools that are required in the DDS are those that check for consistency within the database. Not all the data pertaining to a particular project is provided by a single user. Project definitions will obtain data indirectly from other sources. The database itself must be searched to provide some of this critical data. There is a requirement for the instantiation of the tool that provides this service after updates have been made to the database.

Where these tools are stored and how they are executed is of critical importance.

In environments where all information pertaining to a particular project is at the workstation, some facility should be available to process all modifications (direct and indirect) to the data, at the workstation, prior to transferring the information back to the database. If data definitions in a data dictionary system are deleted as the result of the update process, the database should recognize the deletions upon the updating of the database from the temporary file.

Framework for a Distributed Database Interface.

Although the scope of this thesis limits the investigation to only a workstation interfacing with a single central data base, there are clear applications for distributed database environments. Wilson proposed a conceptual framework for the distributed database interface (42:382-3). Five categories of knowledge provide the foundation for this framework: database schema knowledge, database semantic knowledge, generic and individual user model knowledge, communication knowledge, and equipment knowledge. Each of these knowledge bases are important to the designer. Each provides capabilities and limitations that will impact the design of the system and must be considered. Although endowing an interface with an abundance of knowledge from all five categories is not possible yet, providing some from all

categories and more from those particularly relevant to the interface, is a genuine goal. Figure 1 shows a graphical representation of this framework, and an explanation of each body of knowledge follows.

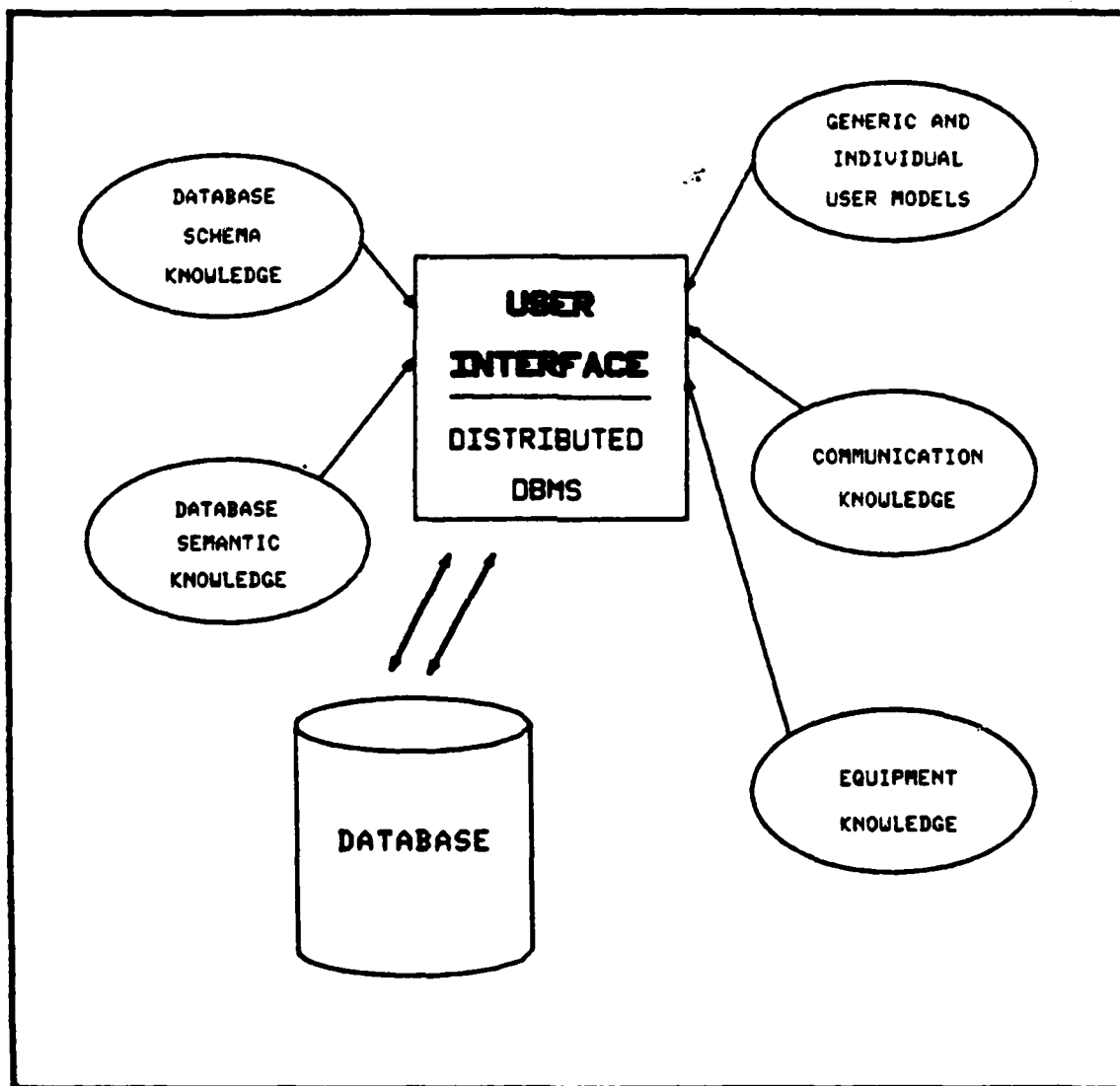


Figure 1. Framework for a Distributed Database Interface (42:382).

Database Schema Knowledge. The schema knowledge is the logical structure of the entire database (42:382). The system must know how and where the data is stored within the distributed network of nodes. The version of the database stored at each node is important information. The type database management systems employed at the various nodes is important. The query languages supporting the local database must be known. The methods of concurrent update operations to data must be known. Database schema knowledge is used in the formulation of queries to the database. Where users directly formulate their own queries this knowledge is used as an error checking mechanism. This knowledge can also be used for translating natural language representations of queries into the required database management system query language.

Semantic Database Knowledge. This knowledge is similar to data dictionary knowledge, but is more extensive in scope. It includes knowledge about how the information is stored in the database, the limitation on information that is stored, and, most importantly, the relationships among the different fields of data in the database. Increasing the intelligence of the system through semantic modeling of the database design and definition is a much needed capability to improve database usability (3:1).

Generic and Specific User Model Knowledge. In any distributed environment there will be different categories of users (42, 44). The general categories of users will be: 1) infrequent or novice users who having sporadic and limited direct access requirements of the data (for example, managers, decision makers); 2) casual users who use the database frequently, but only in limited ways; 3) experienced users who are the regular and frequent users of the database; and 4) programmer users who manage and extend the database (known as Data Administrators, DA, or Database Administrators, DBA). Capturing the knowledge of the target user group will lead to more user-friendly environments (5:18). Generic models include those items that all interfaces should provide, such as preferred modes of interaction for classes of users. Specific user models are tailored to the needs and requirements unique to certain users.

Communication Knowledge. Communication knowledge contains information on the costs necessary to acquire data from other nodes within the network (42). A primary objective of most interfaces is to keep the structure of the distributed system transparent to the user, but at times it is necessary to provide this information directly to the user. Alternatives available to transfer data between nodes, with their associated costs, is known and should be provided to the user where required. Costs associated with

transforming data into various forms should also be known and made available to the user as necessary.

Equipment Knowledge. The equipment available at any site will affect the design of the interface. The functional capabilities of the devices will dictate, among other things, the alternatives available for information representation. The user interface must know whether graphics or color, for example, can be supported at a particular node. The interface should be designed to take advantage of all the capabilities currently available at any node, with the ability to accommodate future upgrades.

Summary

The purpose of this literature search was to uncover information found in the current literature that pertains to data dictionary systems, human-computer interfaces, and distributed development environments. Establishing a foundation of understanding the fundamental concepts in each these areas was necessary before the requirements for a data dictionary system in the AFIT environment could be established. The information in this chapter provided that foundation.

III. Data Dictionary System Requirements

The contents of this chapter present the requirements of the data dictionary system within the distributed software development environment. The chapter is divided into three sections. The first section devoted to the analysis of the overall system, the second section devoted to the requirements of the data dictionary editor, and the final section devoted to the requirements of the interface to the database.

Overall System Analysis

The distributed software development environment at AFIT is shown in Figure 2. It consists of a contingent of workstations, a central computer, and the communications links between the computers. The workstations include a variety of types from personal computers at students' homes to the sophisticated Sun workstations in the computer laboratories. The central computer is a Vax-11/780 that can operate under the Unix or VMS operating systems. The database management system INGRES is available with both operating systems (different versions for each) and is to be used for the storage of data dictionary data. The workstation links to the central computer are through a Gandalf switch. This Gandalf switch also provides a dialup capability for the home computers to communicate with the central computer.

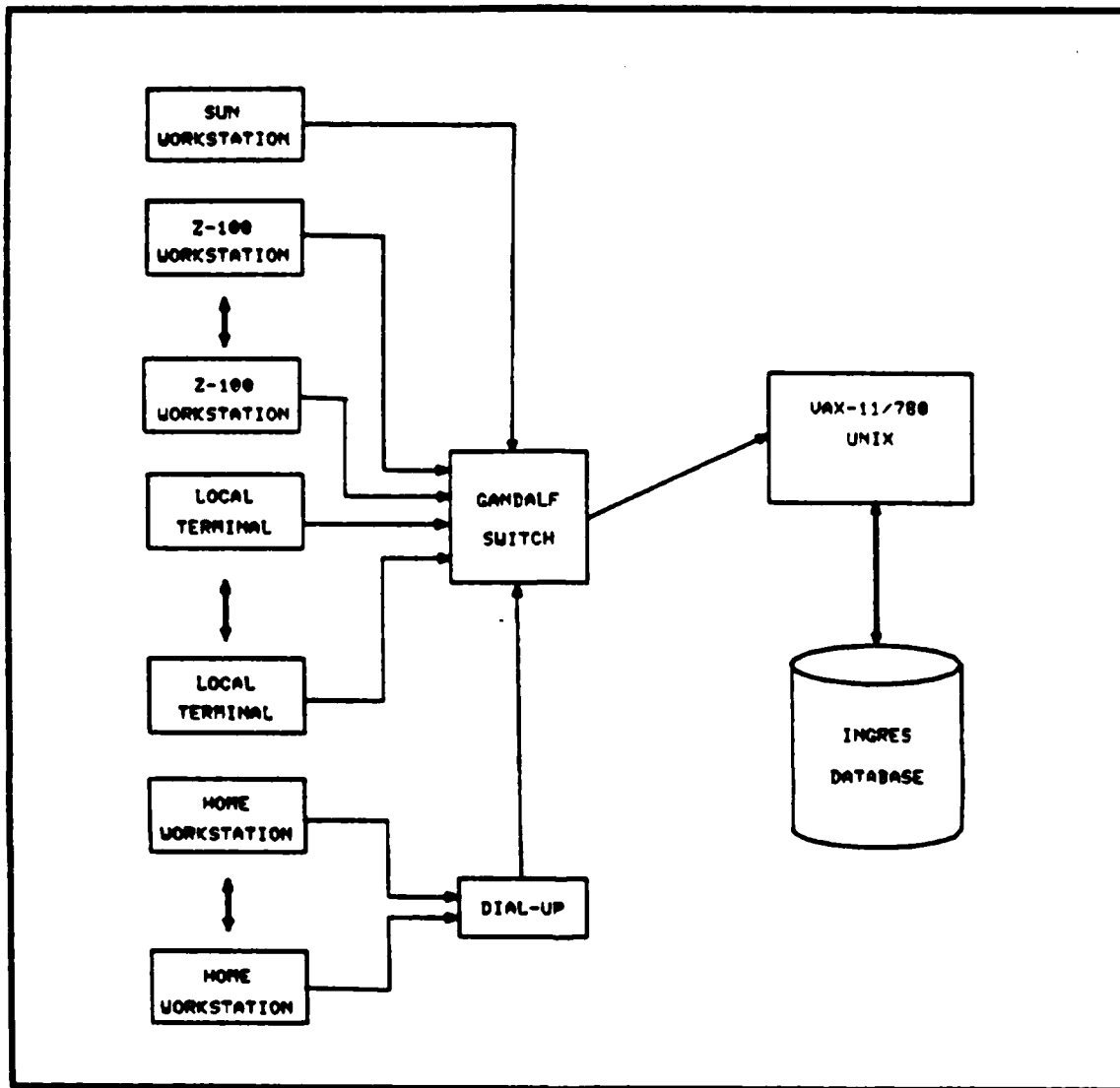


Figure 2. Distributed Development Environment.

The objective of a distributed development environment is to provide the facilities for users to perform the majority of their research at individual workstations in the

comfort and convenience of their local work areas or home. Using workstations for the majority of information processing substantially reduces the student's reliance on the availability of the central computer central processor unit (cpu) and consequently provides favorable conditions for improving efficiency of student research.

The target user group consists of graduate students at AFIT pursuing a curriculum in electrical or computer engineering. All users will have some experience with computers and will be familiar with the requirements document for data dictionary information. Users will be familiar with the Unix Vax-11/780 (SSC) computer and will most likely have some experience with the use of microcomputers.

Software development documentation standards at AFIT follow the software lifecycle phases consisting of the requirements analysis phase, the preliminary and detailed design phases, and the implementation phase. Graphical representations often used to support the documentation requirements include Structured Analysis Design Technique (SADT) diagrams (SADT is a registered trademark of SofTec) for the requirements analysis phase, and Structure Charts (SC) and Data Flow Diagrams for the design phase. Data dictionaries often accompany these representations in addition to accompanying the actual code (37).

The distributed environment is required to support the

capability to generate and update data dictionary definitions at the workstation and transmit these definitions between the workstation and the central database. New definitions created at the workstation are transmitted to the central computer for storage in the database. Existing definitions requiring updates are retrieved from the database and transmitted to the workstation for updating. Inherent in this requirement is the need for a user-friendly designed editor on the workstation, a communications interface that provides the transmission capability between computers, and the corresponding interface on the central computer that can load and retrieve definitions to and from the database.

The contents of data dictionary definitions for the phases of the lifecycle stated above are detailed in the Department of Electrical and Computer Engineering Software Development Documentation Guidelines and Standards (37). Sample definitions are shown in Figures 3 and 4 for a structure chart process and parameter in the design phase. This represents the data that should be represented in the database and is often referred to as operational data (7:7).


```

PRNAME: Process Message
PROJECT: NETOS-ISO
NUMBER: 4.0.1
DESCRIPTION: Process a NETOS message.
INPUT DATA: msgptr
INPUT FLAGS: none
OUTPUT DATA: none
OUTPUT FLAGS: error2
ALIASES: PROC_MSG
CALLING PROCESSES: Process Messages and Data
PROCESSES CALLED: Decompose Message
                  Process Network 4 Messages
                  Determine Channel Number
                  Build Queue Buffer for Qty = 1
                  Put Buffer in Queue
                  Level 4 Cleanup

ALGORITHM:
    Decompose message.
    If network message
        Process Network 4 Messages
    else
        Determine channel number
        Build queue buffer
        Put buffer in queue
        Cleanup Level 4.

REFERENCE: PROCESS SPOOLER MESSAGE
REFERENCE TYPE: SADT
REFERENCE: Smith's Algorithm, p. 23-24
REFERENCE TYPE: text
VERSION: 1.1
VERSION CHANGES: Added module "Level 4 Cleanup"
DATE: 11/25/85
AUTHOR: J. W. Foley

```

Figure 3. Data Dictionary Format for Structure Chart-Process (37:27)

The use of structure charts in the design phase is one area that has received considerable attention for research at AFIT. Attempts to automate the storage of information

```

PANAME: mess_parts
PROJECT: NETOS-ISO
DESCRIPTION: Decompose parameters.
DATA TYPE: Composite
MIN VALUE: none
MAX VALUE: none
RANGE: none
VALUES: none
PART OF: none
COMPOSED OF:  SRC
               DST
               SPN
               DPN
               USE
               QTY
               Buffer
ALIAS: Message Parts
WHERE USED: Passed from Decompose Message to Val Parts
COMMENT: Part of earlier design
REFERENCES: SADT - MSG_PARTS
REFERENCE TYPE: SADT
VERSION: 1.2
DATE: 11/05/85
AUTHOR: J. W. Foley
VERSION COMMENT: USE added to allow network msgs.

CALLING PROCESS: Process Message
PROCESS CALLED: Decompose Message(parts_list)
DIRECTION: up
I/O PARAMETER NAME: parts_list

```

Figure 4. Data Dictionary Format for Structure Chart-Parameter (37:29-30).

directly from structure chart into the database management system identified certain problems that required additional constraints to be placed on the design methodology (16). A typical structure chart is shown in Figure 5. It presents a graphical representation of information to help designers visualize and understand the relationships between processes and parameters (16).

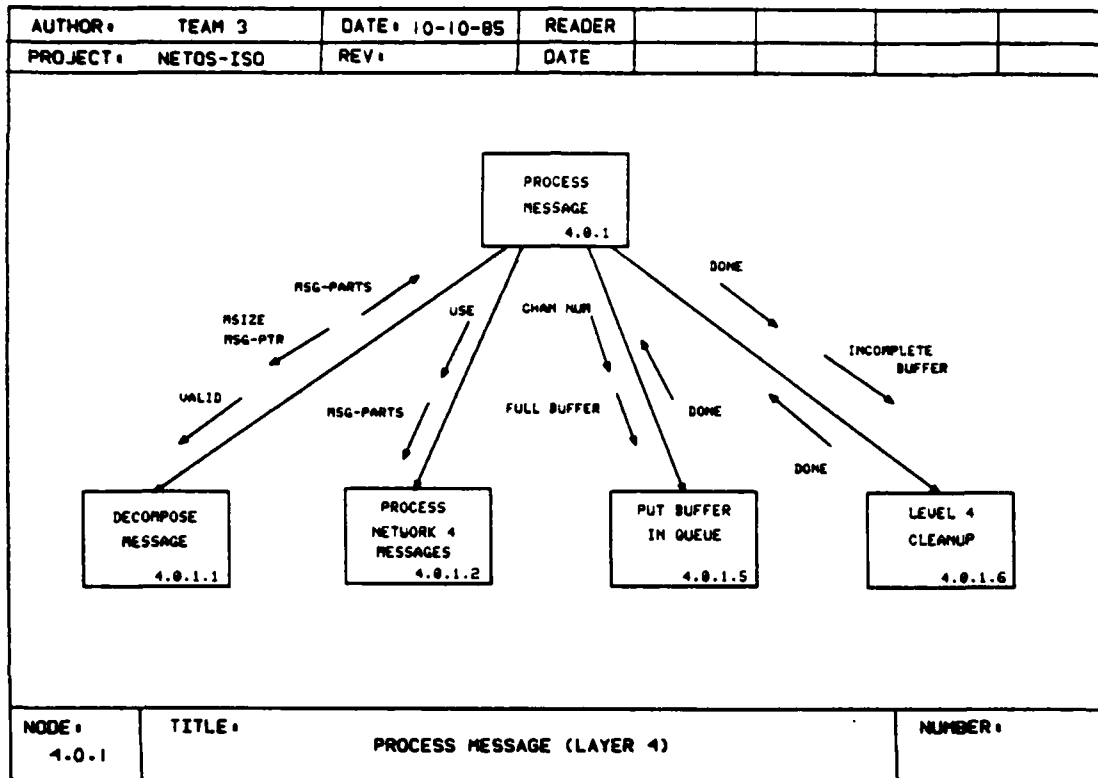


Figure 5. Typical Structure Chart (37).

A particular problem that involved the passing of parameters between processes is shown in Figure 6. As shown in a conventional structure chart in Figure 6 (a), two separate processes call a common process "sort2".

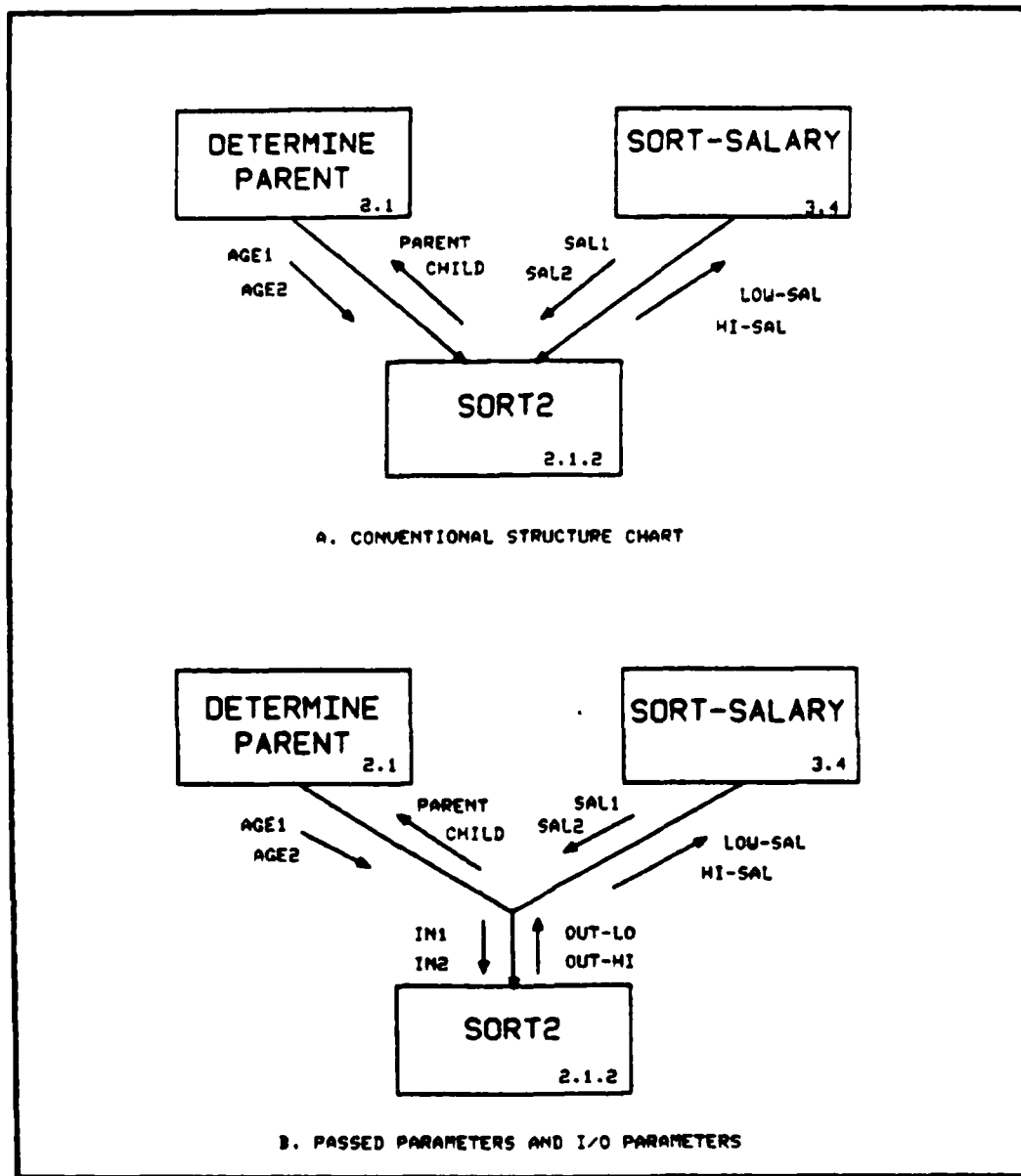


Figure 6. Example of Parameter Passing (37).

Each calling process passes to "sort2" two different parameters. The values returned from "sort2" are similar in

that they both represent numerical values, but their meanings are different -- "determine parent" expects a "parent" and "child" returned, while "sort-salary" expects a "low_sal" and "hi_sal" to be returned. The ambiguity in the returned values from "sort2" led to the anomaly condition that automated tools could not properly handle (16). Figure 6 (b) shows how the modified structure chart appears, adhering to the new parameter constraints. As research continues additional modifications to the conventional design tools should be incorporated as necessary. These seemingly small details of information are important and must be captured and provided for in the data dictionary system.

A second category of information that needs to be considered is support data, that data which directly supports the requirements of the Data Administrator (7, 22). To manage the database the DA has requirements over and above those of the user. The data dictionary system must be able to maintain an audit trail for the DA to monitor the contents of the database. The DA must also keep an accurate list of all personnel authorized to use the system, along with their privileges (read and/or write) and passwords. The DA must also have the capability to send messages to all users of the system when necessary. Although this data is important, it was estimated that this requirement was beyond the scope of this thesis. It is included in this chapter

for information purposes only to provide the reader with a better understanding of the requirements of a complete data dictionary system.

The database design to support the operational data requirements of design phase of the lifecycle was fundamentally determined prior to this thesis investigation. The chronology of events that led to the current design is summarized below. The original third normal form relations were designed as a result of one thesis effort (39) and numerous other classroom projects. This original set was subsequently modified, based on user experience with the system, which resulted in the removal of three items from process relations. Global variable information was deleted since the use of global variables is traditionally discouraged and the fields were seldom used in the design phase. Hardware I/O and File I/O were also removed primarily because they were much more implementation dependent and seldom used in the design phase. Other minor modifications to field names and sizes were made to arrive at the set of the third normal form relations currently implemented in the database and shown in Figures 7 and 8. Direct mapping techniques were employed to confirm that all data items within the data dictionary definitions were satisfactorily supported by the database shown above.

process:	
*project	c12 - Project name
*prname	c25 - Process name
number	c20 - Process number
prdesc:	
*project	c12
*prname	c25
*line	i2 - Description line number
description	c60 - Description text
pralg:	
*project	c12
*prname	c25
*line	i2 - Algorithm line number
algorithm	c60 - Algorithm text
processio:	
*project	c12
*prname	c25
*pname	c25 - Name of i/o parameter
direction	c4 - Input "in"/output "out"
pctype	c4 - "data" or "flag"
prcall:	
*project	c12
*prcalling	c25 - Calling process name
*prcalled	c25 - Process called name
prreference:	
*project	c12
*prname	c25
*reference	c60 - Reference description
reftype	c25 - Reference type
pralias:	
*project	c12
*prname	c25
*aliasname	c25 - Name of alias for prname
comment	c60 - Why alias is needed
prhistory:	
*project	c12
*prname	c25
*version	c10 - Version number of this entry
date	c8 - Date of this entry
author	c20 - Author of this entry
comment	c60 - Changes from last version

Figure 7. Third Normal Form Relations for a Design Structure Chart Process.

parameter:	
*project	c12 - Project name
*paname	c25 - Parameter name
datatype	c25 - Language independent data type
low	c15 - Lowest value allowed, if any
high	c15 - Highest value allowed, if any
span	c60 - Range of allowed values, if any
 padesc:	
*project	c12
*paname	c25
*line	i2
description	c60 - Parameter description
 pavalue:	
*project	c12
*paname	c25
*value	c15 - An allowable value for paname
 pahierarchy:	
*project	c12
*hipaname	c25 - Name of composite parameter
*lopaname	c25 - Name of component parameter
 paref:	
*project	c12
*paname	c25
*reference	c60 - Reference description
reftype	c25 - Reference type
 paalias:	
*project	c12
*paname	c25
*aliasname	c25 - Name of alias for paname
comment	c60 - Why this alias is needed
whereused	c25 - Process name where found
 pahistory:	
*project	c12
*paname	c25
*version	c10
date	c8
author	c20
comment	c60 - Changes from last version
 papassed:	
*project	c12
*paname	c25
*prcalling	c25 - Calling process name
*prcalled	c25 - Called process name
direction	c4 - Direction "up" or "down"
iopaname	c25 - Name of i/o parameter

Figure 8. Third Normal Form Relations for a Design Structure Chart Parameter.

There is a requirement to provide the communications interface between the workstation and the central computer. Ideally, this interface should be designed as an integral part of the "system" and activated within the environment of the workstation editor or interface software on the central computer. However, it was anticipated that this portion of the system would not be a high priority because of the existing file transfer programs available within the academic laboratories or commercially available to the students.

The user should have the capability to perform add, update, print, and view operations on the database from any location in the distributed environment (39). Users also have a requirement for additional operations (or queries) on the database, for example:

- 1.) Determine all processes that call a particular process.
- 2.) Determine all processes that use a particular parameter or aliases of the parameter.
- 3.) Determine all parameters associated with a particular project.
- 4.) Determine all I/O parameters of a particular process.
- 5.) Determine all aliases for a particular parameter.
- 6.) Determine all processes associated with a particular author.
- 7.) Determine all authors associated with a particular project.

The word "determine" could be interpreted as "display" on

the screen, "send" to a line printer or file, or "place" in a standard report. The operations listed above use the terminology for the design phase, but the operations should be interpreted as requirements for all phases of the lifecycle.

A data administrator should have access to all the user operations, plus some additional operations peculiar to supervisory responsibilities. It is understood that the DA has direct access to the database where the users do not, and, subsequently, can use the data definition language to extract any data item desired. A complete data dictionary system would provide software to facilitate the DA's efforts in managing the database. This software would assist the DA in

"...deciding the information content of the database, ...deciding the storage structures and access strategies, ...defining authorization checks and validation procedures, ...defining a strategy for backup and recovery, ...[and] monitoring performance and responding to changes in requirements" (7:25-26).

It would be desirable to have some facility that would automate the generation of data dictionary entries given a file containing structure charts or modules of code. Data entities could then be entered into the dictionary database without requiring the user to input it separately through the use of a special editor. There is a potential for increased efficiency, but not without cost. First, the amount of information extracted from the files may be limited and, therefore, may necessitate a substantial user

session with the database. Second, possibilities exist that undesirable inconsistencies will develop that could result in serious damage to the database. Third, recognition of redundancies (planned and unplanned) and multiple uses of entities are, at best, difficult by mechanical means. These redundancies and uses of entities will enter the database and may seriously decrement the effectiveness of the DDS system (22:2-13). This alternative is mentioned here for information purposes only because of its potential for use in data dictionary systems. It will not be addressed again within this thesis report.

Data Dictionary Editor Requirements

The data dictionary editor was required to run on the workstation. It was required to be a user-friendly designed editor that would allow the user to input all necessary data dictionary definition information as specified in the department guidelines. The purpose of the special editor was to provide the user with an efficient means of creating and modifying data dictionary definitions. SADT diagrams located in Appendix D outline the general requirements of the data dictionary editor as stated here.

The editor must be designed for evolution. The documentation standards, published by the Electrical and Computer Engineering department, are reviewed and updated periodically. Some department and research advisors require various other forms of documentation. Other environmental

conditions may dictate other changes in the system.

The editor must be designed to anticipate, prevent, identify, and correct errors. In addition to the items outlined in Chapter II, peculiarities of INGRES, such as unrecognizable characters, must be guarded against. When errors in processing information to the database occur, a transaction log of some kind would help identify what information was processed correctly, what failed to get processed and, if possible, the reasons for the errors. Some method of "pretesting" the file for correct format and nonblank key fields would also be useful.

A major requirement of the editor is that it be generic. This means that the editor software is not to be designed to support only a single phase of the lifecycle. It should be designed in such a fashion that it would provide the same editing facilities for any of the phases of the software lifecycle.

The editor is to be designed to be portable to other workstations with the minimum of modifications. Hardware dependent functions are to be separated from the main portion of the code to the extent possible to permit easy substitution modules for other machines. Only the standard 24 row by 30 column size screen is to be used and only one color is to be used.

The output of the editor was required to be in a format that can be read and understood by the interface

software running on the central computer. Likewise, the editor was required to read and understand the format of the output generated by the central computer.

Central Computer Database Interface

The interface software on the central computer is required to translate the definitions generated by the workstation editor and load these definitions into the database. Definitions needing updates are required to be retrieved from the database and sent to the workstation in a format that could be read and understood by the editor. Ideally, these operations should be accessible from any location within the distributed environment.

Database Interface Facilities. On-line processing facilities provide the user with immediate access to the database and should be provided (22). Interaction with the database is accomplished while the user is using the system. This method is generally used when only a short period of time is needed to perform the task at hand, or a small amount of information is needed quickly. Examples would include a user who needs to make a simple modification to an existing data dictionary entry, or a user who needs to verify a certain piece of data exists within the database.

Batch processing facilities should be provided since interaction with databases can be a time consuming and frustrating experience for a user (22). Batch facilities permit new data dictionary entries to be submitted to the

database through a batch of data already prepared in a fixed format acceptable by the system. The advantage of such a facility is that large amounts of information can be entered into the dictionary database without the user's presence required at the terminal.

Security of the Database. Protection of any data dictionary system from unauthorized access is important for two primary reasons. First, the dictionary database contains, by its very nature, a complete description of the organization's processing system, information that usually is not desired for public exposure. Second, and more important for the AFIT environment, is that the dictionary database must be a trusted and reliable source of information. Accidental or intentional tampering with the data will degrade its reliability and render the system less useful (22:2-20).

The purpose of the data dictionary system, the environment in which the system is implemented, and a host of other factors will affect how security measures will be employed. Generally, security systems should distinguish between the reading, creating, modifying, and deleting of data. The DA is usually tasked with managing the security issues (22:2-20,21).

In the AFIT academic environment, intentional tampering with the database is not a major concern. This does not, however, eliminate the requirement for security measures.

Security measures should include the following:

- 1.) Access to the dictionary database should be restricted to only those persons who have a validated requirement.
- 2.) Operations on the data that include destructive action (i.e. delete, modify) must be carefully managed to preclude any intentional or unintentional damage to the database.
- 3.) Direct access to the database via Ingres should be restricted to the DA. Users should access the data only through the provided tools.
- 4.) A responsible owner (programmer or team) should be defined for all data. Only the owner can modify the data.

Summary

This chapter described the requirements for a complete data dictionary system that supports the development of software in the AFIT distributed development environment. The most important requirement in this environment was capability of the workstation to effectively communicate information to the central computer database and vice-versa. Generic and portability requirements were two important issues that had to be considered throughout the design stage of the editor to facilitate the growth potential of that portion of the system. The contents of following chapter will focus on the design of the distributed data dictionary system based on the requirements established here.

IV. Data Dictionary System Design

The requirements identified in Chapter III established the foundation for the actual design of the data dictionary system. The data dictionary system consists of three separate but integrated components. The first component is the special editor designed for the workstation. The second component is the communications element between the workstation and the central computer. The third component is the interface to the database that exists on the central computer. The design and development of these three components are addressed separately in this chapter.

Data Dictionary Editor Design

The Zenith Z-100 microcomputer was chosen as the prototype workstation for the implementation of the data dictionary editor. This computer was chosen for two primary reasons. First, it supported the standard 16-bit operating system in use today by microcomputers (MS-DOS). Second, There are a large number of Z-100 computers available for student use in the academic areas.

The "C" programming language was the language of choice for this editor. This was because the Berkeley version of the database management system INGRES, used on the mainframe, only supports "C" in accessing the database through its embedded query language. Since the software on

the mainframe was going to be implemented in "C", it was logical (although not required) to use "C" in the workstation software as well. Another important consideration was the fact that "C" was supported by the majority of the other computers available for student use in the labs. This facilitates the portability of the code to these other computers.

A set of design structure charts for the data dictionary editor are located in Appendix E.

User-Machine Interface.

Dialogue. The user-machine interface dialogue consists of a screen-oriented combination of menu-selection and form-filling displays. Menu-selection is used in the initial stages of the tool while form-filling dominates the editing session of the tool.

Generally, menu-selection is the preferred method of obtaining user input when only a small, limited number of options exist for the user to choose (1, 10, 27). The initial stages in the execution of the editor tool require the user to identify whether a Create or Update session is desired (see Figure 9). In the event of a Create, the lifecycle 'phase' and 'category' must be identified (sample menus for creating a definition in the design phase are shown in Figures 10 and 11). If an Update session is desired, the user is prompted for the name of the file to update.

<u>PLACE CURSOR IN BOX, PRESS <RETURN></u> C] CREATE NEW DEFINITION C] UPDATE EXISTING DEFINITION C] EXIT PROGRAM			

Figure 9. Opening Menu on the Editor.

Choosing these menu selections, rather than requiring the user to type entries at the keyboard, provides a more efficient system requiring a minimum amount of training necessary to use. Menu-selection reduces the memorization

CREATE			
--------	--	--	--

PLACE CURSOR IN BOX, PRESS <RETURN>

☐ REQUIREMENTS ANALYSIS PHASE

☐ DESIGN PHASE

☐ CODE PHASE

☐ EXIT TO PREVIOUS MENU

Figure 10. Menu #2 for Create.

required by the user, reduces the potential for typographical errors, and reduces the number of required keystrokes (1, 10). It is expected that users of this tool

CREATE	DESIGN		
--------	--------	--	--

PLACE CURSOR IN BOX, PRESS <RETURN>

[] STRUCTURE CHART -- PROCESS

[] STRUCTURE CHART -- PARAMETER

[] EXIT TO PREVIOUS MENU

Figure 11. Menu #3 for Create.

will include those who use it occasionally (for classroom purposes only) and those who will use it frequently (for thesis research).

Menu-driven control of a program is sometimes

criticized for being too slow for experienced users (10). Because a maximum of three and a minimum of two menus can be presented to the user, the experienced user should not suffer from potential frustration or boredom with the tool. Other input devices were considered, such as a mouse or light pen, but were eliminated due to reduced tool portability and limited usefulness in predominantly text editing environments.

The form-filling method was chosen as the primary method for display and input in the editing portion of the tool. Predefined, formatted structures, called templates, are used as the basic form. There are three main reasons for this choice. First, the nature of the data dictionary definition lent itself very well to a blank form-filling operation, since a variety of fields exist for each definition. There are limitations on some of the entries for these fields, mostly related to the length of the input, but the vast majority of fields contained no limitations or restrictions on the information entered or format upon which it is entered.

Second, the form-filling method presented the information on the screen in a manner that closely resembled the format required by the department. Being familiar with the display of information, along with the syntax and semantics of the required input, reduces the initial shock and possible apprehension of the system users might

otherwise experience.

Third, form-filling provided a "highly disciplined mode of modification that guaranteed the structural integrity [of the data dictionary definition]" (40:102). Because movement of the cursor into unauthorized areas is prohibited, movement between fields is easily accomplished and input error checking capability is enhanced.

One alternative available for obtaining textual input was the existing implementation of the system where entries are made line-by-line as the system prompts the user with various data fields. Modification to any single item within a definition requires the review of the entire definition. This was the method employed in previous versions of the tool and was considered unsatisfactory. Line-by-line editing of a definition was determined to be too slow, too clumsy, and too inefficient to warrant its use.

Another alternative for inputting textual information was simply using an existing text editor to create definitions from scratch or update definitions inside a readable flat file. This alternative holds some merit in that it would eliminate the requirement for a special editor, but not without significant cost. The database management system INGRES is sensitive to precise characteristics of input data, such as upper case letters versus lower case letters, semi-colons versus colons, blank spaces versus commas or hyphens, and hidden commands such as

control codes or escape codes (11, 12). Attempts to communicate with INGRES in the wrong syntax would fail or possibly cause bad data to be stored in the database. A problem of even greater concern is one where a portion of the data definition is successfully loaded into the database when a syntax or format error causes the remainder of the data definition to fail to get loaded. The database is left with an incomplete definition, leading to an inconsistent and unreliable database.

The cost of overcoming these types of interface problems could be substantial. The interface software must be able to carefully analyze every line of text within the file for every possible syntax, grammatical, and format error. This includes identifying misspelled field identifiers, handling out-of-order sequences of fields, input that exceeds field lengths, and so on. There is no question that this type of interface software would provide a valuable service, but its design and development would require considerable time.

Screen Display. The actual display of information on the screen plays an important role in the design of user-friendly interfaces (1, 10, 18, 27). The screen display shown in Figure 12 was designed for this tool. All menus are displayed consistently throughout the tool with regard to their location on the screen, appearance, and method of choice selection.

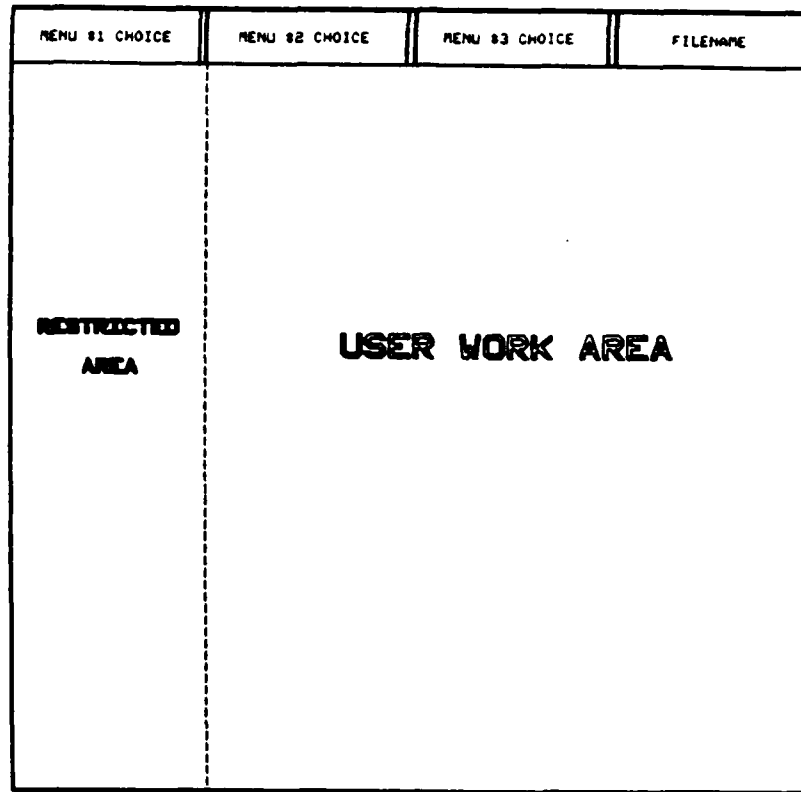


Figure 12. Sample Screen Display.

The top line of information provides the user with the location within the hierarchical structure of the program, and remains on the screen throughout the use of the tool. The results of each menu selection are provided in this line as the user traverses through the tool. This facility eliminates the requirement for the user to memorize this information.

The remainder of the screen displays the menus. General editing takes place in this area as well. While in

the edit session, the screen is divided into two parts, the left one-fourth of the screen and the right three-fourths of the screen. The left portion is reserved for field names only (title), while the right portion of the screen is reserved for user input (data).

While editing a file, the cursor is restricted from moving into the left portion of the screen. All input areas are highlighted in reverse video, clearly indicating the maximum length of each entry. The restrictions on the movement of the cursor (outside of any reverse video block) reduce the potential for error. These restrictions are designed to permit the user to concentrate on what information goes into each block rather than the mechanics involved in getting to the proper location on the screen.

An alternative in presenting the information on the screen would include the elimination of the top line that contains important information for the user. This would enhance the editing capabilities by permitting scrolling the text off the top of the screen as required, versus the repainting the screen that is employed with the presence of the window. The decision was made to keep the line of information because of its projected value to the user and accept the degraded mode of scrolling.

The information displayed on the screen is exactly what is stored in the data file. This system is commonly known as "what you see is what you get." This provides immediate

feedback to the user that the tool is doing useful work. This contributes to providing the user with the psychological closure that is important in the design of any user-friendly system (27).

One issue that might generate some concern is the screen becoming cluttered with textural information. This is a distinct possibility if some of the multiple line entries in a definition contain a large number of lines of text. The degree of distraction or confusion this might cause the user is expected to be minimal since the text on the screen will be similar to any text file with which most users will be familiar. Blank lines are used to separate each field, which will alleviate some of the potential for clutter.

Data Structures. One primary data structure is employed in the editing phase. This particular structure represents a single line of data in the buffer (and on the screen) and contains additional fields that define the type of information in that line. The term "buffer" refers to the entire data dictionary definition in memory (it was established during the design of the editor that only one data dictionary definition would, initially, reside in memory at any one time). The contents of the buffer are linked together in a doubly-linked circular list. Linked lists are used extensively in editors because they so effectively support insertions, deletions, allocation and

deallocation of resources (26:173). The doubly-linked structure permits easy traversing forward and backward through the buffer. The circular list concept provides immediate access to the front and rear of the linked list.

The data structures that were linked were actual 'structures' in the C programming language, similar to 'records' in Pascal or Ada. Each structure defines one complete line of information that appears on the screen (see Figure 13).

The first and last items in the structure are pointers and provide the forward and backward links in the linked list.

The Title field stores the title of the type of information found in the next field called Data. The character string contains a maximum of 20 characters. Examples of entries in the Title field are "NAME:", "PROJECT:", and "VERSION:". Three additional keywords, or symbols, are used to identify related information. The keyword "blank" represents a empty structure and is used to provide blank lines on the screen that separate entries. The keyword "(cont)" is used to identify the first line of a multiple line, multiple entry item that has been inserted into the linked list. The symbol "***" is used to identify the second, third, or fourth line of a similar entry. The printable information in this field appears on the screen in the area labeled "RESTRICTED AREA".

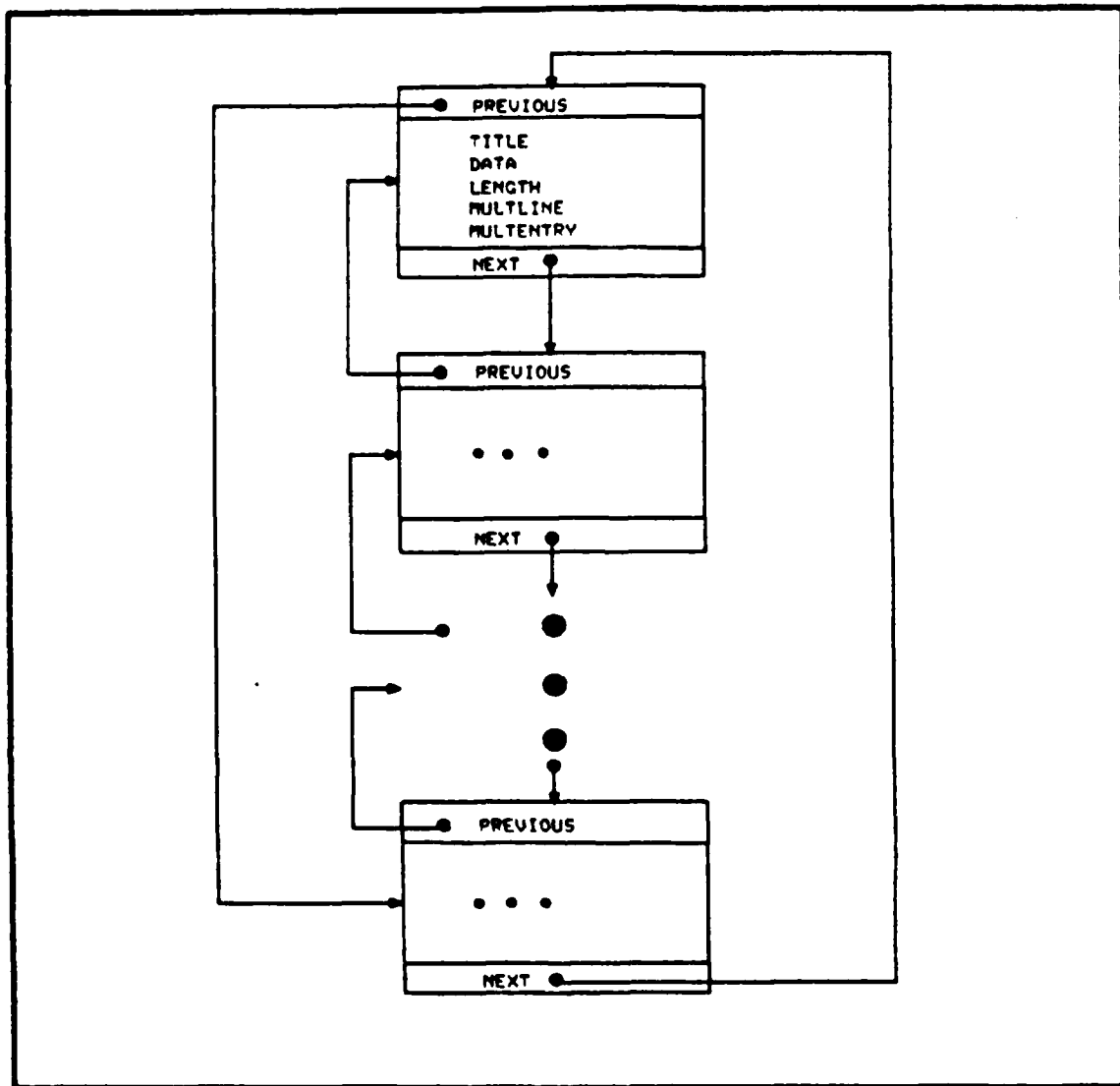


Figure 13. Linked Structure Used in the Editor

The data field contains the input from the user, its length determined by the integer field length. The maximum length of the field is 60 characters, restricted by the size

of the video screen. The information in this field appears in reverse video on the screen in the area labelled "USER WORK AREA".

The fields 'multiline' and 'multentry' together provide important information as to the type of entry permitted in the particular structure. There are three types of entries available for any data dictionary definition. The first is a single line entry. A value of '0' in the 'multiline' field indicates this type of entry. The second is a single line with multiple entries permitted. A value of '0' in the 'multiline' field and '1' in the 'multentry' field indicate this type of entry. The third is a multiple line with multiple entries permitted. Values greater than '0' in both fields indicate this type of entry. The 'multiline' field contains the number of the line with respect to the total number of lines (for example, a 1, 2, or 3 in a 3-line entry). The 'multentry' field contains the total number of lines per entry (either 2, 3, or 4 in the current implementation). Therefore a structure containing the values 'multiline' = 2, 'multentry' = 4 indicates that the data field in the structure contains the second of four lines of information for a particular entry. These fields are used during the edit phase to control line insertion and deletion. They are also used in preparing the data for downloading into the database.

As the tool reads a template (if CREATE) or an actual

definition (if UPDATE) from a file, structures are dynamically allocated space in memory and inserted into the buffer in the form of a linked list. This enables the tool to make efficient use of memory. Line insertion operations, within the editor, cause space to be dynamically allocated for new structures which are inserted into the proper location in the linked list. Line deletion operations cause the reverse set of actions to occur. Line insertions and deletions clearly demonstrate the advantages of using linked data structures for editing environments.

Windowing Scheme. The elements of the visual display discussed in this section focus on how information gets to the screen. It should be apparent that no data dictionary definition is going to fit on the screen in its entirety, or even come close. A system had to be developed that would control what portion (or window) of the definition buffer would be displayed on the screen and how windows of information could be moved on and off the screen. This is a basic requirement of screen-oriented editors.

The system designed and implemented in this tool involves a "windowing scheme" (see Figure 14).

Global pointers are defined to keep track of the top of the buffer (topbuffer), the bottom of the buffer (botbuffer), the top of the window (topwindow), and the bottom of the window (botwindow). Topbuffer points to the

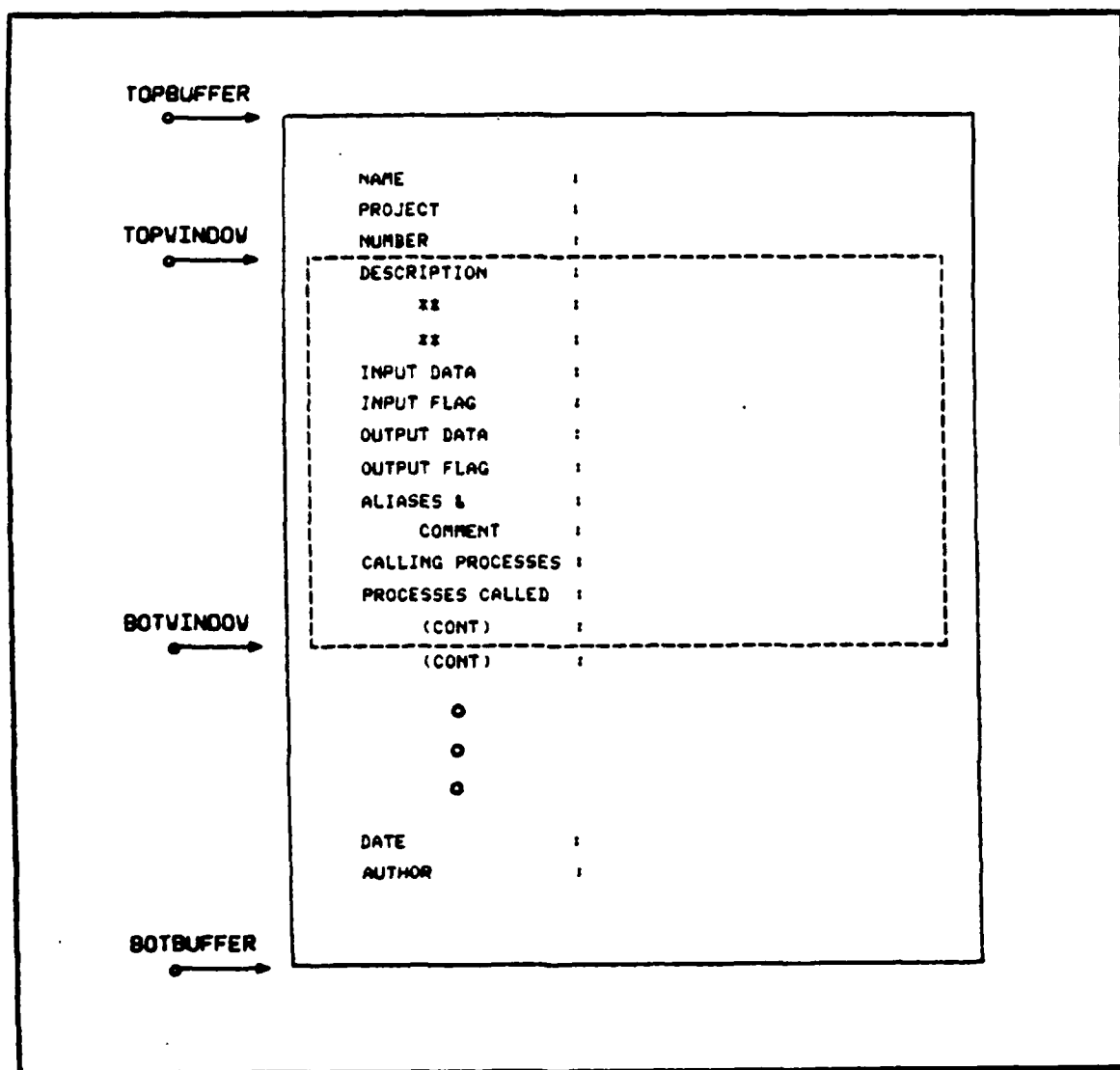


Figure 14. Screen Editor "Windowing Scheme".

very first structure in the linked list, while botbuffer points to the very last structure in the linked list. These

two pointers generally do not move once initially established, unless the structure to which either points to can be added onto or deleted. The topwindow and botwindow pointers, on the other hand, move about the buffer frequently always remaining 21 items apart (the size of available screen space for this editor is 21 lines). Initially, when entering the editor, the topwindow is set to topbuffer and the top 21 items in the buffer are displayed. As the user attempts to move the cursor down off the bottom of the screen, the topwindow and botwindow pointers are adjusted accordingly and the new window is presented on the screen. The window pointers will change frequently during an average editing session with the obvious restrictions of the topwindow pointer never "passing" the topbuffer pointer and likewise with the botwindow and botbuffer pointers.

Although the concept of using window pointers is simple, it was tricky to implement at times. The difficulties resulted from two objectives of designing a user-friendly system: 1) to minimize the redrawing of the screen (take advantage of screen memory operations), and 2) provide for dynamic insertion and deletion of lines of text. The first objective was negotiable to some extent, but the second objective was clearly not negotiable.

Screen memory operations permit the screen to appear to move instantaneously up one line when a line is deleted. The opposite occurs when lines are inserted. The lines of

data are inserted and deleted in screen memory only and actions must take place to ensure the buffer under goes the same changes. Also, every time a line is inserted or deleted the window pointers have to be changed accordingly. Checks have to be made to ensure window pointers do not exceed their boundaries. Additional lines have to be printed to the bottom of the screen during deletions. Multiline entries, where inserts and deletes affect groups of lines, complicated the matter further.

Three additional global variables (two integer and one pointer) were required for the implementation of the screen editor. The integer variables are 'curx' and 'cury', the current x and y coordinates of the location of the cursor. The additional pointer is called 'current' and always points to the structure in memory that corresponds to the line of text on the screen where the cursor is located. The management and control of these variables is crucial in the storing of data in its proper location in memory. The values of 'curx' and 'cury' change every time the cursor moves horizontally or vertically, respectively, on the screen. The structure pointed to by the pointer 'current' changes every time the cursor moves vertically.

The same complications arise in the management and control of these three variables as discussed earlier with the window management. The system must be constantly aware of these variables and their values to provide a working

editing environment.

The justification for making these seven variables global lies primarily in the fact that many modules must have access to them and possess the ability to change their values. They could be declared as local variables, but, that would necessitate the passing of lengthy parameter lists between modules. The inherent danger in using global variables is that their values can be changed by virtually any module. Careful consideration was used in determining which modules only needed access to the values of these variables and which modules needed the capability to alter their values.

The effective control and implementation of these seven global variables was the key to the successful implementation of the screen oriented editor.

Data File Input and Output. Ideally, the method for handling data files used by the tool is to hide the implementation details from the user completely. The user would be required to know only the names of the processes or parameters, with all file I/O being based on this information.

The current implementation requires the user to define filenames and keep track of them as necessary. Upon completion of creating a new data dictionary definition the user is prompted for a filename to store the definition. Safeguards are employed to ensure accidental overwriting of

other files does not occur. When updating an existing definition the user is prompted for the name of the file to update. After updating the definition the user is offered a choice of: 1) overwrite the current file; 2) make a backup copy of the current file before saving the new definition under the same name; or 3) define a new filename for the definition. The only option that requires the user to enter a filename is when defining a new name. The other options proceed automatically when the response key is pressed.

Although not ideal, this method of file handling does provide a working system that appears clean and simple to the user, yet is flexible and powerful enough to provide most desirable alternatives for the user.

The format of the flat files generated by the workstation editor and the database interface software is shown below in Figure 15.

The first six lines provide important information about the definition contained in the file. Line 1 contains a verification code that identifies the file as one compatible with the data dictionary software. Lines 2 and 3 contain the phase of the software lifecycle and the category within that phase. Line 4 is the status line and provides the status of the definition. It will have a value of 0 if the definition is unchanged, a 1 if the definition was changed, and a 2 if the definition is to be deleted from

```

1 --          #*#
2 --          DESIGN
3 --          PARAMETER
4 --          1
5 --          Mon 5 May 86 13:25 hrs
6 --          Mon 5 May 86 13:32 hrs
Title --      NAME
Data --      mess_parts
Length --     25
Multiline --  0
Multentry --  0
Title --      blank
Data --
Length --     0
Multiline --  0
Multentry --  0
Title --      PROJECT
Data --      NETOS-ISO

      o
      o
      o

```

Figure 15. Format of Flat Files

the database. This status enhances the communication between the workstation and mainframe, and eliminates the need for reloading a definition into the database if no changes were ever made by the user. The next two lines contain data relating to the time required to edit a definition. The values are obtained from system calls to the MS-DOS operating system. These values are stored in the database and are used to analyze the times required to edit a definition.

The remainder of the file contains the actual definition of the data item. The information is stored in groups of 5 lines that correspond directly to the storage

structure used in the actual code, as described in the data structures section earlier in this chapter. The names of the fields are "Title", "Data", "Length", "Multline", and "Multentry".

Editor Commands. The commands available to a user while using the editor clearly have a major impact on the usefulness of the tool. "Concern for human engineering dominates the design" (19:163). The minimum set of commands that will provide a working system include "insert" characters, "delete" characters, and "save" the text at the completion of the edit session. Although these commands provide a working system, more commands were required to increase productivity and enhance the user-friendliness of the system.

The capability of moving the cursor freely to any portion of the input area is important. The user must have this capability to view or modify any portion of the file at any time during the session. Cursor movements by character, line, or screen-at-a-time were considered essential for the editor to be of valuable use.

The files to be edited contain three categories of data items. The first category is single line entries. The other two categories provide for multiple line entries within their respective data fields. Facilities were provided to accommodate the requirement of creating additional lines as appropriate for these fields.

on the design of the editor" (19:163). Editors must be able to handle incorrect commands gracefully. Handling includes the detection and correction of errors, where possible. Error handling is provided at the lowest possible level in the code.

Illegal strings of characters for filenames and dates are identified upon their entry. Facilities are provided to permit the user to reenter the correct information. Errors in reading from and writing to files cause error messages to be displayed on the screen with specific information as to the error, if known. Facilities are provided, when appropriate, to allow the user to reenter a filename.

While inside the actual editor, illegal characters from the keyboard are suppressed at the time the key is pressed. Illegal characters include escape and control codes not used by the editor and also characters that are not accepted by the database management system INGRES (namely square brackets "[" and "]"). In most cases the bell sounds to indicate an illegal character.

Other errors that are checked by the tool are those that relate to the definition requirements and not the actual codes from the keyboard. An example of this type of error is verifying that key fields (required by the database system) are non-empty. The checking is performed when the "exit" command is requested. Blank key field errors must be corrected before the buffer is written to a file.

System errors, particularly in the area of dynamic allocation of memory, are also checked for and handled at the lowest possible level. If an error occurs in dynamic memory allocation during the initial reading of a template or data file, any memory that was allocated is freed, an error message is displayed on the screen, and the system returns the user to the top level menu. Workstations with 128K, or more, of memory should not experience problems due to memory limitations, and therefore memory allocation errors may be a sign of some other, possibly more severe, problem with the system.

The form-filling construct of the editor itself prevented a number of otherwise common errors from occurring, for example entering characters in the wrong location or typing characters beyond the length of the field. Input fields were clearly identified and cursor movement is limited to only input areas. This was one of the major reasons that the form-filling format was the method of choice for this editor.

Communications Interface

The communications interface between computers is an important issue in the discussion of distributed systems of any kind. Because there exists software that provides for the transfer of text files between computers, this component of the data dictionary system was not assessed further in this thesis.

Database Interface Design

The "C" programming language was the language of choice for the implementation of the database interface software. This was due, as stated previously, to the fact that the Berkley version of the database management system INGRES (that runs under Unix) only supports the "C" programming language in accessing the database through its embedded query language EQUQL. Database query languages provide users with the ability to retrieve information from the database. Often database queries, in the query language, can be embedded in other software programs to enhance the user friendliness and efficiency of database management. "C" was the only programming language available for this facility.

The interface software currently only supports the design phase database (for structure charts). As the other databases are designed, the interface software required to support them should be straight forward, based on this code prepared for this thesis.

The database interface software was designed to translate a text file containing a data dictionary definition, generated by the workstation editor, into the database and vice-versa. The text file is first read into memory where it is placed into structures, also known as records in Pascal or Ada programming languages. From memory the definition is placed systematically into the proper

relations in the database through the use of the database embedded query language. During the reverse process, definitions are retrieved from the database, placed into structures in memory, and subsequently written out to a file.

The format of the text file generated by the interface software is exactly like the format of the file generated by the editor. This facilitates the use of the files in both locations and eliminates the need for additional translator software.

The opening menu prompts the user for his desired use of the tool. The options available are Load a definition into the database, Retrieve a definition from the database, Delete a definition from the database (must access the password), or Exit the tool. Depending on the user's choice, the names of the process (or parameter) and the filename are requested. System messages are frequently placed on the screen to keep the user informed as processing takes place inside the database. At the completion of the action the user is informed of the same and control is returned to the top level menu. The top level menu is the only location where users can exit the tool other than aborting the program.

Currently the tool provides operations on single files (data dictionary definitions) only. Also, at the time of publication of this thesis, the user is not provided the

facility to perform the operations in the background and therefore the user must remain with the program during the uploading or downloading operations.

Summary

The highlights of the design and development processes of the data dictionary system were presented in this chapter. What follows in Chapter V are discussions on some of the key implementations issues and the testing processes used in this project.

V. Implementation and Test

There are three areas that merit discussion that relate specifically to the implementation of the software. The portability of the editor and the generic implementation are the first two areas discussed. A brief description of how the system operates is the third area. The last section in this chapter focuses on the testing phase of the software lifecycle and how testing was performed for this system.

Portability of the Code

Every effort was made to use standard C-language syntax and library functions to enhance the workstation editor's portability between environments. System calls to MS-DOS were kept to a minimum (two each) to reduce the dependency of the software to that operating system. Modifications to the code, with respect to these two areas, should be minimal when transporting it to other machines.

The hardware dependent software is contained in a separate library of code. This library is linked with the remainder of the code to produce the working editor for the Z-100 microcomputer. The primary functions found in the library are those that address screen memory operations. These functions are dependent upon a 24 row X 30 column video display and the distinct codes sent from the keyboard and to the screen that are unique to the Z-100. The Z-100

is especially well equipped in providing easy access to these operations. The functions must be analyzed to determine necessary modifications for other machines. The code was modularized to the extent possible to enable modifications to be made with minimum difficulty. Preprocessor define statements were employed to the extent possible to permit easy access to the codes that drive the editor operations. A configuration file of this nature is required for any destination machine for which the tool is targeted.

The "C" software written for the mainframe computer (under Unix) contains no specific calls to the Unix operating system. The embedded queries to the database are basic queries in that they follow specific examples in the documentation. Although recompilation of the code will be necessary, the database interface software should interface with the version of INGRES running under the VMS operating system with minimum modifications.

Some costs were associated with the efforts to keep the code portable to other computers. Designing the code for a 24 line by 80 column screen eliminated the option to employ the 25th line available on the Z-100 screen. This line could have been activated and used to present the information currently shown at the top of the screen. This line is not affected by line insertion and deletion or any other scrolling operations. Because this line was not

available, redrawing the screen was necessary when attempts were made to move the cursor beyond the top or bottom of the screen. Other coding methods may have resolved portions of this problem but they would have required significant time to implement. Paging commands that scrolled up and down 16 lines in the buffer and quick movements to the top and bottom of the buffer were provided to minimize the need for redrawing the screen. Other niceties such as color and the convenience of the keypad were not not considered since these two items are not provided for on many types computers.

Generic Editor Implementation Issues

The configuration of the buffer structure used in the editor code provides a facility to handle a variety of data dictionary definitions. Any number of different templates can be created and used with this editor with the basic restrictions being the length of the title and data fields.

The original code was developed using the data dictionary definitions associated with the design phase, structure charts in particular. After the initial integration of the editor software with the mainframe software was completed, templates for the code phase data dictionary definitions were generated. The stubs for the code phase in the original code were replaced by calls to modules that loaded these new templates when requested. All editor commands were successful in editing these definitions

with one exception. The definition of a module contained a multiline entry as the last line in the buffer which was not previously required for the design phase definitions. Minor modifications to the "delete line" module were necessary to handle this additional requirement. The definition of a variable required no modifications. The definition of this structure was the single most critical element in the design of a generic editor.

Operation of the Tool

A brief overview of the operation of the system is presented below. A complete users' manual is located in Appendix D for the interested reader.

The workstation editor is activated by the user entering the name of the program at the MS-DOS system prompt. The date is requested, which remains in memory for the duration of the session. The initial menu presented to the user requests the user's purpose for using the tool: 1) CREATE a new definition, 2) UPDATE an existing definition, or 3) EXIT the program. This is the only location where the user can exit the program (other than rebooting the system). Standard interrupts such as 'CTRL C' are not recognized by the system.

If CREATE is chosen, the user must identify the phase of the software lifecycle (Requirements, Design, or Code) at the next menu. The third, and final, menu presented to the user requests the user to identify the category of data

dictionary definition (for example, Process or Parameter in the Design phase). After this decision is made the system reads in the appropriate buffer template from a data file and places the empty template on the screen ready to accept editing commands from the user.

If UPDATE is chosen, the user is requested to enter the filename of the data dictionary definition to update. If the filename is found the file is loaded into memory and the user is placed into the editor with the definition visible on the screen. If the file is not found the user is notified of the error and control is returned to the main menu.

At the top of the screen there are four locations where the result of menu choices and the current filename, if an UPDATE operation was chosen, appear to keep the user informed of his current status at all times.

The user signals the end to the editing phase of the session by executing an EXIT or QUIT command. Once the specifics of this operation are completed, control returns to the top level menu.

The definition files created or updated by the editor are then transferred to the central computer by the Kermit communications program or any other program available to the user. The database interface software is activated by entering the name of the program. Menus prompt the user for the desired activity - upload a definition to the database,

retrieve a definition from the database, or delete a definition from the database (requires a password). The system either reads a definition file designated by the user or writes a definition out to a file designated by the user. Files are transferred back to the workstation in a manner similar to the method used to get them to the central computer.

Testing

The testing process of large computer programs incorporates, on the average, 30-50% of the effort (4:7). This large percentage indicates that the testing of software is an important part of software development and the time allotted for it should not be underestimated.

Testing is generally defined as "the process of executing a program (or portion of a program) with the intention, or goal, of finding errors" (30:172-173). Testing is often confused with debugging, which is generally defined as determining the cause of the error and correcting it. They are closely related as the output from the testing process provides the input for the debugging process.

Testing of the software should occur throughout the development process. The earlier testing takes place the better, since errors found in the late stages of the process have proven to be more costly than those found earlier (33). The categories of testing differ, depending on the location in the development process. The following paragraphs

describe the different categories and how each was incorporated in the development process of the data dictionary editor and associated software.

Individual module testing took place whenever possible. Small test programs were frequently generated to test unfamiliar aspects of the C-language, as well as verifying the correctness of algorithms employed for various functions. Once a module was sufficiently tested to be free of errors it was tagged as ready for incorporation into the next phase of testing.

Integration testing followed modular testing as modules became available. Once the minimum set of modules for a functional editor were ready, the modules were integrated and tested as a system. After the basic foundation of the editor was tested, enhancements to the editor, in areas such as screen displays and editing commands, were individually tested (when possible) and integrated into the editor one at a time. The editor software was again tested with the enhancement installed.

This procedure proved to be a very efficient method for building a working system from an established, working foundation. Each module was integrated into the editor with the confidence that all existing software was correct. Although it did occur, it was not common to find errors in the existing code that were previously overlooked.

Regression testing involved the retesting of the

software after an error was found and corrected. How much retesting was required depended primarily on how much code was affected. As errors were found in the later testing phases, regression testing was required to identify additional errors that resulted from the effects of the updated software.

Topdown testing focused on the control program, the data flow, and the control flow of the software. Test stubs were used frequently to verify correct flow of control between modules. This testing method was employed in conjunction with integration testing.

Once the editor code was complete, acceptance testing of the system was performed. The results of this testing phase are included in the following chapter on Evaluation.

In retrospect, it was clear that the systematic approach to building a system on a sound foundation, one block at a time, enabled the tool to be developed smoothly and efficiently with a high degree of confidence in its performance. At only one point, in the latter part of the code phase, was there any significant debugging delay. That delay occurred during some integration testing of the editor software. It would be hard to predict how much longer it would have taken to produce the same quality of software had the above procedures not been followed.

Summary

This chapter presented the significant factors that resulted from the implementation of the code. Testing was discussed to the extent that it was applied during the implementation phase. Validation testing was not performed to the extent of employing a separate group of people to execute the program with the intent of finding errors. Evaluation of the workstation editor was conducted by a group of 24 persons, however this evaluation focused on the user-friendliness and usefulness of the program as apposed to validating it. The following chapter summarizes this evaluation by a subset of the target user group.

VI. Evaluation of the Data Dictionary Editor

It is important to provide systems that are user-friendly and perform a useful service to increase the productivity of information systems (3:530). The subject of measuring user satisfaction will be discussed in this chapter with particular attention being paid to the background and justification for the tool chosen for this evaluation. The data dictionary editor for the Z-100 was evaluated by one faculty member and 22 graduate students currently enrolled in the computer engineering or computer systems program at AFIT. The results of this evaluation will also be presented.

Measuring User Satisfaction

Measuring a user's satisfaction with any degree of statistical proof of validity has proven difficult. Bailey and Pearson (1983) performed extensive research on the subject of measuring user satisfaction with information systems (3). The results of their research indicated that many such evaluations had been performed but none had established any standard of measure for which to analyze the evaluation results (3:530-1). Questionnaires used asked for numerical responses to questions without providing a place for the user to justify their response. There was little evidence as to why specific items or factors were chosen for

the various questionnaires. In addition, there were discrepancies as to researchers' opinions on factor importance in measuring user satisfaction. As discussed by Bailey and Pearson, these problems clearly showed a need to establish

a definition of satisfaction which contains a complete and valid set of factors and an instrument which measures not only the user's reaction to each factor but why the respondent reacted as he did (3:531).

A formula for defining user satisfaction as "the sum of the user's weighted reactions to a set of factors" (3:531),

$$S_i = \sum_{j=1}^n R_{ij} W_{ij}$$

was developed by Bailey and Pearson. In it, R is the reaction to factor j by individual i, and W is the importance of factor j to individual i. This equation suggests that one's opinion of satisfaction is the sum of his or her positive and negative reactions to a particular set of factors.

Bailey and Pearson conducted extensive tests in an attempt to identify all factors that people considered important in measuring satisfaction. They reviewed twenty-two studies to arrive at an initial list, and tested its completeness and accuracy by consulting data processing professionals and middle manager users in 8 different organizations. Critical incident analysis techniques were used for these tests. A list containing thirty-nine factors ranging from flexibility to vender support resulted.

To measure a user's perceptions of these factors,

Error Recovery. The extent and ease with which the system allowed you to recover from user induced errors.

unforgiving	_ _ _ _ _ _ _	forgiving
incomplete	_ _ _ _ _ _ _	complete
complex	_ _ _ _ _ _ _	simple
slow	_ _ _ _ _ _ _	fast
unsatisfactory	_ _ _ _ _ _ _	satisfactory
To me this factor is		
unimportant	_ _ _ _ _ _ _	important
Comments:		

Figure 16. Sample Survey Question

Bailey and Pearson used the semantic differential technique (3:533). Four bipolar adjective pairs ranging from negative to positive feelings were identified for each factor. Additional scales were included to test the internal consistency and validity of the four pairs, and to obtain a value for the weights users assigned to each factor. A seven-interval scale was adopted to measure the user's satisfaction with each pair. Figure 16 shows a sample question (or factor) that was taken from the actual questionnaire.

Numerical values were assigned to each of the seven

intervals ranging from -3 to +3. The importance scale was assigned values from 0.10 to 1.00 in increments of 0.15. The higher the value the more important the factor. The overall satisfaction was measured by

$$S_i = \sum_{j=1}^{39} \frac{W_{ij}}{4} \sum_{k=1}^4 I_{i,j,k}$$

where W = the weight assigned to factor j by user i , and I = the numeric response of user i to adjective pair k of factor j .

The results of this equation can be deceiving if a user rates half of the factors very high and very important while rating the remainder neutral and very unimportant. The numerical result would be approximately one-half of the total possible (60 out of 117) indicating a moderate degree of satisfaction when in fact the user was extremely satisfied. Normalizing the scores and filtering factors whose top four adjectives pairs were all rated at 0 eliminated this problem.

Extensive reliability and validity tests were conducted by Bailey and Pearson on their measurement tool (3:535-537). "Reliability is defined as the absence of measurement error" (3:537). The reliability coefficients calculated for each factor were found to be very high (average of .93, minimum of .75). The authors concluded that their questionnaire was reliable.

Three categories of validity tests were performed to determine if the questionnaire measured what it was designed

<u>Normalized Score</u>	<u>Translation</u>
+1.00	maximally satisfied
+0.67	quite satisfied
+0.33	slightly satisfied
0.0	neither satisfied or dissatisfied
-0.33	slightly dissatisfied
-0.67	quite dissatisfied
-1.00	maximally dissatisfied

Figure 17. Score Boundaries for Normalized User Satisfaction (3:535).

to measure. These types of tests considered were content validity, predictive validity, and construct validity.

Content validity implies that all aspects of the attribute being measured are considered by the instrument... Predictive validity implies that the instrument is consistent and agrees with other independent measures... Construct validity implies that the measurement instrument performs as expected relative to the construct of the attribute being measured" (3:535-6).

The results of their tests indicated that their instrument was valid.

As a result of their extensive investigation and analysis, Bailey and Pearson concluded that they have an effective instrument for measuring average levels of user satisfaction. The normalized scores for each user provide this data. A normalized score ranges from -1.00 to +1.00 with the translation of scores shown in Figure 17.

Evaluation of the Data Dictionary Editor

A graduate level software engineering class containing students studying computer systems or computer engineering were targeted for the evaluation. Although they were not knowledgeable in database management systems and were currently learning about the software development lifecycle and data dictionaries, it was determined that they would still provide valuable feedback in an evaluation of the human-interface, or user-friendliness, aspects of the data dictionary editor.

The class was briefed by the author on the overall data dictionary system and the editor's place within the system. Each student was provided with a copy of the questionnaire, a users' manual for the editor (Appendix B), and a set of instructions outlining the steps necessary to execute the tool and obtain a hard copy of the data definition file (Appendix D). Two Z-100 computers were set aside in one of the school labs for their use.

Many factors identified by Bailey and Pearson did not apply to the evaluation of the data dictionary editor and were not included in its evaluation. Examples of these were vendor support and management factors. Mallory, in a previous thesis effort (25), added several factors to the list that provided greater emphasis on the human-computer interface aspect of software tools. The list was extended by one more factor, for this evaluation, that covered the

<u>User Number</u>	<u>Normalized Score</u>
19	0.977
8	0.814
18	0.704
5	0.680
2	0.662
7	0.660
13	0.602
17	0.593
6	0.593
4	0.586
11	0.577
3	0.546
9	0.520
15	0.517
21	0.493
1	0.478
23	0.466
12	0.430
22	0.395
16	0.382
10	0.351
14	0.210
20	0.140
mean for all users = 0.538	

Figure 18. Normalized Values Overall Satisfaction
by User

area of error prevention. The complete survey used in the evaluation of the data dictionary editor is included in Appendix C.

The completed surveys were returned one week after they were issued. The average time to complete the evaluation was 37 minutes. The results were analyzed with the normalized scores for each user shown in Figure 18.

AD-A172 406

DESIGN OF A DATA DICTIONARY EDITOR IN A DISTRIBUTED
SOFTWARE DEVELOPMENT ENVIRONMENT(U) AIR FORCE INST OF
TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.

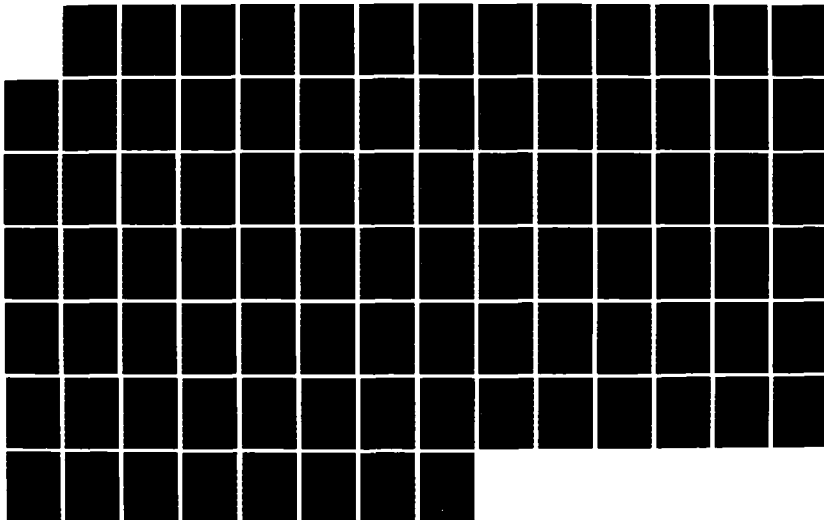
2/2

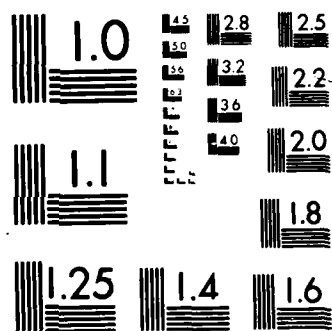
UNCLASSIFIED

J M FOLEY JUN 86 AFIT/GCS/ENG/86J-5

F/B 9/2

NL





The user number represents an arbitrary number used for accounting purposes only. These results indicate that the users were satisfied with the editor. The mean scores for each of the 11 factors and the mean overall satisfaction from question #12 in the survey are shown in Figure 19 (max score is +3.0, min score is -3.0). The mean score for each factor was obtained by first averaging the score of the first four adjective pairs for each user, then averaging these scores for all users.

<u>Factor Number</u>	<u>Mean Score</u>
1	2.13
2	1.85
3	1.57
4	1.51
5	1.61
6	1.62
7	1.83
8	2.30
9	1.79
10	1.84
11	1.74
12	1.91

Figure 19. Mean Scores for Each Factor.

Question 8 received the highest score which indicates that the users were pleased with the ease-of-learning aspect of the editor. Error prevention and recovery (questions 3 and 4) received the lowest scores. Two bugs were discovered during the evaluation which directly affected these marks.

The single aspect of the editor most often complained about (13 users) was the redrawing of the screen when the cursor attempted to move beyond the top or bottom on the screen. "The redrawing of the screen was totally annoying to me" was one user comment. This reaction was anticipated. The objective to keep the editor generic eliminated the use of the 25th line associated with the Z-100 screen. This line is not affected by scrolling operations and could have been used for the information window that currently is at the top of the screen. It is interesting to note that only one user specifically commented on the usefulness of the information window, although 3 other users made favorable comments about the display of information on the screen.

Seven users commented favorably on the ease-of-learning aspect of the editor, which reinforces the results of that factor shown in the previous figure (number 8). "I sat down and immediately wrote my data definition -- very easy to use" was one user's perspective.

Five users complained about the lack of an insert/delete character command in the editor. Having to retype the entire line of text for a single character change was a bother to them.

Five favorable comments were made about the compactness and usefulness of the on-line help screen, "You seem to have enough help not to get in my way but to be useful". One user added a recommendation to provide a help screen for the

initial menu as well.

Three users recognized an immediate need for the editor to support other related classroom and thesis work and expressed a desire to use it.

A variety of other comments were made that are worthy of mention despite only being addressed by one or two users. One user recognized a limited usefulness of the editor since it only handles single files rather than multiple files and records. One user, who had some experience with previous versions of the data dictionary interface, indicated that this editor was a vast improvement over the old system. One user complained that a directory listing was not available from inside the program. Two users desired more explicit error messages be displayed rather than just the computer beeping when an illegal character or command was entered. One user was not sure what this editor did that a word processing program could not do. One user commented favorably on the ability to move freely about the buffer to enter the definition.

Conclusions

The overall normalized measured satisfaction mean of 0.535 generated by the formula presented, coupled with the user assessed satisfaction mean of 1.91 from question #12, and the fact that there were no negative mean scores for any single factor indicate that the data dictionary editor was well received by the users. Enhancements are certainly in

order to make the environment better, particularly in reducing the redrawing of the screen.

It is important to emphasize the point that this evaluation was limited to the editor only, and not to the overall system. In this respect the actual usefulness of the editor, as it pertains to the entire system, could not be measured.

VII. Conclusions and Recommendations

Conclusions

A data dictionary system was researched and designed specifically to support the AFIT distributed software development environment. The system was broken down into three separate but related subsystems, namely the workstation, the communications links between computers, and the central computer.

A special data dictionary editor was designed and implemented on a prototype microcomputer workstation. It was designed under the constraints of being generic and portable to other workstations. The generic objective was achieved, as the editor demonstrated its ability to handle code phase definitions after design phase definitions were used for the original implementation and testing of the software. The portability of the software to other workstations was not tested.

Twenty-three users evaluated the user-friendliness of the editor and were satisfied with its implementation. This level of satisfaction was based on a normalized mean score of 0.533 on a scale of -1.0 to +1.0 where the higher the score the more satisfied the user was with the system. Although the previously designed interface to the data dictionary was not evaluated by the same standards, the new interface is believed to be a major improvement based on the

enhanced features, such as screen oriented editing and error checking. Editing time should be reduced with increased accuracy. Data definitions templates for particular projects can be created by the user to provide an even more efficient system.

Comparisons of system response time cannot be based solely on the editing times. Total response time for the distributed data dictionary system includes the time required to transfer files between computers. This requires additional resources (communications links, communications software) and time that the earlier versions of the system did not. How the overall systems compare in terms of overall response time will not be known until such evaluations take place.

The communication links of the system were not addressed in detail in this thesis. This was because communication software was already available for use and did provide all the requirements for this first implementation of the system.

The database interface software was designed and implemented on the Vax-11/780 computer under the Unix operating system. The Berkeley version of the INGRES database management system was the host database for the data dictionary definitions.

A great appreciation was gained for the concept of user-friendliness, particularly its wide variety of

definitions. What is user-friendly to one person is not necessarily the case for others. Making design decisions to accommodate anticipated desires for the best environment was based on research conducted and presented in the literature review chapter. The additional constraints placed on the system, as a result of the goals of a generic and portable system, proved to be costly as brought out by the evaluation of the editor.

A great appreciation for the design and development of large (relative to personal experiences) software development projects was achieved. The software lifecycle demonstrated its validity more than ever before.

Much was learned about the workings of the Z-100 microcomputer and the MS-DOS operating system. Specifically, controlling the keyboard and manipulating screen memory of the Z-100 was enlightening.

A great appreciation was gained for the efforts required to write an effective screen oriented editor. Experiencing the power of linked lists, understanding editing environments and how fast editing code grows were all educational.

Recommendations for Further Study

Enhancing and expanding the editing environment on the workstation are needed. Improving the rough areas, as noted in the evaluation, and implementing additional commands would be beneficial. Expanding the capabilities of the editor to handle multiple files is necessary. Porting the editor code to other workstations is desirable, including the more sophisticated workstations in the labs (for example, the Sun workstation) and the IBM compatible family of computers since such a large number of potential users own these types of machines.

Enhancing the database interface software is needed. The interface needs to be enhanced by providing batch and background processing. Expanding the interface code to handle the other phases of the software development lifecycle is also required.

The communication interface between the workstations and the central computer needs to be integrated into the system more effectively. A totally integrated system is desirable, and correspondingly more complicated to design.

A significant area of the data dictionary system that has not been addressed to any depth is the database administration environment. The ability to effectively control and manage the database is often discussed (as in this thesis) but has not, to date, been implemented for this or previous versions of the data dictionary system. This

topic needs attention and will provide a great service to the system once it's implemented.

The data dictionary system designed in this thesis needs to be evaluated, in its entirety, by users who employ its services over an extended period of time. Only through this evaluation will the effectiveness and the usefulness of the design be measured.

Appendix A

Evaluation of an Automated/Interactive Software Engineering Tool to Generate Data Dictionaries

This appendix contains a summary and evaluation of the original tool designed to automate the generation of data dictionaries. The comments reflected here are based primarily on the author's personal experience with the system. The original software was implemented by Thomas (39) and later modified and improved by Hamberger (16). This evaluation took place while some of the modifications were being implemented and, hence, may not reflect all of the latest updates to the system.

The System Description section describes the objectives of the tool, the environment the tool was designed to function in, and identifies the target user group of the tool. The Documentation section identifies all available documentation for the tool. The Human-Computer Interface section contains the results of the evaluation performed on the tool. The Conclusion contains a summary of the prevalent strengths and weaknesses of the system.

System Description

The objective of this software tool was to assist program designers generate data dictionaries in support of other software development documentation efforts.

The tool ran on the Vax-11/730 under UNIX with the Berkeley version of the INGRES relational database management system. It was written in the C programming language with the INGRES embedded query language QUEL.

The tool was designed to support multiple phases of the software lifecycle including SADT's, Data Flow Diagrams, Structure Charts, and actual code. Specific requirements for the system were taken from the Software Development Guidelines and Standards (37) published by the AFIT School of Engineering's Department of Electrical and Computer Engineering.

The target user group was students at AFIT pursuing a masters degree in some area of computer systems or engineering. The tool obtained information from the user primarily through direct input from the keyboard with a small set of data obtained through analysis of the existing database.

Specific abilities the tool claimed to perform include:

<u>Functions</u>	<u>Items</u>	<u>Categories</u>
* Input	* Action	SADT
* Retrieve	* Data	Structure Chart
Delete		* Code
* Modify		

Only those items marked with an asterisk have been fully implemented and were available for evaluation.

Documentation

This tool was originally conceived as a class project by a group of graduate students, and later modified and expanded as part of a masters degree thesis effort at AFIT (39). In that respect, the only written documentation for the tool was the thesis itself. Although the thesis did include information on the genesis of the design, the relations in INGRES originally created, and other information generally included in thesis reports, no user manual of any kind was ever written.

The system was advertised as a straight forward "question and answer/fill in the blank" tool. The information requested by the system generally followed the standards and guidelines published by the Electrical and Computer Engineering department and the user was expected to be familiar with these guidelines prior to executing the tool. No on-line help was available.

The source code for the entire project was available (approximately 150 pages). Each module of code had a module header of which there existed no entries. Comments within the code itself were almost nonexistent.

Human-Computer Interface

The human-computer interface was the primary target for evaluation. A host of attributes exist that are used to characterize a "friendly, effective, efficient" human-computer interface. This tool's evaluation was based

on the attributes described below.

Response Time. Response time is generally defined by how long it takes the computer to react to user input (10, 27).

The original implementation of the data dictionary tool ran interactively with INGRES. After a small amount of information was input by the user, processing by the user stopped and processing information into INGRES commenced. Only after the processing of information to INGRES had completed did control return to the user for further input.

Interfacing with INGRES was a time consuming process by itself. Coupled with a user-loaded system, INGRES interface processing time increased dramatically. The result was that the user spent the majority of the time at the terminal waiting for the computer to process information into INGRES. The user got bored, frustrated, and wasted a significant amount of valuable time. Typical time requirements for each interaction with INGRES, in a heavily loaded system, were on the order of minutes.

A recent modification to the tool consolidated all interaction with INGRES to the beginning or end of the editing session. This improvement reduced the numerous frustrating delays at the terminal but the time required to interface with the database did not change.

An additional option the user had, as a result of a recent modification, was the ability to update multiple

existing definitions. The system retrieved from the data base as many definitions as the user requested. The time it took to retrieve them is directly proportional to the number of definitions, but, all of the interaction with INGRES was performed at one time. Once the definitions were retrieved, the tool allows modifications to be made. Only after all the modifications have been made to all requested definitions did the tool interface with the database again to delete the old definitions and restore the new definitions.

Regardless of the time of the interfacing with the database, there remained the requirement for the user to wait for the processing. This slow response time was the most significant drawback to the system.

Methods of Input. There are a variety of methods employed by software systems to generate input from the user (1, 10). The manner in which this tool obtained information from the user was a combination of menu selection and text entry.

The manner in which text entries were made was far from stimulating. It consisted of a routine of typing in single entries or lengthy phrases at the bottom left corner of the screen. Earlier versions of the tool had entries scroll off the top of the screen as new information was entered at the bottom. Recent modifications required one input item per screen with that item disappearing as a new request for

input was presented.

Once the user reached the third or fourth input screen there was no information provided as to which project or module name was being entered. Experience showed that most students who used the system sat at the terminal only when they were prepared to input several definitions into the database. Since data dictionary definitions usually include 10 - 30 lines of information, users sometimes forgot which item they were entering into the database. Only during certain requests for input did the current implementation provide information as to what project and module name was being entered.

Psychological Closure. Breaking lengthy input processes into parts provides the user with positive feedback through a feeling of accomplishment and success. This is what James Martin (27) refers to as psychological closure. The overall lengthy delays in response time seriously detracted from the user's ability to achieve psychological closure.

Generally, lengthy textural descriptions were entered early in the input process. This required the user to spend a large amount of time inputting large textural descriptions before he or she was certain that the information typed actually entered the system. This prohibited the user from achieving that critical initial confidence in the system.

Another problem that detracted from the user's ability

to achieve psychological closure is that the information requested by the system was not necessarily what the user expected. This topic is addressed in more detail in the next section.

One significant improvement, in the latest version of the tool, provided messages on the screen indicating exactly what the computer was doing when information was being sent to and from the database. This provided the user with important feedback and reduced the users' anxieties that frequently accompany empty visual displays.

Expectations From the Tool on Information Requested.

The information requested by the program differed slightly in format and terminology from the published guidelines. Doubt was created in the user's mind when attempting to interpret queries from the program correctly. Doubt was also manifested when required data was omitted from the set of queries when, unbeknownst to the user, this data was extracted directly from the database. This exemplifies the importance of keeping the user informed. On-line help facilities or messages would assist in reducing some of these problems.

Error Checking and Recovery.

Error checking facilities are quite useful and reduce many potential problems when users are expected to type large amounts of input from the keyboard. If the errors can be identified and corrected

before other dependencies are built upon them, then the system will perform with more reliability and efficiency.

Experience has shown that certain symbols on standard keyboards are not accepted by INGRES (for example, square brackets '[']'). When these symbols are entered, INGRES returned an error to the screen and the user was left in doubt as to exactly what information "made it" to the database, if any at all. These types of errors were not screened for by the tool.

When inputting textural type information, the screen indicated with "guides" the maximum length of the input before a RETURN was required. When the user typed beyond these "guides" a variety of things happened, none of which were desirable, and most of which were unrecoverable. An analysis of the original code indicated that no error checking capabilities were included for this particular problem.

Another type of entry considered to be an error was a blank entry entered for a key field (for database purposes). The system readily accepted blank entries which could lead to problems when storing the information in, or attempting to retrieve it from, the database.

A variety of wrong keys were pressed at the various menu displays. The tool correctly handled all standard characters and strings of characters that were entered, in that a message was displayed to the screen and another

opportunity to enter a correct alternative was made available. Escape and control characters, however, caused the system to do strange things -- all of which were undesirable.

Visual Display of Information. The quality of presentation of information can have a substantial impact of the acceptance of the tool by users and can arouse an interest in its function. Any human-machine interface has an inherent objective of being easy to use and stimulating to the user. If the user does not enjoy using the tool, he or she will avoid using it (10).

Overall, the visual display used by this tool was unappealing. The ineffective use of the entire screen, margins and white space, and cursor positioning seriously detracted from the friendliness of the system. For example, a significant annoyance was the entering of information, line-by-line, always beginning at the lower left corner of the screen, with the entire screen scrolling up as each new line was entered.

Favorable Aspects of the Interface. The items observed that were effectively implemented, as determined by this author, are listed below.

1. The interface did follow consistency guidelines for most of the input routines. The user quickly adapted to the methods used to input data and became comfortable in anticipating system prompts.

2. After a complete definition was entered the user had the option to review and modify the entry prior to storing the information in INGRES. This was a major improvement from earlier versions of the tool where once the data item was typed on the keyboard there was no way to modify that information.

3. Although response times were long (as a function of the system, INGRES, and other items) the latest version of the tool provided what comfort it could by printing messages to the screen to keep the user informed as to the status of the processing.

4. Error checking at the Menu selections was implemented and effective in protecting the user from making unforgiving mistakes.

Recommendations for Improving the Interface. The major areas recommended for improvement include the following.

1. The single biggest problem with the system was the long response times. This time must be reduced for the tool to be of any use.

2. It is necessary to keep the user informed on what data dictionary item is being input into the dictionary. The project name and the module or variable name should appear on every screen once they are entered. They should appear in the same place and highlighted in some way so as to be easily recognized.

3. Methods of error checking, and possibly correcting, should be incorporated to reduce the potential for error. This includes, at a minimum, scanning for entries that INGRES does not accept, blank entries for key items, and accidentally typing beyond guides.

4. The visual display of information needs to be redesigned to provide a more active and stimulating environment for the user. "Blank form filling" should be considered for inputting groups of single word entries.

5. To achieve a faster psychological closure in users, a short set of information should be requested by the tool early in the session with

the lengthy algorithms and text descriptions coming later. This will give the user a quicker feeling of confidence in the system.

6. A delete option is offered to the user when it may not be desirable. Casual users should, generally, not have permission to remove definitions from the database -- only modify them as needed. Removal requests should be directed to the database administrator, or instructor as appropriate.

Conclusions

System Strengths. Basically the tool performed one of its main objectives successfully -- that objective being to support the code phase of the software lifecycle.

The system was advertised as a straight forward/fill-in-the-blank tool with no need for any documentation as to how to execute the program. This claim held true for the most part, in that most students had little trouble properly executing and using the tool.

System Weaknesses. The majority of the system's objectives were not achieved. The system performed too slowly and was too cumbersome to use to be of any great value to students. The other major objectives of the tool, that include supporting the other phases of the software lifecycle, have not been successfully implemented. It should be noted that once an effective interface is designed for one phase and the time delays are substantially reduced, then implementing the other phases of the lifecycle are straightforward.

The genesis of this project, it appeared, led to some of the major difficulties in its implementation. A number of students were involved in the original design, with a later thesis effort dedicated to "patching things" to a workable state. In all fairness to the original design, a number of very valuable lessons were learned in the analysis of the problems resulting from its implementation that would never have been learned otherwise. For instance, trying to store all information about every data item or action was found to be a very complex and difficult task. The complexity is especially prevalent when attempting to maintain relationships between items.

The time required for a user to enter information into the database needs to be reduced. This time factor includes time required to enter user information to the system and time required to interface with the database.

Large time delays occurred in entering information when the computer is loaded with users. During normal duty hours the Vax-11/780 computer usually felt this load. One alternative to overcome this problem is to distribute the processing of information over multiple machines. For instance, the user could enter data dictionary entries on a personal computer (PC) or a workstation and download the results of the input to the Vax when it is convenient for the user. Multiple entries can be made on the workstation and the overall time saved would be significant with user

anxiety and frustration levels being substantially reduced.

The second time factor centers on how the information is stored in the actual database. There are three types of internal storage structures that INGRES supports: heap, indexed sequential access method (isam), and hash (10). Generally, heap structures are the least efficient of the three, which just happens to be the type structure used in the database supporting this tool. In heap structures duplicate tuples are not removed and nothing is known about the location of the tuples. As a result, retrieval times are directly proportional to the size of the relation. The other two types of storage structures are more sophisticated and produce more efficient results when queried. These alternative methods need to be tested and evaluated to determine if access times can be reduced.

Closely associated with storage structures used in the database are the methods used to manipulate, or query, that data. Specifically, the two more sophisticated storage structures support more complex relational algebra. In the current implementation of the code very simple relational algebra is used to retrieve information from the database. As the storage structures are modified, so should the relational algebra used to access the data.

Appendix B
User's Manual for the Data Dictionary Editor
on the Zenith Z-100

Starting the Editor Program

1. Turn the power to the computer on (switch on left rear of machine).
2. At the MS-DOS system prompt enter "ddedit" and press RETURN.
3. The editor title slide will appear on the screen and you will be asked to enter the current date. You must enter a valid date before you will be permitted to continue. After entering the date you will begin your magical mystery tour through the land of the data dictionary!!
4. Menus will follow on the next several slides. The first menu provides the options of CREATing or UPDATing a data dictionary definition. If a data definition file is stored on a disk then updating this definition is possible. To create a data definition from scratch CREATE is the proper choice. Traversing between levels of menus (up and down) is provided.
 - a.) CREATE. Should the create option be selected the user will be provided with two follow-on menus, the first asking what phase of the software lifecycle is desired, and the second asking what category of definition within that phase (either an action or data item). The

three phases on the menu are the Requirements Analysis phase (SADT diagrams), Design phase (Structure Charts), or Code phase. As the selections from the menu are made they appear on the top line of the screen to keep the user informed of prior choices. Once the category is selected, the template for the selected category of definition is displayed and the user can begin to edit the definition.

b.) UPDATE. Should the update option be selected, the user will be queried for the name of the file to update. If the file is found and if it is in the correct format for the editor it will be loaded and displayed on the screen prepared for the user to edit as desired. If the file is not found or is in the wrong format for the editor, or some other problem prevents it from being read by the program, the user will be notified and control will be returned to the top level menu.

Editing Data Definitions

Actual editing a data definition is a form filling exercise that requires the user to type data in the designated locations. The editor commands available are described below and listed in Figure B-1. This same list of commands is available on-line by pressing the HELP key while in the actual editor.

Text Input. Characters are placed in the definition at the cursor position as they are pressed on the keyboard. Illegal characters are signaled by a "beep" from the

machine, no action takes place on the screen. The text as it appears on the screen is exactly what will be stored in the definition file.

Cursor Movements. The cursor can be moved by pressing the 4 arrow keys, RETURN, or BACK SPACE. Scrolling the text up or down 16 lines is provided by pressing appropriate control characters or function keys. Movement to the top and bottom of the definition is also provided. the cursor is restricted from moving into unauthorized areas (areas in normal video).

Insert and Delete Lines. The editor provides the facility to insert and delete lines for those attributes of a data definition indicated as having multiple lines or entries in the Software Development Documentation Guidelines and Standards. "Descriptions" of processes or parameters, for example, can contain an unlimited number of lines of text. Additional lines can be inserted by pressing the INS LINE key (unshifted). The DEL LINE key (shifted) will delete the line the cursor is on if it was created by the insert command (the basic layout of the definition template cannot be altered).

Some entries in a data dictionary definition contain a group of lines (2, 3, or 4) such as the "reference" or "alias" group in the definition of a design parameter. This "group" is considered one entry. An additional "group" can be inserted by placing the cursor on the bottom most line of

the group and pressing the INS LINE key (unshifted). A complete group of lines will be inserted immediately below the cursor position. These groups can be deleted by placing the cursor at the top most line of the group and pressing the DEL LINE key (shifted).

Exiting the Editor. EXITing the editor session signifies a desire to save the definition to a file. QUITing the editor session signifies a desire to not save the definition. Both commands are available by pressing the appropriate control or function keys (see Figure B-1). If the SAVE command is chosen you will be presented a menu with 3 options as to how you desire to save the file. These three options include overwriting the existing file, creating a backup copy of the existing file and saving the new version under the original name, and defining a new file name for the definition. The QUIT command will return you to the top level menu inside the tool. (** Note: Database requirements dictate that all definitions must have a non-empty entry for the PROJECT and NAME fields. You will NOT be permitted to save a file if either of these entries in your definitions are empty).

<u>Key</u>	<u>Control</u>	<u>Key</u>	<u>Action</u>
F0	or	^f	exits editor -- with save
F1	or	^q	quits editor -- no save
F3	or	^t	moves cursor to top of buffer
F4	or	^b	moves cursor to bottom of buffer
F6	or	^c	pages screen down 16 lines
F7	or	^r	pages screen up 16 lines
HOME			moves cursor to top left corner of screen
(arrow keys)			moves cursor 1 position in direction of arrow.
BACK SPACE			moves cursor 1 position left
RETURN			moves cursor to next available input field
INS LINE			inserts line (or group of lines) below cursor (multiple lines only)
DEL LINE			deletes line (or group of lines) at cursor (multiple lines only)
HELP			presents this screen

Figure B-1. Available Editor Commands.

Appendix C

Evaluation Questionnaire

Evaluation of the Human-Computer Interface of the Data Dictionary System

The following questionnaire is designed to provide feedback on the human-computer interface of the Data Dictionary system developed for the design phase of the software lifecycle. Through your responses, I hope to measure your degree of satisfaction with the system, with primary emphasis on the "user-friendliness" of the human-computer interface on the microcomputer workstation.

The questionnaire consists of a list of 12 factors. I hope to obtain your reactions and attitudes based on your response to six possible adjective pairs used to describe each factor. Each adjective pair has a seven interval range where you are indicate your feelings. Responses placed in the center of the range will indicate that you have no strong feelings one way or the other, or that you cannot effectively evaluate the given factor.

I would appreciate any specific comments that you would care to make about any of the factors or the system in general.

Start Time : _____

1. System Feedback or Content of the Information Displayed.
The extent to which the system kept you informed about what was going on in the program.

insufficient | _ | _ | _ | _ | _ | _ | sufficient

unclear | _ | _ | _ | _ | _ | _ | clear

useless | _ | _ | _ | _ | _ | _ | useful

bad | _ | _ | _ | _ | _ | _ | good

unsatisfactory | _ | _ | _ | _ | _ | _ | satisfactory

To me this factor is

unimportant | _ | _ | _ | _ | _ | _ | important

Comments:

2. Communication. The methods used to communicate with the tool.

complex |__|__|__|__|__|__| simple

weak |__|__|__|__|__|__| powerful

bad |__|__|__|__|__|__| good

useless |__|__|__|__|__|__| useful

unsatisfactory |__|__|__|__|__|__| satisfactory

To me this factor is

unimportant |__|__|__|__|__|__| important

Comments:

3. Error Prevention. Your perception of how well the system prevented user induced errors.

bad |__|__|__|__|__|__| good

insufficient |__|__|__|__|__|__| sufficient

incomplete |__|__|__|__|__|__| complete

low |__|__|__|__|__|__| high

unsatisfactory |__|__|__|__|__|__| satisfactory

To me this factor is

unimportant |__|__|__|__|__|__| important

Comments:

4. Error Recovery. The extent and ease with which the system allowed you to recover from user induced errors.

unforgiving |__|__|__|__|__|__|__|__| forgiving
incomplete |__|__|__|__|__|__|__|__| complete
complex |__|__|__|__|__|__|__|__| simple
slow |__|__|__|__|__|__|__|__| fast
unsatisfactory |__|__|__|__|__|__|__|__| satisfactory
To me this factor is
unimportant |__|__|__|__|__|__|__|__| important

Comments:

5. Documentation. Your overall perception as to the usefulness of documentation.

useless |__|__|__|__|__|__|__|__| useful
incomplete |__|__|__|__|__|__|__|__| complete
hazy |__|__|__|__|__|__|__|__| clear
insufficient |__|__|__|__|__|__|__|__| sufficient
unsatisfactory |__|__|__|__|__|__|__|__| satisfactory
To me this factor is
unimportant |__|__|__|__|__|__|__|__| important

Comments:

6. Expectations. Your perception as to the services provided by the system based on your expectations.

displeased |__|__|__|__|__|__|__| pleased
low |__|__|__|__|__|__|__| high
uncertain |__|__|__|__|__|__|__| definite
pessimistic |__|__|__|__|__|__|__| optimistic
unsatisfactory |__|__|__|__|__|__|__| satisfactory
To me this factor is
unimportant |__|__|__|__|__|__|__| important

Comments:

7. Confidence in the System. Your feelings of assurance or certainty about the services provided by the system.

low |__|__|__|__|__|__|__| high
weak |__|__|__|__|__|__|__| strong
uncertain |__|__|__|__|__|__|__| definite
bad |__|__|__|__|__|__|__| good
unsatisfactory |__|__|__|__|__|__|__| satisfactory
To me this factor is
unimportant |__|__|__|__|__|__|__| important

Comments:

8. Ease of Learning. Ease with which you were able to learn how to use the system to generate data dictionary definitions.

difficult	_ _ _ _ _ _ _	easy
confusing	_ _ _ _ _ _ _	clear
complex	_ _ _ _ _ _ _	simple
slow	_ _ _ _ _ _ _	fast
unsatisfactory	_ _ _ _ _ _ _	satisfactory
To me this factor is		
unimportant	_ _ _ _ _ _ _	important

Comments:

9. Display of Information. The manner in which both program control and data dictionary information was displayed on the screen.

confusing	_ _ _ _ _ _ _	clear
cluttered	_ _ _ _ _ _ _	well defined
incomplete	_ _ _ _ _ _ _	complete
complex	_ _ _ _ _ _ _	simple
unsatisfactory	_ _ _ _ _ _ _	satisfactory
To me this factor is		
unimportant	_ _ _ _ _ _ _	important

Comments:

10. Feeling of Control. Your ability to direct or control the activities performed by data dictionary editor.

low	_ _ _ _ _ _ _	high
insufficient	_ _ _ _ _ _ _	sufficient
vague	_ _ _ _ _ _ _	precise
weak	_ _ _ _ _ _ _	strong
unsatisfactory	_ _ _ _ _ _ _	satisfactory
To me this factor is		
unimportant	_ _ _ _ _ _ _	important

Comments:

11. Relevancy or System Usefulness. Your perception of how useful the system is as an aid to a software developer.

useless	_ _ _ _ _ _ _	useful
inadequate	_ _ _ _ _ _ _	adequate
hazy	_ _ _ _ _ _ _	clear
insufficient	_ _ _ _ _ _ _	sufficient
unsatisfactory	_ _ _ _ _ _ _	satisfactory
To me this factor is		
unimportant	_ _ _ _ _ _ _	important

Comments:

12. Overall Evaluation of the System. Your overall satisfaction with the system.

unsatisfied |__|__|__|__|__|__| satisfied

Comments on the Overall System:

Finish Time : _____

Total Time Required for Evaluation = _____

Thanks for your help!

Appendix D

Data Dictionary Editor Evaluation Handout

1. You have been asked to evaluate the software interface developed for the Data Dictionary System on the Zenith Z-100 microcomputer. Your feelings, perceptions, and comments are solicited on the survey form provided with this handout.

2. You will be provided with 2 computers (room 242), all necessary software (already on the machine), a set of sample data to input into the system (ref. Software Development Documentation Guidelines and Standards, pp. 26-31), a brief user's manual on how to execute the program, and a survey form containing 12 items to evaluate.

3. It is not necessary that you understand all of the specifics of what the input data means. Your evaluation should be based solely on the human interface aspects of the tool.

4. Sequence of Steps to Follow:

a.) Begin your evaluation by CREATing (entering) the data dictionary definition provide for you. Enter the information EXACTLY as indicated. Press the HELP key to see all available editing commands.

b.) Save the complete definition to a file using the SAVE facility that the tool provides.

c.) UPDATE this definition making any changes you desire.

d.) Execute all options available in the preliminary set of menus (phases other than the design phase will be stubs). Exit the tool and verify that your files were saved as expected. Reenter the tool as necessary to complete your evaluation.

e.) Exit the tool when you feel that you have satisfactorily evaluated the interface.

5. Based on your experiences with the tool, enter your responses on the survey. Please answer all questions to the best of your ability. The results of the survey are analyzed statistically and numerous blank responses may reduce the accuracy of the evaluation.

6. Make a printout of one of your files that you created or updated and turn it in with your survey. The format of file will be analyzed and WILL NOT be in a form readable by you.

Appendix E

Editor SADT Diagrams

This appendix contains the SADT diagrams for the data dictionary editor. A node index is included for the set of diagrams and a text description accompanies each diagram

Node Index

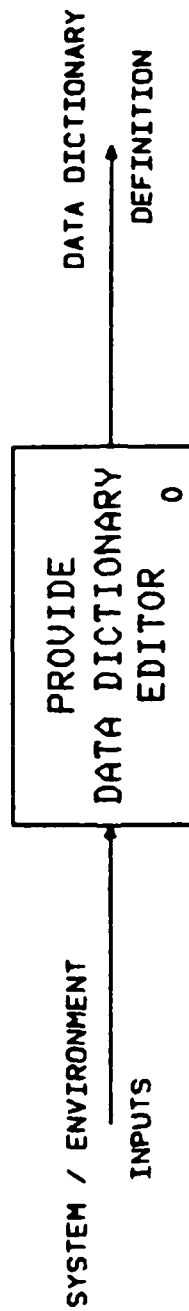
C1		Data Dictionary Editor
C2	0	Provide Data Dictionary Editor
	1	Perform System Configuration
C3	2	Provide Editor
	2.1	Initialize Editor
	2.2	Provide Menu
C4	2.3	Create New Definition
	2.3.1	Determine Phase
	2.3.2	Determine Category
	2.3.3	Load Edit Template
C5	2.4	Update Old Definition
	2.4.1	Determine File to Update
	2.4.2	Load File Into Buffer
	2.4.3	Determine Phase
	2.4.4	Determine Category
	2.5	Exit System
	2.6	Process Errors
C6	2.7	Edit Definition
	2.7.1	Get Keyboard Input
	2.7.2	Evaluate ESC Codes
	2.7.3	Convert ESC Codes to CTRL Codes
	2.7.4	Perform Command

Data Dictionary Editor

Abstract: This diagram shows the overall requirement of the data dictionary editor.

1 Provide Data Dictionary Editor uses the system and environmental inputs to determine the required characteristic files to configure the system software. System/Environmental inputs include the type of computer and terminal being used for the system. The editor is configured and data dictionary definitions are produced.

AUTHOR:	J. FOLEY	DATE:	3-8-86	READER			
PROJECT:	DD EDITOR	REV:		DATE			



NODE:	TITLE:	DATA DICTIONARY EDITOR	NUMBER:	C1
-------	--------	------------------------	---------	----

0 Provide Data Dictionary Editor

Abstract: Provide Data Dictionary Editor uses the system and environmental inputs to determine the required characteristic files to configure the system software. System/Environmental inputs include the type of computer and terminal being used for the system. The editor is configured and data dictionary definitions are produced.

1 Perform System Configuration configures the software to operate based on the hardware descriptions provided.

2 Provide Editor accepts the the configured system and provides an editing environment to produce data dictionary definitions.

AUTHOR: J. FOLEY		DATE: 3-8-86		READER			
PROJECT: DO EDITOR		REV:		DATE			

SYSTEM/ENVIRONMENT

↓

INPUTS

PERFORM SYSTEM
CONFIGURATION

1

↑

CONFIGURED SYSTEM

PROVIDE
EDITOR

2

↑

DATA
DICTIONARY
DEFINITION

MODE: 0	TITLE: PROVIDE DATA DICTIONARY EDITOR	NUMBER: C2
---------	---------------------------------------	------------

2 Provide Editor

Abstract: Provide Editor accepts the the configured system and provides an editing environment to produce data dictionary definitions.

2.1 Initialize Editor draws the initial screens and obtains basic administrative information from the user (such as the date).

2.2 Provide Menu presents the opening menu and obtains the user's desires for the system.

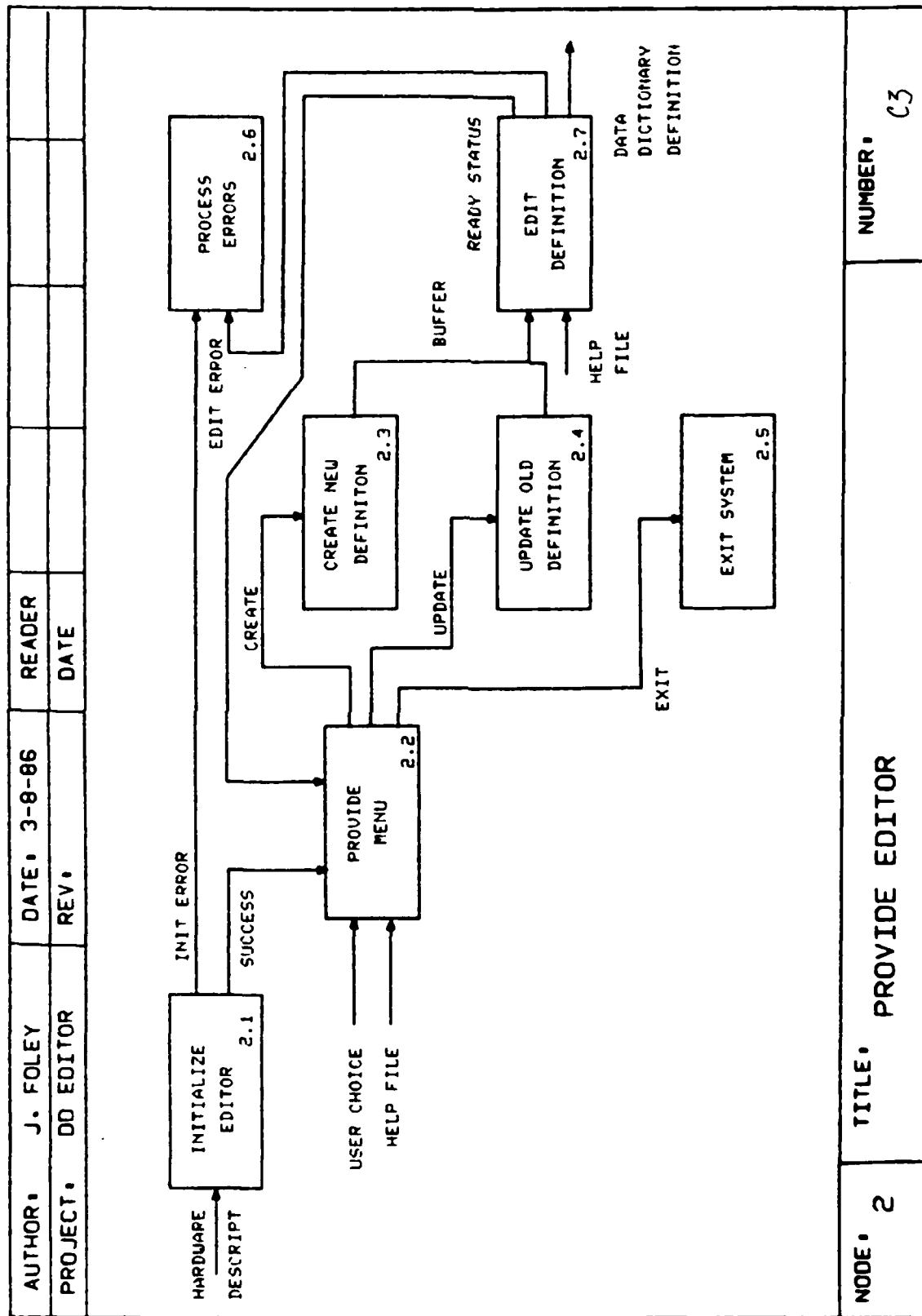
2.3 Create New Definition obtains, from the user, the information required to determine the correct data definition template to load into the buffer.

2.4 Update Old Definition loads the existing file, as designated by the user, into the buffer.

2.5 Exit System enables the user to exit the system.

2.6 Process Errors performs the necessary actions to correct errors, if possible, and provide error information to the user as required. The errors that reach this level are errors that cannot be handled at lower levels.

2.7 Edit Definition provides the editing environment for the user to edit the information inside the buffer.



NODE: 2

TITLE: PROVIDE EDITOR

NUMBER: C3

2.3 Create New Definition

Abstract: Create New Definition obtains, from the user, the information required to determine the correct data definition template to load into the buffer.

2.3.1 Determine Phase obtains the phase of the software lifecycle from the user.

2.3.2 Determine Category Obtains the category of definition, within the particular phase identified previously, from the user (such as process or parameter for the design phase structure chart).

2.3.3 Load Edit Template loads the proper definition template into the buffer.

AUTHOR: J. FOLEY		DATE: 3-8-86		READER	
PROJECT: DD EDITOR		REV:		DATE	

CREATE

DETERMINE PHASE 2.3.1

PHASE

DETERMINE CATEGORY 2.3.2

CATEGORY

LOAD EDIT TEMPLATE 2.3.3

TEMPLATE FILES

BUFFER

PHASE

CATEGORY

LOAD EDIT TEMPLATE 2.3.3

TEMPLATE FILES

BUFFER

NODE: 2.3	TITLE: CREATE NEW DEFINITION	NUMBER: C4
-----------	------------------------------	------------

2.4 Update Old Definition

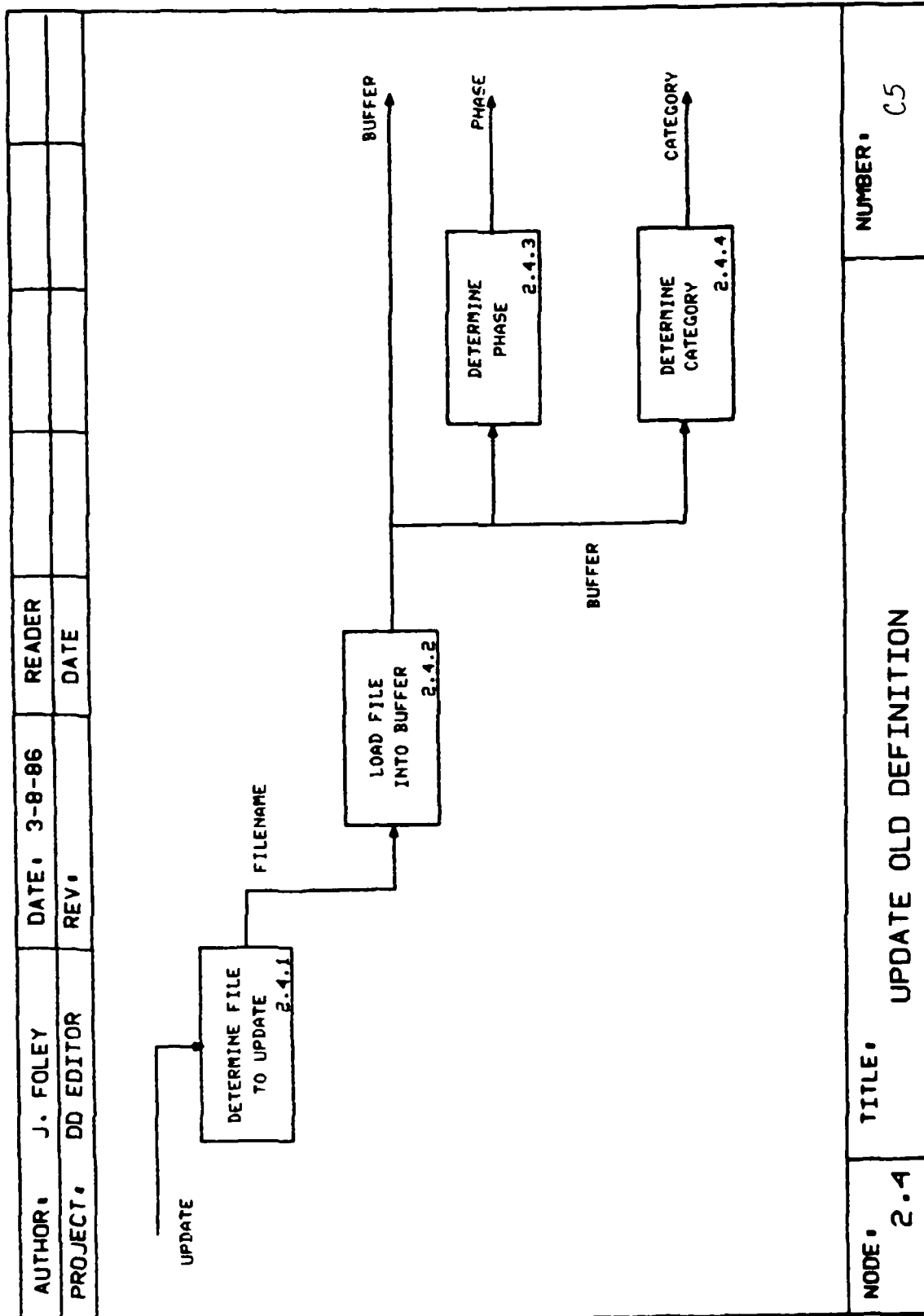
Abstract: Update Old Definition loads the existing file, as designated by the user, into the buffer.

2.4.1 Determine File to Update obtains the name of the file from the user.

2.4.2 Load File into Buffer loads the file designated by the user into the buffer.

2.4.3 Determine Phase obtains the phase of the software lifecycle from the buffer after the definition is loaded.

2.4.4 Determine Category obtains the category of the definition, within the phase of the lifecycle, from the buffer after the definition is loaded.



2.7 Edit Definition

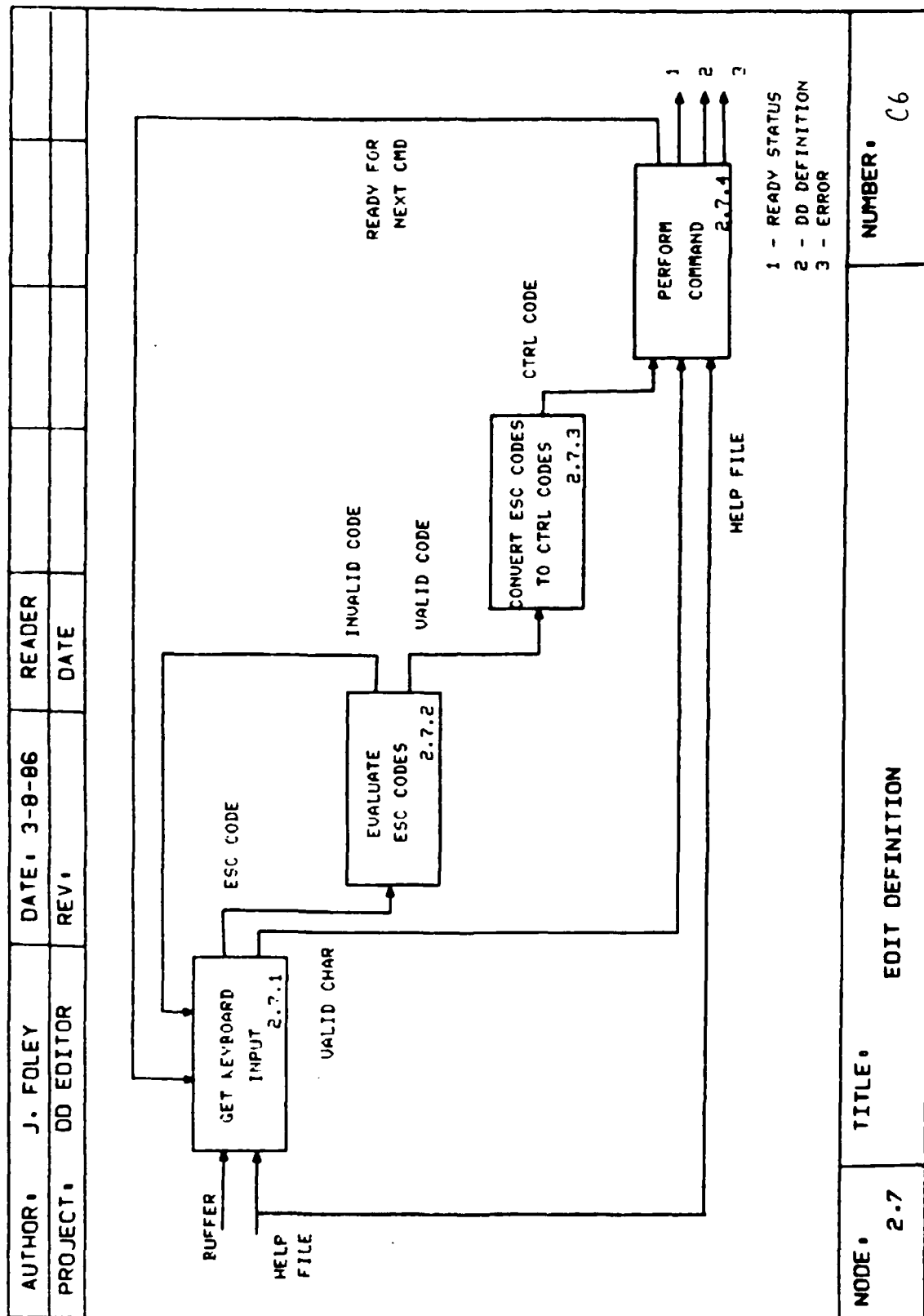
Abstract: Edit Definition provides the editing environment for the user to edit the information inside the buffer.

2.7.1 Get Keyboard Input reads the keyboard for input by the user. Valid characters are distinguished from ESCAPE code and each are sent to different places for processing.

2.7.2 Evaluate ESC Codes analyzes ESCAPE codes as they are entered at the keyboard. Valid codes are processed while invalid codes are flagged returning control back to the keyboard for next input.

2.7.3 Convert ESC Codes to CTRL Codes converts valid ESCAPE codes to CONTROL codes. CONTROL codes can be represented by single ascii codes where ESCAPE codes require more than one ascii code.

2.7.4 Perform Command performs the command as defined by the characters or codes received. Examples of commands would include display the character pressed at the keyboard, insert/delete lines of text, page the screen up or down.

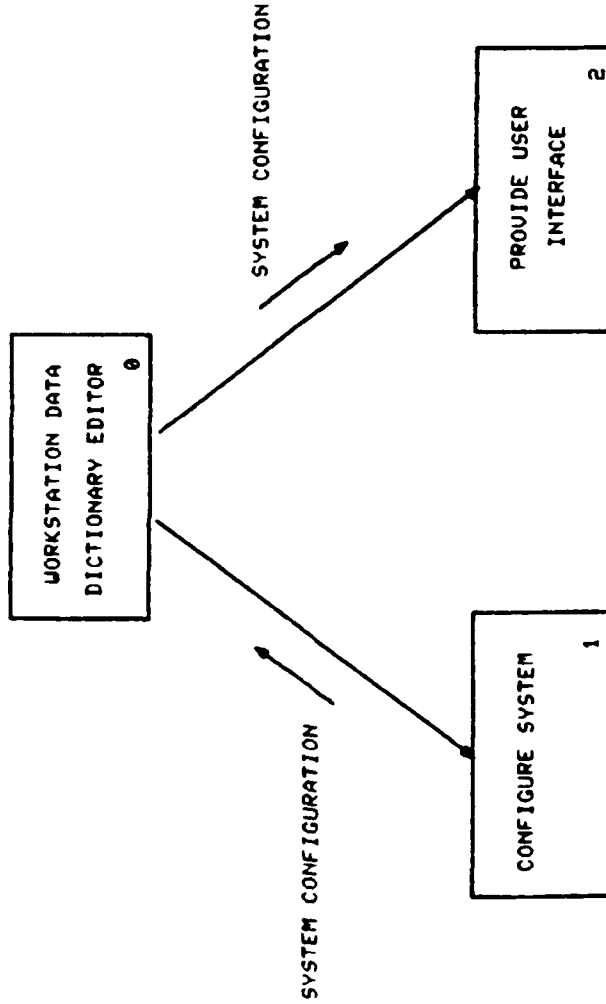


Appendix F

Design Structure Charts for the Editor

This appendix contains the Design Structure Charts for the data dictionary editor.

AUTHOR:	J. FOLEY	DATE:	3 APR 86	READER			
PROJECT:	OD EDITOR	REV:		DATE			



NODE:	0	TITLE:	WORKSTATION DATA DICTIONARY EDITOR	NUMBER:	
-------	---	--------	------------------------------------	---------	--

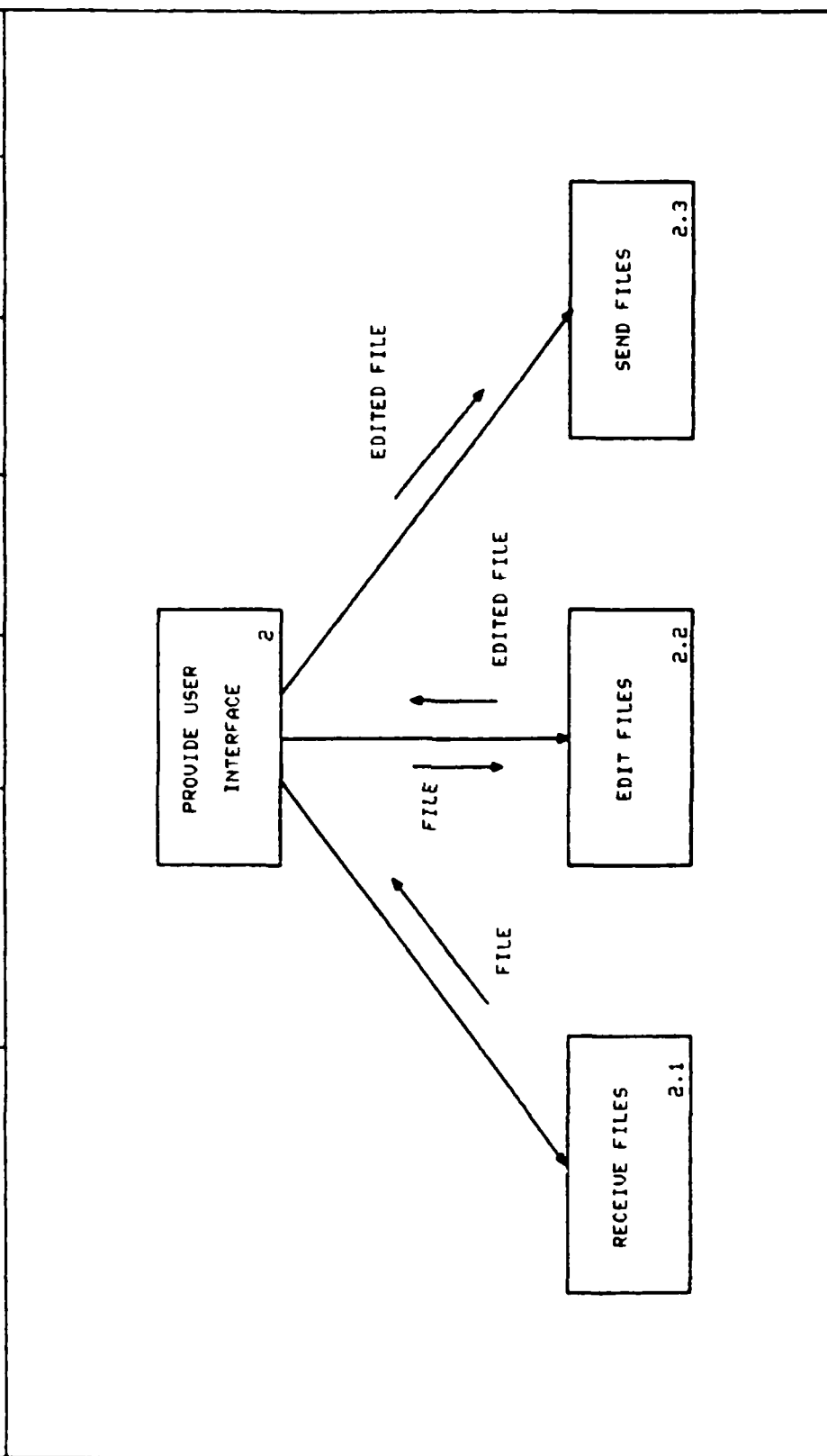
AUTHOR: J. FOLEY		DATE: 3 APR 86		READER			
PROJECT: DD EDITOR		REV:		DATE			

```

graph TD
    A["CONFIGURE SYSTEM  
1"] -- "TERMINAL  
DEFINITION" --> B["DEFINE TERMINAL  
CHARACTERISTICS  
1.1"]
    A -- "KEYBOARD  
DEFINITION" --> C["DEFINE KEYBOARD  
CHARACTERISTICS  
1.2"]
  
```

NODE: 1	TITLE: CONFIGURE SYSTEM	NUMBER:
---------	-------------------------	---------

AUTHOR:	J. FOLEY	DATE:	3 APR 86	READER	
PROJECT:	OD EDITOR	REV:		DATE	



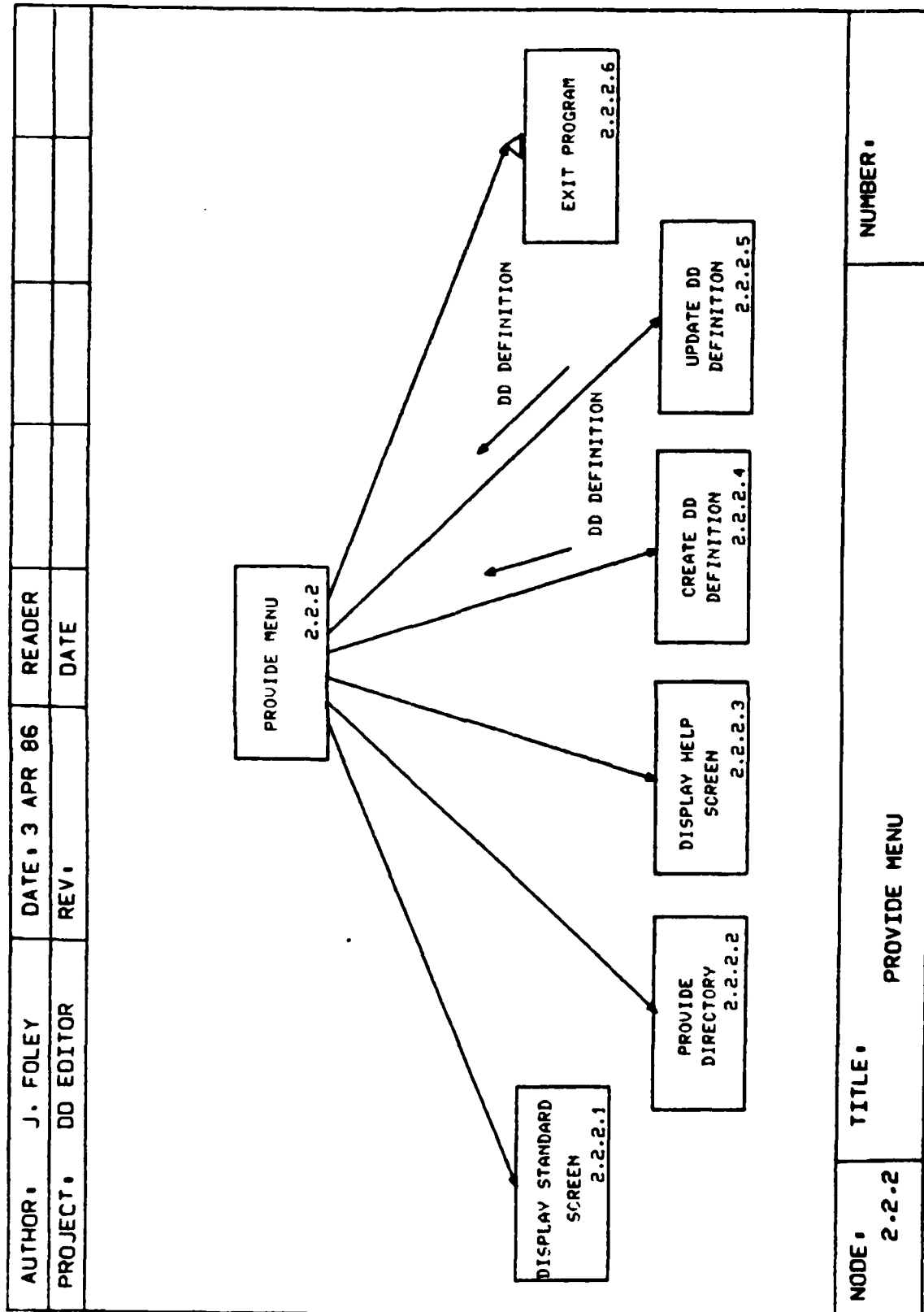
NODE:	2	TITLE:	PROVIDE USER INTERFACE	NUMBER:	
-------	---	--------	------------------------	---------	--

AUTHOR: J. FOLEY		DATE: 3 APR 86		READER			
PROJECT: DD EDITOR		REV:		DATE			

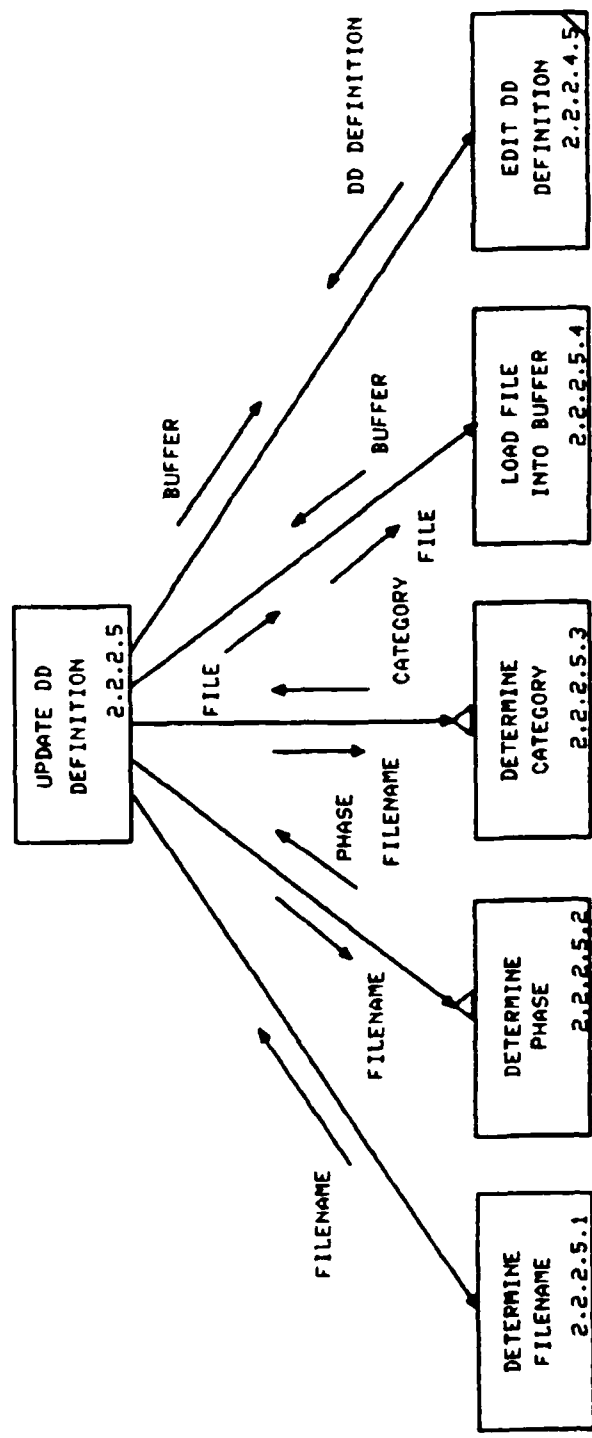
```

graph TD
    A[EDIT FILE 2.2]
    B[DISPLAY WELCOME MESSAGE 2.2.1]
    C[PROVIDE MENU 2.2.2]
    D[DISPLAY ERROR MESSAGES 2.2.3]
    B -- DATE --> A
    C -- DD DEFINITION --> A
    D -- ERROR --> A
  
```

NODE: 2.2	TITLE: EDIT FILE	NUMBER:
-----------	------------------	---------

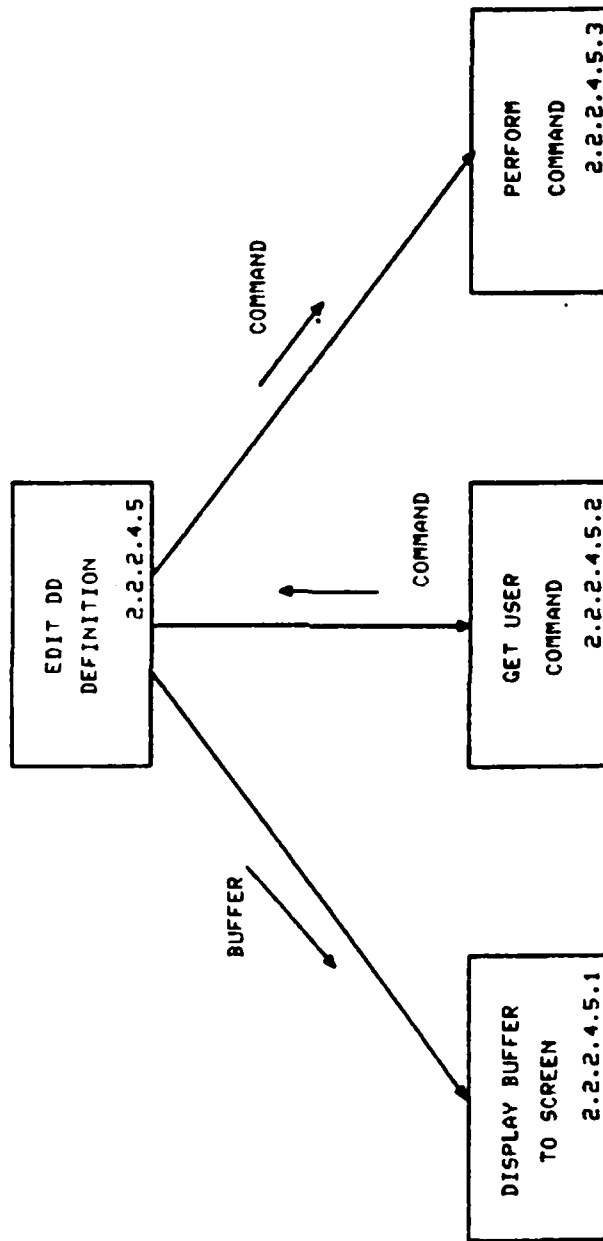


AUTHOR:	J. FOLEY	DATE:	9 APR 86	READER	
PROJECT:	DD EDITOR	REV:		DATE	



NODE:	2.2.2.5	TITLE:	UPDATE DD DEFINITION	NUMBER:	
-------	---------	--------	----------------------	---------	--

AUTHOR:	J. FOLEY	DATE:	9 APR 86	READER	
PROJECT:	DD EDITOR	REV:		DATE	



NODE:	TITLE:	NUMBER:
2.2.2.4.5	EDIT DD DEFINITION	

Appendix G
Summary Paper

Introduction

The Department of Electrical and Computer Engineering at the Air Force Institute of Technology (AFIT) sponsors a large amount of research in the area of software development. In conjunction with this research, the department has established documentation standards that include data dictionaries to support the requirements, design, and coding phases of the software lifecycle (37). Originally these data dictionaries were created and managed by hand. As the dictionaries grew in size, the effective control and management of their contents became increasingly difficult (16). Several efforts at AFIT led to the development of a limited data dictionary system implemented on a Vax-11/780 computer under the Unix operating system, and using the Berkeley version of the relational database management system INGRES.

The overall objective of this research was to expand and enhance the current data dictionary system to support a distributed development environment. The distributed environment is required to support the capability to generate and update data dictionary definitions at the workstation and transmit these definitions between the

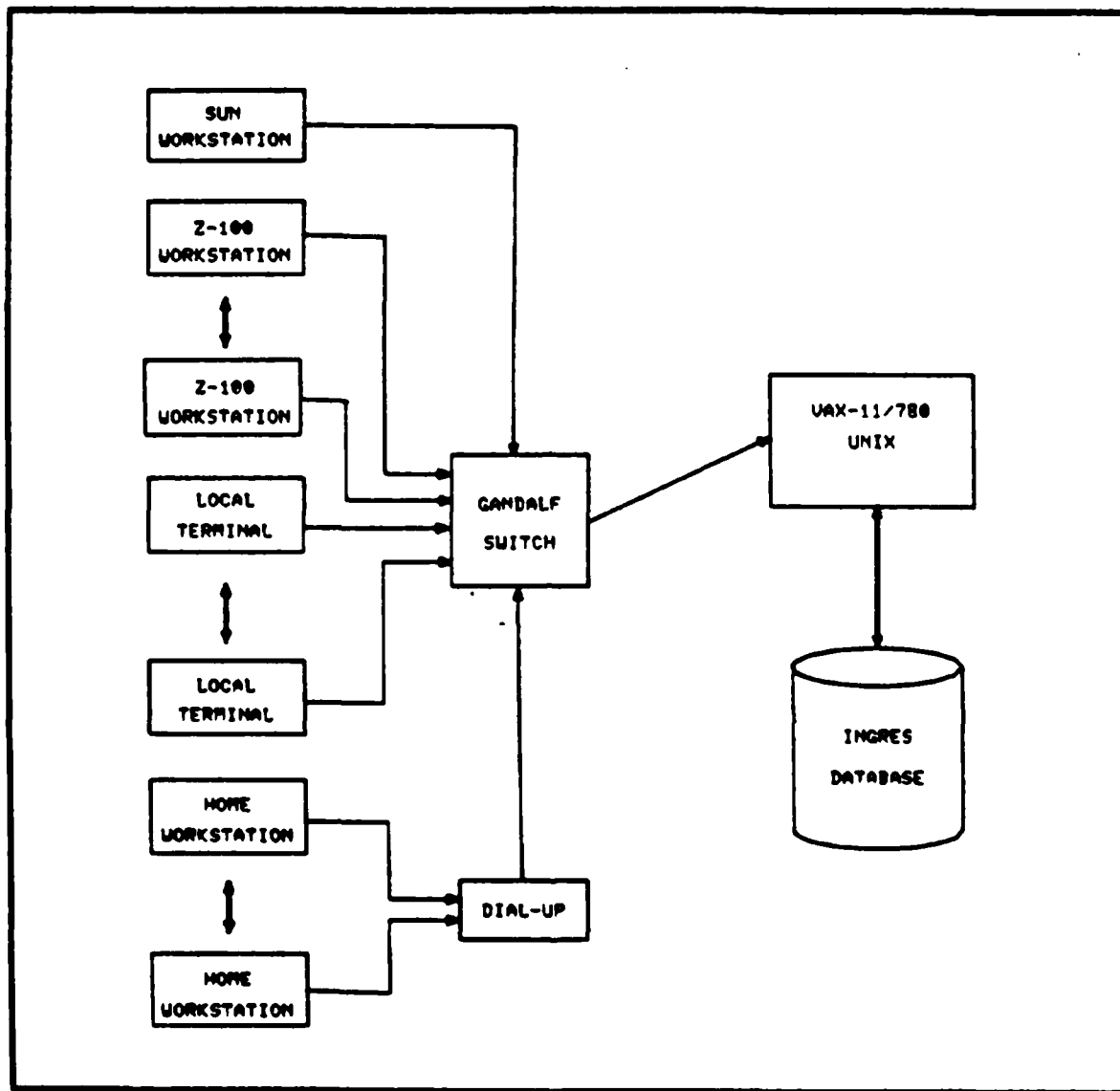


Figure G-1. Distributed Development Environment.

central computer.

The target user group consists of graduate students at AFIT pursuing a curriculum in electrical or computer engineering. All users will have some experience with computers and will be familiar with the requirements document for data dictionary information. Users will be familiar with the Unix Vax-11/780 (SSC) computer and will most likely have some experience with the use of microcomputers.

Software development documentation standards at AFIT follow the software lifecycle phases consisting of the requirements analysis phase, the preliminary and detailed design phases, and the implementation phase. Graphical representations often used to support the documentation requirements include Structured Analysis Design Technique (SADT) diagrams (SADT is a registered trademark of SofTec) for the requirements analysis phase, and Structure Charts (SC) and Data Flow Diagrams for the design phase. Data dictionaries often accompany these representations in addition to accompanying the actual code (36). The contents of data dictionary definitions for the phases of the lifecycle stated above are detailed in the Department of Electrical and Computer Engineering Software Development Documentation Guidelines and Standards (37). Sample definitions are shown in Figures G-2 and G-3 for a structure chart process and parameter in the design phase. This is the

```

PRNAME: Process Message
PROJECT: NETOS-ISO
NUMBER: 4.0.1
DESCRIPTION: Process a NETOS message.
INPUT DATA: msgptr
INPUT FLAGS: none
OUTPUT DATA: none
OUTPUT FLAGS: error2
ALIASES: PROC_MSG
CALLING PROCESSES: Process Messages and Data
PROCESSES CALLED: Decompose Message
                  Process Network 4 Messages
                  Determine Channel Number
                  Build Queue Buffer for Qty = 1
                  Put Buffer in Queue
                  Level 4 Cleanup

ALGORITHM:
    Decompose message.
    If network message
        Process Network 4 Messages
    else
        Determine channel number
        Build queue buffer
        Put buffer in queue
    Cleanup Level 4.

REFERENCE: PROCESS SPOOLER MESSAGE
REFERENCE TYPE: SADT
REFERENCE: Smith's Algorithm, p. 23-24
REFERENCE TYPE: text
VERSION: 1.1
VERSION CHANGES: Added module "Level 4 Cleanup"
DATE: 11/25/85
AUTHOR: J. W. Foley

```

Figure G-2. Data Dictionary Format for Structure Chart-Process (37:27).

```

PANAME: mess_parts
PROJECT: NETOS-ISO
DESCRIPTION: Decompose parameters.
DATA TYPE: Composite
MIN VALUE: none
MAX VALUE: none
RANGE: none
VALUES: none
PART OF: none
COMPOSED OF: SRC
              DST
              SPN
              DPN
              USE
              QTY
              Buffer
ALIAS: Message Parts
WHERE USED: Passed from Decompose Message to Val Parts
COMMENT: Part of earlier design
REFERENCES: SADT - MSG_PARTS
REFERENCE TYPE: SADT
VERSION: 1.2
DATE: 11/05/85
AUTHOR: J. W. Foley
VERSION COMMENT: USE added to allow network msgs.

CALLING PROCESS: Process Message
PROCESS CALLED: Decompose Message(parts_list)
DIRECTION: up
I/O PARAMETER NAME: parts_list

```

Figure G-3. Data Dictionary Format for Structure Chart-Parameter (37:29-30).

data that is represented in the relational database. The third normal form relations shown in Figures G-4 and G-5 were designed to support the operational data requirements of the design phase of the lifecycle.

process:	
*project	c12 - Project name
*prname	c25 - Process name
number	c20 - Process number
prdesc:	
*project	c12
*prname	c25
*line	i2 - Description line number
description	c60 - Description text
pralg:	
*project	c12
*prname	c25
*line	i2 - Algorithm line number
algorithm	c60 - Algorithm text
processio:	
*project	c12
*prname	c25
*paname	c25 - Name of i/o parameter
direction	c4 - Input "in"/output "out"
pctype	c4 - "data" or "flag"
prcall:	
*project	c12
*prcalling	c25 - Calling process name
*prcalled	c25 - Process called name
prreference:	
*project	c12
*prname	c25
*reference	c60 - Reference description
reftype	c25 - Reference type
pralias:	
*project	c12
*prname	c25
*aliasname	c25 - Name of alias for prname
comment	c60 - Why alias is needed
prhistory:	
*project	c12
*prname	c25
*version	c10 - Version number of this entry
date	c8 - Date of this entry
author	c20 - Author of this entry
comment	c60 - Changes from last version

Figure G-4. Third Normal Form Relations for a Design Structure Chart Process.

parameter:	
*project	c12 - Project name
*paname	c25 - Parameter name
datatype	c25 - Language independent data type
low	c15 - Lowest value allowed, if any
high	c15 - Highest value allowed, if any
span	c60 - Range of allowed values, if any
padesc:	
*project	c12
*paname	c25
*line	i2
description	c60 - Parameter description
pavalueset:	
*project	c12
*paname	c25
*value	c15 - An allowable value for paname
pahierarchy:	
*project	c12
*hipaname	c25 - Name of composite parameter
*lopaname	c25 - Name of component parameter
paref:	
*project	c12
*paname	c25
*reference	c60 - Reference description
reftype	c25 - Reference type
paalias:	
*project	c12
*paname	c25
*aliasname	c25 - Name of alias for paname
comment	c60 - Why this alias is needed
whereused	c25 - Process name where found
pahistory:	
*project	c12
*paname	c25
*version	c10
date	c8
author	c20
comment	c60 - Changes from last version
papassed:	
*project	c12
*paname	c25
*prcalling	c25 - Calling process name
*prcalled	c25 - Called process name
direction	c4 - Direction "up" or "down"
iopaname	c25 - Name of i/o parameter

Figure G-5. Third Normal Form Relations for a Design Structure Chart Parameter.

System Design

Data Dictionary Editor Design. The Zenith Z-100 microcomputer was chosen as the prototype workstation for the implementation of the data dictionary editor for two reasons. First, it supported the standard 16-bit operating system in use today by microcomputers (MS-DOS). Second, there are a large number of Z-100 computers available for student use in the academic areas.

The "C" programming language was the language chosen for this editor. This was because the Berkeley version of the database management system INGRES, used on the mainframe, only supports "C" in accessing the database through its embedded query language, and the fact that "C" was supported by the majority of the other computers available in the labs, facilitating the portability of the code.

User-Machine Interface. The user-machine interface dialogue consists of a screen-oriented combination of menu-selection and form-filling displays. Menu-selection is used in the initial stages of the tool while form-filling dominates the editing session of the tool. The initial stages in the execution of the editor require the user to identify his (or her) desires of the tool. The initial menu is shown in Figure G-6. Definitions can be created from scratch or updated from an existing file.

The form-filling method was chosen as the primary

<u>PLACE CURSOR IN BOX, PRESS <RETURN></u> [] CREATE NEW DEFINITION [] UPDATE EXISTING DEFINITION [] EXIT PROGRAM			

Figure G-6. Opening Menu on the Editor.

method for display and input in the editing portion of the tool. Predefined, formatted structures, called templates, are used as the basic form. There were three reasons for this choice. First, the nature of the data dictionary definition lent itself very well to a blank form-filling operation, since a variety of fields exist for each definition. Second, the form-filling method presented the

MENU 01 CHOICE	MENU 02 CHOICE	MENU 03 CHOICE	FILENAME
RESTRICTED AREA		USER WORK AREA	

Figure G-7. Sample Screen Display.

the edit session, the screen is divided into two parts, the left one-fourth of the screen and the right three-fourths of the screen. The left portion is reserved for field names only (title), while the right portion of the screen is reserved for user input (data). While editing a file, the cursor is restricted from moving into the left portion of

information on the screen in a manner that closely resembled the format required by the department. Third, form-filling provided a "highly disciplined mode of modification that guaranteed the structural integrity [of the data dictionary definition]" (40:102). Because movement of the cursor into unauthorized areas is prohibited, movement between fields is easily accomplished and input error checking capability is enhanced.

The actual display of information on the screen plays an important role in the design of user-friendly interfaces (1, 10, 18, 27). The screen display shown in Figure G-7 was designed for this tool. All menus are displayed consistently throughout the tool with regard to their location on the screen, appearance, and method of choice selection.

The top line of information provides the user with the location within the hierarchical structure of the program, and remains on the screen throughout the use of the tool. The results of each menu selection are provided in this line as the user traverses through the tool. This facility eliminates the requirement for the user to memorize this information.

The remainder of the screen displays the menus. General editing takes place in this area as well. While in

the screen. All input areas are highlighted in reverse video, clearly indicating the maximum length of each entry. The restrictions on the movement of the cursor (outside of any reverse video block) reduce the potential for error. These restrictions are designed to permit the user to concentrate on what information goes into each block rather than the mechanics involved in getting to the proper location on the screen.

The information displayed on the screen is exactly what is stored in the data file. This system is commonly known as "what you see is what you get." This provides immediate feedback to the user that the tool is doing useful work. This contributes to providing the user with the psychological closure that is important in the design of any user-friendly system (27).

It should be apparent that no data dictionary definition is going to fit on the screen in its entirety. A system had to be developed that would control what portion (or window) of the definition buffer would be displayed on the screen and how windows of information could be moved on and off the screen. This is a basic requirement of screen-oriented editors.

The system designed and implemented in this tool involves a "windowing scheme" (see Figure G-8). Global pointers are defined to keep track of the top of the buffer (topbuffer), the bottom of the buffer (botbuffer), the top

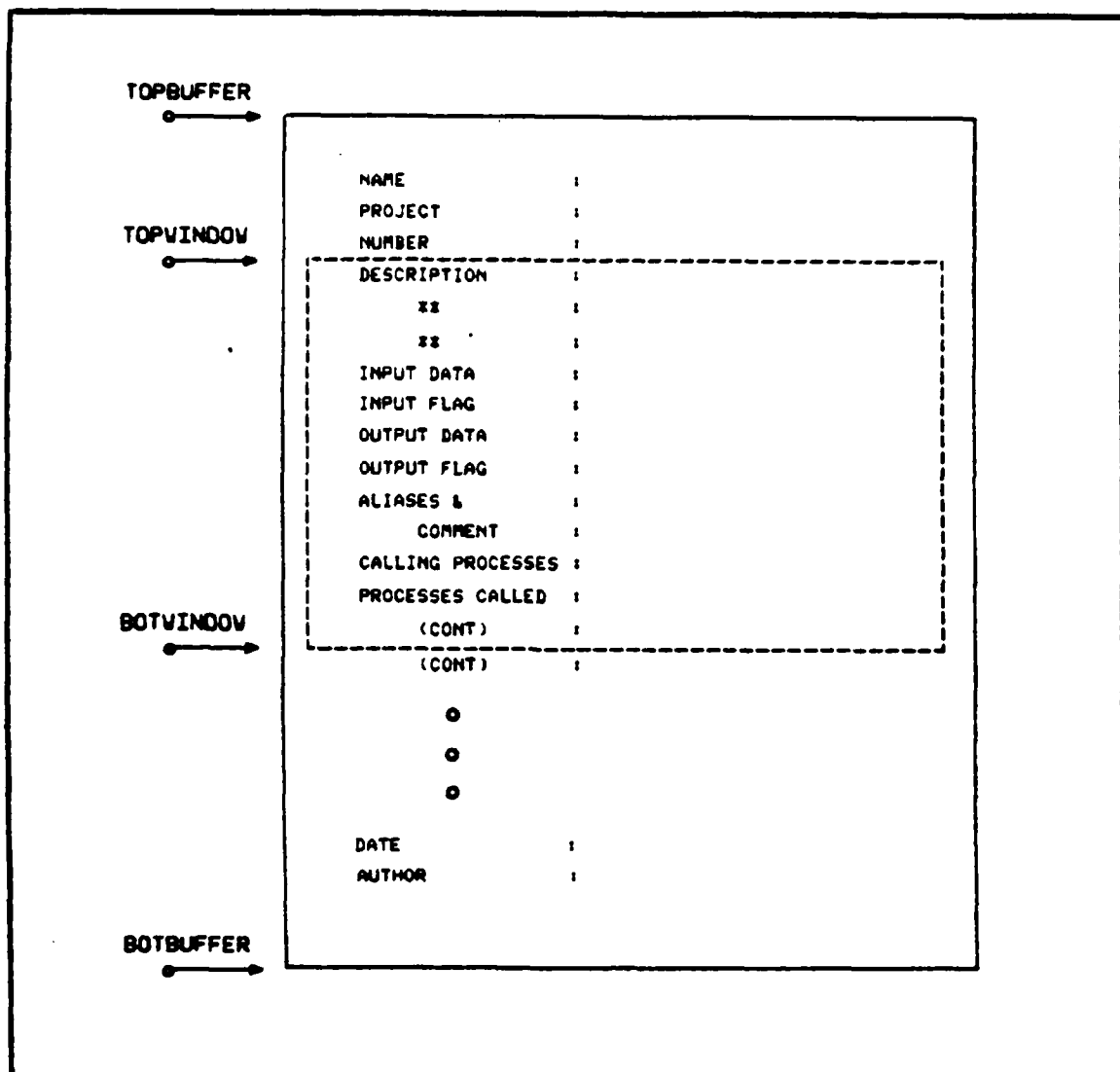


Figure G-8. Screen Editor "Windowing Scheme".

of the window (topwindow), and the bottom of the window (botwindow). Topbuffer points to the very first structure in the linked list, while botbuffer points to the very last structure in the linked list. These two pointers generally do not move once initially established, unless the structure to which either points to can be added onto or deleted. The topwindow and botwindow pointers, on the other hand, move about the buffer frequently always remaining 21 items apart (the size of available screen space for this editor is 21 lines). Initially, when entering the editor, the topwindow is set to topbuffer and the top 21 items in the buffer are displayed. As the user attempts to move the cursor down off the bottom of the screen, the topwindow and botwindow pointers are adjusted accordingly and the new window is presented on the screen. The window pointers will change frequently during an average editing session with the obvious restrictions of the topwindow pointer never "passing" the topbuffer pointer and likewise with the botwindow and botbuffer pointers.

Ideally, the method for handling data files used by the tool is to hide the implementation details from the user completely. The user would be required to know only the names of the processes or parameters, with all file I/O being based on this information. The current implementation requires the user to define filenames and keep track of them as necessary. Several standard options were provided for

storing files, for example overwriting the existing file, or creating a backup version of the old file. Although not ideal, this method of file handling does provide a working system that appears clean and simple to use, yet is flexible and powerful enough to provide most desirable alternatives for the user.

The commands available to a user while using the editor clearly have a major impact on the usefulness of the tool.

"Concern for human engineering dominates the design"

(19:163). The minimum set of commands that will provide a working system include "insert" characters, "delete" characters, and "save" the text at the completion of the edit session.

The capability of moving the cursor freely to any portion of the input area is important. The user must have this capability to view or modify any portion of the file at any time during the session. Cursor movements by character, line, or screen-at-a-time were considered essential for the editor to be of valuable use.

Line insertion and deletion were required to support the various entries that contained multiple lines of data.

Standard termination commands are included to provide for quitting the editing session (without saving the file) and exiting the session (saving the file).

On-line help facilities were considered essential to give users immediate access to all available editor

commands.

Error handling is provided at the lowest possible level in the code. Illegal strings of characters for filenames and dates are identified upon their entry. Illegal characters from the keyboard are suppressed at the time the key is pressed. Other errors that are checked by the tool are those that relate to the definition requirements and not the actual codes from the keyboard, for example verifying that key fields (required by the database system) are non-empty.

The form-filling construct of the editor itself prevented a number of otherwise common errors from occurring, for example entering characters in the wrong location or typing characters beyond the length of the field. Input fields were clearly identified and cursor movement is limited to only input areas. This was one of the major reasons that the form-filling format was the method of choice for this editor.

Database Interface Design. The interface software currently supports the design phase database (for structure charts). As the other databases are designed, extension of the interface software to support them should be straightforward.

The database interface software was designed to translate a text file containing a data dictionary definition, generated by the workstation editor, into the database and vice-versa. Currently, the database interface

software provides operations on single files (data dictionary definitions) only.

Implementation Issues

Some costs were associated with the efforts to keep the code portable to other computers. Designing the code for a 24 line by 80 column screen eliminated the option to employ the 25th line available on the Z-100 terminal. This line is not affected by line insertion and deletion or any other scrolling operations. Because this line was deemed not available, redrawing the screen was necessary when attempts were made to move the cursor beyond the top or bottom of the screen. Paging commands that scrolled up and down 16 lines in the buffer and quick movements to the top and bottom of the buffer were provided to minimize the need for redrawing the screen.

The configuration of the buffer structure used in the editor code provides a facility to handle a variety of data dictionary definitions. Any number of different templates can be created and used with this editor with a minimum number of restrictions. This was demonstrated as templates for the code phase definitions were integrated into the system after the original code was completed using the design phase templates only.

Measuring User Satisfaction

Measuring a user's satisfaction with any degree of

statistical proof of validity has proven difficult. Bailey and Pearson (1983) performed extensive research on the subject of measuring user satisfaction with information systems (2). A formula for defining user satisfaction as "the sum of the user's weighted reactions to a set of factors" (3:531),

$$S_i = \sum_{j=1}^n R_{ij} W_{ij}$$

was developed by Bailey and Pearson. In it, R is the reaction to factor j by individual i, and W is the importance of factor j to individual i. This equation suggests that one's opinion of satisfaction is the sum of his or her positive and negative reactions to a particular set of factors.

To measure a user's perceptions of these factors, Bailey and Pearson used the semantic differential technique (3:533). Four bipolar adjective pairs ranging from negative to positive feelings were identified for each factor. Additional scales were included to test the internal consistency and validity of the four pairs, and to obtain a value for the weights users assigned to each factor. A seven-interval scale was adopted to measure the user's satisfaction with each pair. Figure G-9 shows a sample question (or factor) that was taken from the actual questionnaire.

Error Recovery. The extent and ease with which the system allowed you to recover from user induced errors.

unforgiving | _ | _ | _ | _ | _ | _ | _ | forgiving

incomplete | _ | _ | _ | _ | _ | _ | _ | complete

complex | _ | _ | _ | _ | _ | _ | _ | simple

slow | _ | _ | _ | _ | _ | _ | _ | fast

unsatisfactory | _ | _ | _ | _ | _ | _ | _ | satisfactory

To me this factor is

unimportant | _ | _ | _ | _ | _ | _ | _ | important

Comments:

Figure G-9. Sample Survey Question.

Numerical values were assigned to each of the seven intervals ranging from -3 to +3. The importance scale was assigned values from 0.10 to 1.00 in increments of 0.15. The higher the value the more important the factor. The overall satisfaction was measured by

$$S_i = \sum_{j=1}^{39} \frac{W_{ij}}{4} \sum_{k=1}^4 I_{i,j,k}$$

where W = the weight assigned to factor j by user i , and I = the numeric response of user i to adjective pair k of factor j .

The results of this equation can be deceiving if a user rates half of the factors very high and very important while rating the remainder neutral and very unimportant. The

numerical result would be approximately one-half of the total possible indicating a moderate degree of satisfaction when in fact the user was extremely satisfied. Normalizing the scores and filtering factors whose top four adjectives pairs were all rated at 0 eliminated this problem. A normalized score ranges from -1.00 to +1.00 with the translation of scores shown in Figure G-10.

Many factors identified by Bailey and Pearson did not apply to the evaluation of the data dictionary editor and were not included in its evaluation. Examples of these were vendor support and management factors. Mallery, in a previous thesis effort (25), added several factors to the list that provided greater emphasis on the human-computer interface aspect of software tools. The list was extended by one more factor, for this evaluation, that covered the area of error prevention.

Evaluation of the Data Dictionary Editor

A graduate level software engineering class containing students studying computer systems or computer engineering were targeted for the evaluation. Although they were not knowledgeable in database management systems and were currently learning about the software development lifecycle and data dictionaries, it was determined that they would

<u>Normalized Score</u>	<u>Translation</u>
+1.00	maximally satisfied
+0.67	quite satisfied
+0.33	slightly satisfied
0.0	neither satisfied or dissatisfied
-0.33	slightly dissatisfied
-0.67	quite dissatisfied
-1.00	maximally dissatisfied

Figure G-10. Score Boundaries for Normalized User Satisfaction (3:535).

still provide valuable feedback in an evaluation of the human-interface, or user-friendliness, aspects of the data dictionary editor.

The class was briefed by the author on the overall data dictionary system and the editor's place within the system. Each student was provided with a copy of the questionnaire, a users' manual for the editor and a set of instructions outlining the steps necessary to execute the tool and obtain a hard copy of the data definition file. Two Z-100 computers were set aside in one of the school labs for their use.

The completed surveys were returned one week after they were issued. The average time to complete the evaluation was 37 minutes. The results were analyzed with the normalized scores for each user shown below in Figure G-11.

<u>User Number</u>	<u>Normalized Score</u>
19	0.977
8	0.814
18	0.704
5	0.680
2	0.662
7	0.660
13	0.602
17	0.593
6	0.593
4	0.586
11	0.577
3	0.546
9	0.520
15	0.517
21	0.493
1	0.478
23	0.466
12	0.430
22	0.395
16	0.382
10	0.351
14	0.210
20	0.140
mean for all users = 0.538	

Figure G-11. Normalized Values Overall Satisfaction by User.

The overall normalized measured satisfaction mean of 0.538 generated by the formula presented, coupled with the user assessed satisfaction mean of 1.91 from question #12 (not shown), and the fact that there were no negative mean scores for any single factor indicate that the data dictionary editor was well received by the users. Problems with the editor were brought out, however, in the comments. Over half of the users commented on their frustration at the

editor's need to redraw the screen upon attempts to move the cursor beyond the top or bottom of the screen. Enhancements are certainly in order to make the environment better, particularly in correcting this deficiency.

Conclusions

A data dictionary system was designed to support the AFIT distributed software development environment. The system was broken down into three separate but related subsystems; the workstation, the communications links between computers, and the central computer.

A special data dictionary editor was designed and implemented on a prototype microcomputer workstation. It was designed under the constraints of being generic and portable to other workstations. The generic objective was achieved, as the editor demonstrated its ability to handle code phase definitions after design phase definitions were used for the original implementation and testing of the software.

Twenty-three users evaluated the user-friendliness of the editor and were satisfied with its implementation. This level of satisfaction was based on a normalized mean score of 0.538 on a scale of -1.0 to +1.0 where the higher the score the more satisfied the user was with the system. The new interface is believed to be a major improvement based on the enhanced features, such as screen oriented editing and error checking. Editing time should be reduced with

increased accuracy. Data definitions templates for particular projects can be created by the user to provide an even more efficient system.

The communication links of the system were not addressed in detail, because communication software was already available for use and did provide all the requirements for this first implementation of the system.

The database interface software was designed and implemented on the Vax-11/780 computer under the Unix operating system. The Berkeley version of the INGRES database management system was the host database for the data dictionary definitions.

Bibliography

1. Allen, Craig M. and William A. Ahroon. "Data Entry Displays," IEEE Phoenix Conference on Computers and Communication. 547-551, 1983.
2. Aztec C User's Manual. Manx Software Systems. Inc., Schrewsbury, NJ (1984).
3. Bailey, J. E. and Sammy W. Pearson. "Development of a Tool for Measuring and Analyzing Computer User Satisfaction," Management Science, 29: 530-544 (May 1983).
4. Boehm, Barry. "The High Cost of Software," Practical Strategies for Developing Large Software Systems. Ed. Ellis Horowitz, Reading, Mass: Addison-Wesley, 1975.
5. Choy, David M., Roger J. Bamford, and Frank C. Tung, "A Database Management System for Office Systems and Advanced Workstations," ACM SIGOA Newsletter, 5: 17-18 (Fall 1984).
6. Crenshaw, Jack W. "Towards a 'Friendly' Environment," IEEE Phoenix Conference on Computers and Communication 527-533, 1983.
7. Date, C. J., An Introduction to Database Systems. (Third Edition) Massachusetts: Addison-Wesley Publishing Company, 1982.
8. -----, C. J. "Database Usability," Proceedings of Annual Meeting - ACM SIGMOD, 13: 1 (May 1983).
9. Davis, Richard M. Thesis Projects in Science and Engineering, St. Martins: New York, 1980.
10. Department of the Army. Human Engineering Guidelines for Management Information Systems. Alexandria, Virginia: HQ US Army Material Development and Readiness Command, 9 June 1983.
11. Epstein, Robert. "Creating and Maintaining a Database Using INGRES," Memorandum No. ERL M77-71, Electronics Research Laboratory, College of Engineering, University of California, Berkeley (December 1977).
12. -----, "A Tutorial On INGRES," Memorandum No. ERL M77-25, Electronics Research Laboratory, College of Engineering, University of California, Berkeley (December 1977).

13. Frankosky, Richard J. "Software Interface Ease of Use: Metrics and Methods," IEEE Phoenix Conference on Computers and Communication. 534-537. IEEE Press, New York, 1983.
14. Hansen, Wilfred J. "User Engineering Principles for Interactive Systems", Interactive Programming Environments, edited by David R. Barstow et al., New York: McGraw-Hill, p. 217-231, 1984.
15. Hartrum, Thomas C. Professor of Electrical Engineering. Lecture materials distributed in MA 7.46, Advanced Database Management Systems. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 1985.
16. Hartrum, Thomas C. and Charles W. Hamberger, Jr, "Development of a Distributed Data Dictionary System for Software Development," IEEE NAECON (May 1985).
17. Heathkit Manual for the Video Display Terminal Model H-29. Heath Company, 1983.
18. Heckel, Paul. The Elements of Friendly Software Design, Warner Books: New York, 1984.
19. Kernighan, Brian W. and P. J. Plauger. Software Tools Addison-Wesley: Reading, MA, 1976.
20. Kernighan, Brian W. and Dennis M. Ritchie. The C Programming Language. Prentice-Hall: Englewood Cliffs, NJ, 1978.
21. Kockhan, Stephen G. Programming in C. Hayden: Hasbrouck Heights, NJ, 1983.
22. Lefkovitz, Henry C. Data Dictionary Systems, Forward by Edgar H. Sibley. Wellisley: Q.E.D. Information Sciences Inc., 1980.
23. Leong-Hong, Belkis W. and Bernard K Plagman. Data Dictionary/Directory Systems Administration, Implementation, and Usage. New York: John Wiley & Sons, 1982.
24. Lindquist, Timothy E. "The Application of Software Metrics to the Human-Computer Interface," IEEE Computer Conference-Fall. 239-244 (1983).

25. Mallary, Thomas C. Design of the Human-Computer Interface for a Computer Aided Design Tool for the Normalization of Relations. MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1985.
26. Marcellus, Daniel H. Systems Programming for Small Computers. Prentice-Hall: Englewood Cliffs, NJ, 1984.
27. Martin, James. Design of Man-Computer Dialogues. Englewood Cliffs: Prentice-hall, 1973.
28. McGilton, Henry and Rachel Morgan. Introducing the Unix System. McGraw-Hill: New York, 1983.
29. Moore, Paul A. Extension of the Software Development Workbench to Include Microcomputer Workstations. MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1984.
30. Myers, Glenford J. Software Reliability Principles and Practice. New York: Wiley, 1976.
31. Ream, Edward K. "A Portable Screen-Oriented Editor," Dr. Dobbs Journal, Number 63: 18-26 (January 1982).
32. Rissland, Edwin L. "Ingredients of Intelligent User Interfaces," International Journal of Man-Machine Studies, 21: 377-388 (October 1984).
33. Shooman, Martin L. Software Engineering. New York: McGraw-Hill, 1983.
34. Sisson, Norwood, et al. "Design Methodology for Menu Structures," IEEE Phoenix Conference on Computers and Communication. 557-560 (1983).
35. Smith, F. J. and C. Estall. "Information Retrieval from an Intelligent Terminal," IEEE Phoenix Conference on Computers and Communication. 206-210 (1985).
36. Smith, John M. "Integrated Management of Multiple Information Types," International Journal of Man-Machine Studies, 21: 403-406 (October 1984).
37. Software Development Documentation Guidelines and Standards (Draft #3). Air Force Institute Of Technology Department of Engineering, Wright-Patterson AFB, OH, (March 1986).

38. The American Heritage Dictionary of the English Language. Ed. William Morris, Boston: American Heritage Publishing Co & Houghton Mifflin, 1973.
39. Thomas, Charles W. An Automated/Interactive Software Engineering tool to Generate Data Dictionaries. MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1984.
40. Tietelbaum, Tim and Thomas Reps, "The Cornell Program Synthesizer: A Syntax-Directed Programming Environment", in Interactive Programming Environments Ed. David R. Barstow et al, New York: McGraw-Hill, 97-116, 1984.
41. Ullman, Jeffrey D. Principles of Database Systems. Rockville, MD: Computer Science Press, 1982.
42. Wilson, Gerald A. "Smart User Interfaces for the Distributed Information Environment," IEEE EASCON-83. 377-385 (1983).
43. Wood, John, et. al. INGRES Version 6.3 Reference Manual. (2 April 1981).
44. Woffinden, Duard S. Interactive Environment for a Computer-Aided Design System. MS Thesis, Naval Postgraduate School, Monterey, CA, 1984.
45. Zenith Data Systems Corporation. Technical Manual-Hardware Z-100 Series Computers. Saint Joseph, MI, 1983.
46. Zenith Data Systems Corporation. Z-100 Series User's Manual. Saint Joseph, MI, 1982.

VITA

Captain Jeffrey W. Foley was born on 30 December 1955 in Cincinnati, Ohio. He graduated from Mariemont High School in 1974. He received an appointment to the United States Military Academy in 1974 and graduated in 1978 with an academic area of concentration in Civil Engineering. He was commissioned a Second Lieutenant in the Signal Corps.

Captain Foley's military schooling includes the Signal Officer's Basic and Advanced Courses, Communications - Electronic Staff Officer's Course, and the Teleprocessing Operations Officer Course. His assignments include the 50th Signal Battalion (AbnC), Ft. Bragg, NC, as a platoon leader and staff officer, and as a staff officer in the US Forces, Korea, Assistant Chief of Staff J-6. He served as Company Commander of A Company 304th Signal Battalion prior to his assignment at the School of Engineering, Air Force Institute of Technology in January of 1985.

Permanent address: 3800 Ashworth Dr.

Cincinnati, Ohio, 45208

REPORT DOCUMENTATION PAGE

A172406

1. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS	
2. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution unlimited.	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/EE/86J-5		5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION School of Engineering	6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION	
6c. ADDRESS (City, State and ZIP Code) Wright-Patterson AFB, OH 45433		7b. ADDRESS (City, State and ZIP Code)	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State and ZIP Code)		10. SOURCE OF FUNDING NOS.	
		PROGRAM ELEMENT NO.	PROJECT NO.
11. TITLE (Include Security Classification) See Item 19		TASK NO.	WORK UNIT NO.
12. PERSONAL AUTHOR(S) Jeffrey W. Foley, B.S., Captain, US Army			
13a. TYPE OF REPORT MS Thesis	13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) 1986, June	15. PAGE COUNT 184
16. SUPPLEMENTARY NOTATION			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB. GR.	
09	02		
		Data Dictionary, Databases, Editors, Human-Computer Interface	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)			
11. TITLE: DESIGN OF A DATA DICTIONARY EDITOR IN A DISTRIBUTED SOFTWARE DEVELOPMENT ENVIRONMENT			
Thesis Chairman: Dr. Thomas C. Hartrum			
Approved for public release: IAW AFR 190-1. Lynn E. WOLVER Dean for Research and Professional Development Air Force Institute of Technology (AFIT) Wright-Patterson AFB OH 45433 3 Sep 86			
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT CLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐		21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Thomas C. Hartrum	22b. TELEPHONE NUMBER (Include Area Code) (513) 255-3576	22c. OFFICE SYMBOL AFIT/FNG	

The project involved the design and implementation of a data dictionary system in a distributed development environment. The distributed environment consists of a central computer that hosts a database management system, a conglomerate of workstations, and the communications links between the workstations and central computer.

The emphasis of the research was placed on the design of a user-friendly data dictionary editor that was implemented on a prototype workstation. Data dictionary definitions are created and updated at the workstation and transferred between the workstation and central computer database.

Background information is provided on data dictionary systems, aspects of human-computer interfaces, and distributed environment interface issues. The design and development of the special editor and the database interface software are described in detail.

Evaluation of the special editor was performed by a subset of the target user group. This evaluation was based on a tool designed to measure user satisfaction. The tool is described and the results of the evaluation provided.

END

1/1-S6

DTIC