

University of Montana

ScholarWorks at University of Montana

Graduate Student Theses, Dissertations, &
Professional Papers

Graduate School

1984

SuperBIB: A set of superior programs for the Unix bibliographic system

A. Y. Liao

The University of Montana

Follow this and additional works at: <https://scholarworks.umt.edu/etd>

Let us know how access to this document benefits you.

Recommended Citation

Liao, A. Y., "SuperBIB: A set of superior programs for the Unix bibliographic system" (1984). *Graduate Student Theses, Dissertations, & Professional Papers*. 5090.
<https://scholarworks.umt.edu/etd/5090>

This Thesis is brought to you for free and open access by the Graduate School at ScholarWorks at University of Montana. It has been accepted for inclusion in Graduate Student Theses, Dissertations, & Professional Papers by an authorized administrator of ScholarWorks at University of Montana. For more information, please contact scholarworks@mso.umt.edu.

COPYRIGHT ACT OF 1976

THIS IS AN UNPUBLISHED MANUSCRIPT IN WHICH COPYRIGHT SUBSISTS. ANY FURTHER REPRINTING OF ITS CONTENTS MUST BE APPROVED BY THE AUTHOR.

MANSFIELD LIBRARY
UNIVERSITY OF MONTANA
DATE: 1984


SUPERBIB - A SET OF
SUPERIOR PROGRAMS FOR THE UNIX BIBLIOGRAPHIC SYSTEM

by
A.Y. (Michelle) Liao

Presented in partial fulfillment
of the requirements for the degree of
Master of Science
UNIVERSITY OF MONTANA
1984

Approved by


Chairman, Board of Examiners


Dean, Graduate School

Date

3/19/85

UMI Number: EP40554

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI EP40554

Published by ProQuest LLC (2014). Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against unauthorized copying under Title 17, United States Code



ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Liao, A.Y. (Michelle), M.S., Dec. 1984 Computer Science

SuperBIB - a Set of Superior Programs for the Unix
Bibliographic System (190+viii pp.)

Director: Dr. Alden Wright



This project analyzed, designed and implemented a set of human-engineered superior programs, SuperBIB, for the Unix bibliographic system, Unix-bib. SuperBIB frees its user from the rigid operations of Unix-bib. The user may become comfortable with using SuperBIB in a short time, due to the fact that SuperBIB is menu-driven, screen-oriented and employs windows.

SuperBIB was developed by using modern techniques of software engineering. The functional specifications of SuperBIB were developed through the use of De Marco's Structured Analysis techniques. Due to the fact that predominantly this project was to develop a user interface, the SuperBIB analysis was complemented by developing a menu tree, a hierarchy of menus, and a draft of a user's guide at an early stage. The design of SuperBIB was developed through the use of principles of good software design and Structured Design techniques. To document the design, Structure Charts were used. To verify the design, a prototype was written. Top-Down and Structured Programming techniques were used to implement SuperBIB. Top-down testing was conducted along the way.

This project proved that programs can be made easier to use by adapting the human-engineering techniques that are commonly applied to microcomputer (personal computer) software.

TABLE OF CONTENTS

ABSTRACT	ii
1. PROJECT FORMULATION	1
1.1 Fundamentals of a Bibliography	2
1.2 An Overview of Current Methods	3
1.2.1 A Typical Case	4
1.2.2 Typical Problems of Manual Methods	4
1.2.3 Aids Available But Not Used	5
1.3 Features of Automated Bibliographic Systems	6
1.3.1 Automated Systems Available at UM	6
1.3.2 A Summary of Features	7
1.3.3 Features of Unix-bib	10
1.3.4 Features of BIBLIO	10
1.3.5 Features of LIBHCOR	12
1.4 Problems of Current Automated Systems	12
1.4.1 A Summary of Problems	13
1.4.2 Problems of Unix-bib	13
1.4.3 Problems of LIBHCOR	15
1.4.4 Problems of BIBLIO	16
1.5 Solutions	18
1.5.1 Human Engineering	18
1.5.2 Justification for Developing SuperBIB	20
2. ANALYSIS	22
2.1 Fundamentals of Analysis	23
2.1.1 What Is An Analysis	23
2.1.2 Difficulties of Analysis	23
2.1.3 What Is Structured Analysis ?	24
2.1.4 Elements of the Data Flow Diagram	25
2.1.5 The Derivation of the Data Flow Diagram	26
2.2 An Overview of Unix-bib	26
2.3 Structured Analysis of Unix-bib	29
2.3.1 Physical Data Flow Diagram of Unix-bib	31
2.3.2 Logical Data Flow Diagram of Unix-bib	33
2.4 A Summary of the SuperBIB Requirements	33
2.5 Structured Analysis of SuperBIB	35
2.5.1 Logical Data Flow Diagrams of SuperBIB	36
2.5.1.1 Context Diagrams	37
2.5.1.2 Level-0 Data Flow Diagram of Super-BIB	38

2.5.1.3 Add Reference Entries	41
2.5.1.4 Inquire Reference Entries	42
2.5.1.5 Format the Bibliography	44
2.5.1.6 Modify Reference Entries	45
2.5.1.7 Sort Reference Entries	46
2.5.2 Physical Data Flow Diagram of SuperBIB	47
2.5.3 Data Dictionary Processor	48
2.6 Templates	51
2.7 Analysis of Human-machine Interaction	54
2.7.1 Menu Tree	54
2.7.2 Samples of SuperBIB Menus	58
2.8 Problems Encountered	60
2.8.1 Problems of Deriving Data Flow Diagrams	61
2.8.2 Different Views of Unix-bib	62
2.8.3 Solutions to Different-view Problem	65
2.8.4 Characteristics of A User Interface Pro- ject	66
2.8.5 Conclusions	67
3. DESIGN	68
3.1 Fundamentals of Software Design	69
3.1.1 What IS Design?	69
3.1.2 Software Costs and Design	69
3.2 Principles of Good Software Design	70
3.2.1 Top-down Approach	71
3.2.2 Top-down Design	73
3.2.3 Ease of Expansion and Contraction	74
3.3 A Summary of the SuperBIB Requirements	74
3.4 Structured Design and its Application	75
3.4.1 Structure Chart	76
3.4.2 Transform Analysis	77
3.4.3 Transaction Analysis	82
3.5 Architectural Design for SuperBIB	84
3.5.1 SUPERBIB	85
3.5.2 ADD BIB	88
3.5.3 Action Modules of ADDBIB	88
3.5.4 Detailed Modules	90
3.5.5 Notes from Designer	90
3.6 Pseudo Code of SuperBIB	91
3.6.1 SUPERBIB	92
3.6.2 ADD BIB	93
3.6.3 ADD BOOK	94
3.6.4 ADD AUTHOR'S NAME	95
3.6.5 ADD NAMES	96
3.6.6 INQUIRE BIB	97
3.6.7 FMT BIB	98
3.6.8 MOD BIB	99

3.6.9 HELP	99
3.7 Prototype	99
3.8 Discussions	101
3.8.1 Design Principle	102
3.8.2 Structured Design Techniques	102
3.8.3 Menu Tree and User's Guide	104
3.8.4 Pseudo Code	105
3.8.5 Prototyping	106
3.8.6 Analysis Phase Is Most Important	107
4. IMPLEMENTATION	109
4.1 Principles of Good Software Implementation	109
4.1.1 Top-down Implementation	109
4.1.2 Structured Programming	111
4.2 Implementation Environment for SuperBIB	112
4.2.1 Unix Programming Environment	112
4.2.2 Software Project management System	113
4.2.3 Choice of Programming Language	115
4.3 SuperBIB Implementation	117
4.3.1 SuperBIB Source Files and Modules	118
4.3.2 Header File	118
4.3.3 Name a Reference File	120
4.3.4 SuperBIB's Executive Module	120
4.3.5 Add Reference Entries	121
4.3.6 Add Regular Book Entries	123
4.3.7 Add Author Names	124
4.3.7.1 Add Names	124
4.3.8 Search References	125
4.3.9 Print Bibliography	126
4.3.10 Modify Reference File	127
4.3.11 Sort References	127
4.3.12 Exit SuperBIB System	128
4.3.13 Index File of the Bibliographic Database	128
4.3.14 On-line Help	128
4.4 Problems Encountered	129
4.5 Conclusions	131
5. CONCLUSION	133
5.1 Human-Engineering	133
5.1.1 What are Human-Engineering Techniques?	133
5.1.2 Why not in Mainframe?	134
5.1.3 Theory and Hypothesis	135
5.1.4 Transition to Mainframe	135
5.1.4.1 Variety of Terminals	136
5.1.4.2 Reverse Video	138
5.1.4.3 300-Baud Links	138
5.1.4.4 Lack of Editing Power	138

5.1.5 More Useful?	139
5.2 Value of Software Engineering Methods	140
5.2.1 Analysis	141
5.2.2 Design	143
5.2.3 Implementation	144
5.3 Future Work	145
5.3.1 Minor Enhancements	146
5.3.2 Major Enhancements	146
5.3.3 Interfaces with BIBLIO, LIBHCOR and WLN	148
Appendix A	150
Appendix B	152
BIBLIOGRAPHY	189

LIST OF TABLES

1-1 The Characteristics of UM's Bibliographic Systems	7
1-2 Features of UM's Bibliographic Systems	8
1-3 Problems of UM's Bibliographic Systems	14
2-1 A Sample Unix-bib Reference File	27
2-2 Key-letters Recognized by Unix-bib	28
2-3 A Sample Terminal Session	30
2-4 A Sample Bibliography	31
2-5 The NAME A REFERENCE FILE Menu	59
2-6 The SELECT A COMMAND Menu	60
3-1 The Proposed Window Layout	92
3-2 Key-letters Recognized by SuperBIB	97
4-1 SPMS Project Directory Tree	115
4-2 Source-code Files and Modules	120

LIST OF ILLUSTRATIONS

2-1 The Physical Data Flow Diagram of Unix-bib	32
2-2 The Logical Data Flow Diagram of Unix-bib	34
2-3 The Context Diagram of SuperBIB	37
2-4 The Logical DFD of SuperBIB	39
2-5A The Logical DFD of ADD REFERENCE ENTRIES	41
2-6 The Logical DFD of INQUIRE REFERENCE ENTRIES	43
2-7 The Logical DFD of FORMAT BIBLIOGRAPHY	44
2-8A The Logical DFD of MODIFY REFERENCE ENTRIES	45
2-9 The Logical DFD of SORT REFERENCE ENTRIES	47
2-10 The SuperBIB Menu Tree	55
3-1 The First Cut of the SuperBIB Design	80
3-2 The Architectural Design of SuperBIB	86

CHAPTER 1

PROJECT FORMULATION

Currently, almost all of the many thousands of bibliographies produced on the University of Montana (UM) campus are prepared and maintained by manual methods. The tasks involved are error-prone, repetitious, laborious and time-consuming. In addition, the manual process has limited search and select capabilities, valuable functions for preparing and maintaining bibliographies.

Many bibliographic systems have been developed to automate these tasks. Among these are the Unix bibliographic systems, Unix-bib, a group of the SYSTEM-1022 [1-1] programs, BIBLIO, and LIBHCOR, a BASIC program. All three of these systems are available on UM's computers. However, the programs are used very little. Perhaps some of those who otherwise would use these programs do not do so because they are unaware of their existence, their department lacks a computer services budget, etc. Certainly one important reason is that these programs are not

[1-1] SYSTEM-1022 is a database management system.

easy to use.

The hypothesis of this thesis is that such programs can be made easier to use by utilizing human engineering (user-friendly) techniques that are commonly applied to microcomputer software. To test that hypothesis, a preprocessor (user interface) for the Unix bibliographic system was successfully analyzed, designed, and implemented. This preprocessor, "SuperBIB," is menu-driven, screen-oriented and employs windows.

1.1. FUNDAMENTALS OF A BIBLIOGRAPHY

A regular bibliography contains a list of citations (authors' names, titles, and publication dates, etc.) that refer people to publications for a particular subject. One may use a regular bibliography to record the publications that he has read, or desires to read [1-2]. An educator may use a regular bibliography which lists the publications that he expects his students to read. There should be a bibliography (or reference) at the end of a research paper, an article, or a book. A regular bibliography may

[1-2] To avoid awkward syntax the author has chosen to employ the traditional masculine pronouns as the generic forms.

have different formats, such as the one specified by the Modern Language Association (MLA), social science and natural science formats. Thus, one bibliography differs from another by use and by styles for different disciplines.

Furthermore, some bibliographies are annotated. An annotated bibliography lists citations and describes publications with abstracts, comments, library call number, the nearest location of a copy of the publication available, or other information. It provides us with some further information about the publication beyond the bare citation.

An abstract may have 150-300 words; a comment may have up to 1000 words; a library call number may have 5-12 characters, and the nearest location of an available publication may be a code name (or a full name) of a local library or institution.

1.2. AN OVERVIEW OF CURRENT METHODS

Current methods of processing bibliographic information are these:

- (1) pencil and paper, or index cards,
- (2) computer text editors,
- (3) automated bibliographic systems.

1.2.1. A TYPICAL CASE

When the author started this project, she visited a representative department at this University, History. The author found that they manually prepared their bibliographic information by using index cards. In one case an estimated 3000 sets of bibliographic data (citations, annotations) were recorded on index cards and published later as a bibliographic book [1-3]. This is not an isolated case. Many faculty members and students are preparing their bibliographies manually at UM.

1.2.2. TYPICAL PROBLEMS OF MANUAL METHODS

The manual process is error-prone, laborious, repetitious and time-consuming, at best. First, it requires a great deal of effort to record the bibliographic information on index cards. Once the information is recorded, it is not easy to make changes. This method provides only

[1-3] by L. Frey.

limited sort, search, and select capabilities. To provide basic search and inquiry capabilities, duplicate information may have to be maintained, but in different orders. Furthermore, the printing is entirely separate from the manual preparation of bibliographies. To get camera-ready bibliographies, typesetting or typing is necessary.

The preparation of bibliographies is a perfect application for the computer, since the computer is good at recording, searching and sorting information. The information stored in the computer may be easily modified and camera-ready bibliographies may be generated at a touch of the finger.

1.2.3. AIDS AVAILABLE BUT NOT USED

There are three automated bibliographic systems available on this campus; however, they are used very little. These systems, Unix-bib, BIBLIO, and LIBHCOR, are at least fairly powerful; they accept, alter, search, sort, format and print bibliographic information with a minimum of effort. All UM departments have fairly adequate computer access, and a computer services budget; however, most people at UM still use manual methods to prepare their bibliographies. Why do they continue to use manual methods? Possible answers to that question are these:

- (1) they are unaware of the existence of any automated bibliographic system,
- (2) the bibliographic systems are difficult to use,
- (3) they have difficulties in accessing computers during prime time,
- (4) they do not have a sufficient computer budget,
- (5) they are afraid of computers,
- (6) some bibliographic information is not in English and is therefore difficult to process with current systems.

1.3. FEATURES OF AUTOMATED BIBLIOGRAPHIC SYSTEMS

1.3.1. AUTOMATED SYSTEMS AVAILABLE AT UM

The characteristics of the bibliographic systems available at UM are summarized in Table 1-1.

Table 1-1 Characteristics of the Bibliographic Systems Available at UM.

Name	Host Computer	source code written in	Affiliate Responsible	Source of Information
Unix-bib	Vax-11/750 & 785	the C language	Computer Science & CC.	J.Barr
BIBLIO	DEC-20	PL1022	Computer Center(CC)	R.Walton
LIBHCOR	DEC-20	BASIC-PLUS 2	Chemistry	R.Field

Note that Unix-bib is an abbreviation for the Unix bibliographic system. PL1022 is a programming language in the SYSTEM-1022 database management system.

1.3.2. A SUMMARY OF FEATURES

All three bibliographic systems produce bibliographies with a minimum of effort. Major features of these three systems are summarized in Table 1-2.

Table 1-2 Features of the Bibliographic Systems
Available at UM.

Features	Unix-bib	BIBLIO	LIBHCOR
Reference file creation/update	X	X	X
Search	X	X	X
Regular bibliography	X	X	X
Annotated bibliography	X	X	
Reference file alteration w/o using editor		X	X
Sort	X	X	
Multiple annotation fields	X		
Classification; Library call number	X	X	X
Verification		X	X
DELETE/DFIND		X	

Note that the check mark, X, indicates the feature exists in the particular bibliographic system.

The labels used in Table 1-2 are described below briefly:

- (1) Reference File Creation or Update: The bibliographic system may create a reference file, or append reference entries to the end of a reference file, where a reference file is a collection of reference entries; a reference entry is a collection of all entry elements for a particular publication; an entry element may be a citation (an author's name, a title, or a publication date, etc.), and an annotation (an abstract, comments, etc.),
- (2) Reference File Alteration Without Using Editor: The user may alter a reference file without invoking a text editor.
- (3) Search: The user may search for a particular entry in a reference file.
- (4) Sort: The user may sort the reference entries according to certain criteria, i.e., the authors' last names.
- (5) Multiple Annotation Fields: The user may enter annotation information such as abstracts, comments, or the nearest location of a copy of the publication available, etc.
- (6) Classification, Library Call Number: This is an analogue of a library card catalogue system. For example, the user may give an unique name (or number) to all publications about a particular research project. The purpose is to organize the reference file and to retrieve the information easily.
- (7) Verification: The user may verify and correct the inputs of a particular reference entry.
- (8) DELETE/DFIND: The user may drop some reference entries out of sight, but these entries are still on-line and may be retrieved by using DFIND command.

1.3.3. FEATURES OF UNIX-BIB

Unix-bib is one of the most powerful bibliographic systems currently available. Unix-bib can construct reference files. Each reference entry contains all information about a particular publication. One or more reference files and as many reference entries as desired may be maintained in Unix-bib. Quick access to a reference entry stored in a reference file is provided by maintaining an inverted index. All, or a part, of the entries in the reference file may be formatted into a bibliography. The bibliography may be displayed on a terminal, may be stored in a file, or may be printed on a printer. An imprecise (incomplete) indication of a reference citation, such as '[. Budd 1982 .]', in an Nroff (a Unix text processing system) text file may be replaced by a more precise citation string, such as [1], corresponding to the entry in its bibliography. The information about the nearest location of a copy of the publication is provided optionally.

1.3.4. FEATURES OF BIBLIO

BIBLIO has capabilities which are similar to Unix-bib's. According to "The User's Guide for BIBLIO - A Computerized Bibliographic System" by the University of Chicago [BIBLIO, 1981], BIBLIO can:

- (1) create one or more sets of reference files with as many reference entries as desired,
- (2) easily make alterations to an entire reference file, or to an individual reference entry,
- (3) call up an individual or group of reference entries,
- (4) organize reference entries in accordance with specific classifications, i.e. all reference entries related to a particular research project,
- (5) select reference entries by various criteria, i.e. a particular author's name, or date of publication,
- (6) format reference entries for printing the bibliography,
- (7) print all or a part of a reference file.

BIBLIO allows its user to verify the input before updating the files, and to alter the input if necessary. BIBLIO may easily organize the reference entries in accordance with specific classifications. Four categories of information for each reference entry may be entered by the BIBLIO user. The four categories are the citation, abstract, search words (key-words, or 'list of topics' of BIBLIO) and classification ('library number' of BIBLIO). The DELETE command of SYSTEM-1022 drops some reference entries out of the user's sight, however these references are still on-line and may be retrieved by using the DFIND command. Some users may find this feature useful.

1.3.5. FEATURES OF LIBHCOR

LIBHCOR is the most pleasant one to use among the three. But LIBHCOR is also the simplest and least powerful. According to "Computerized Bibliography on Homogeneous Chemical Oscillating Reactions (LIBHCOR) Search Manual" by M. Burger [Burger,1983], LIBHCOR allows its users to:

- (1) enter titles, authors' names, publishers, other search words (key-words), and classifications ('reference numbers' of LIBHCOR),
- (2) search for the authors' names, titles, publishers, other search words (key-words), classifications, or a combination of these,
- (3) store the retrieved reference entries in a file,
- (4) print the bibliography.

Before pressing the RETURN key on the terminal keyboard, LIBHCOR allows the user to alter the inputs of a reference entry. After the return key is pressed, one must use a text editor to alter input.

1.4. PROBLEMS OF CURRENT AUTOMATED SYSTEMS

1.4.1. A SUMMARY OF PROBLEMS

The problem with all three bibliographic systems is that the software is not easy to use. Problems of the systems are summarized in Table 1-3.

1.4.2. PROBLEMS OF UNIX-BIB

The problem with Unix-bib is that it is not very easy to learn, due to the fact that Unix was written originally for sophisticated software developers, and the fact that the Unix programming environment is highly diversified. And it is easy to forget how to use Unix-bib, due to the fact that the human engineering work was inadequate in this system, especially relating to the creation (or update) of a reference file. A part of the reason is that few users would use a bibliographic system daily. Therefore, they are not likely to remember every little detail.

Unix-bib is particularly difficult to experiment with because it demands a rigid format for its reference file, and a working knowledge of one of the Unix editors (vi, ed, etc.) is required. To produce a bibliography, a minimum of three Unix programs (text editor, Invert and Roffbib) have to be called and in sequence. The alteration of a reference file depends completely on a text editor,

Table 1-3 Problems of the Bibliographic Systems at UM.

Problems	Unix-bib	BIBLIO	LIBHCOR
Knowledge of operating system required	X	X	X
Knowledge of programming language required		X	
Knowledge of editor required	X	sometimes	sometimes
Rigid format for reference file	X		
Rigid Typing instructions			X
Upper/lower case letters are different during a search			X
Annotation not available			X
Sorting routine not available			X
Fixed element length		X	X

Note that the check mark, X, indicates the problem exists in the particular bibliographic system.

there are no alternatives.

In short, Unix-bib is tough enough for an experienced user and, it is much more difficult for a novice user.

1.4.3. PROBLEMS OF LIBHCOR

LIBHCOR is not in a public domain on the DEC-20, therefore its access is limited. In addition, LIBHCOR is not built on any database system, hence its features are limited.

LIBHCOR requires its user to follow a set of rigid typing instructions closely. For example, the title of a journal article must start with an upper case letter. For a book title, the first letter of all words except prepositions must be capitalized. The author's name, and all coauthors' names are typed with upper case letters, last name first, a colon, then the initials of the first names. A colon separates the coauthor's name. The first letter of the full title of a journal is upper case followed by the volume number, year (in brackets) and the starting page of the article. In conference proceedings and multi-authored books, the editor is also shown with upper case letters.

LIBHCOR has a fixed number of input lines for each entry element. For example, there are three lines for each title entry, two lines for each author's name entry and a single line for each entry library call number. LIBHCOR always asks for the inputs to the next line if the user has not used them up. The user has to type 'N' to indicate that there is no more input for a particular entry element. In addition, LIBHCOR cannot accept abstracts or comments, therefore it cannot produce an annotated bibliography.

LIBHCOR does not map upper case letters to lower case letters or vice versa. The LIBHCOR user may have to search for both upper and lower case letters for a single search because LIBHCOR differentiates between upper and lower case letters during a search. In addition, LIBHCOR cannot sort a bibliography.

1.4.4. PROBLEMS OF BIBLIO

BIBLIO is not completely implemented, a minimal knowledge of the SYSTEM-1022 database management system is needed, therefore a naive user may not be able to use BIBLIO at all.

Certainly BIBLIO is not easy for a novice user. For example, BIBLIO requires its user to type 'USE BIBLIO:ADDON' to add information into a reference file. To know an appropriate command to invoke, the BIBLIO user may have to memorize it, or refer to a user's guide.

Part of BIBLIO is more friendly. For example, it prompts its user with 'K, C, A, B, Q or ?'. Thus, 'K' indicates the keyword file, 'C' indicates the classification (library call number) file, 'A' indicates the abstract file, 'B' indicates the citation file, 'Q' indicates exit, and '?' indicates help. This reminds the user of all the options (commands) he has at this point of operation, and the user does not have to memorize the command, or refer to the user's guide constantly.

One of BIBLIO's problems is that it uses code names and the code name is not very descriptive. For example, to select the type of publication, BIBLIO prompts its user with 'B1, B2, B3, B4, J1, J2, C1, C2, ?, Q'. Thus, 'B1' indicates that the book entry to be added has an author's name, a publishing date, a title, a publishing city, and a publisher, but nothing else. The 'B2' indicates that the book entry has all elements of 'B1' plus a volume number, and edition, but nothing else. The 'B3' indicates that

the book entry has an author's name, a publishing date, a title, additional information #1 and additional information #2, but nothing else.

To search for a particular entry element, the BIBLIO user needs to open a specific type of reference file, then enter an appropriate command. For example, to search for a particular author's (or coauthor's) name, the user needs to open an author file, and enter an appropriate System-1022 search command. Commands to search for the publications which are authored by Smith and coauthored by Jones are listed below:

- (1) OPEN author_filename,
- (2) FIND AUTHOR CONTAINS smith OR jones. (This may be abbreviated as 'F AUT CT smith OR jones'.)

A BIBLIO entry element has a maximum length. For example, the maximum length for an author's name entry is 80 characters. This may or may not be a problem.

1.5. SOLUTIONS

1.5.1. HUMAN ENGINEERING

The hypothesis is that programs can be made easier to use (hence, will be more widely used) by adapting human-

engineering techniques that are commonly applied to micro-computer software.

The software written for mainframe computers will likely not be human-engineered, because the mainframe user is often expected to be more sophisticated. Microcomputers are primarily for members of the general public who may not be familiar with computers, or programming languages. Therefore, microcomputer software tends to be easier to use and attempts to be "user-friendly". To accomplish this, microcomputer software is often menu-driven, screen-oriented, and employs windows.

Menu-driven software displays a menu of options (commands) on the user's terminal screen. The user may indicate his selection by pressing one or two keys on his terminal keyboard. The user usually does not need to memorize the command, or refer to the user's guide constantly. Most commands he may need at one point of software execution are displayed on the terminal. Further, to indicate a command desired, the user does not have to spell out every letter of the command; he only needs to press one or two keys. In addition, function keys may be programmed to allow the user to issue commands in shorthand. Therefore, menu-driven software is easy to use.

A screen-oriented program employs the full capabilities of a modern video terminal. The direct cursor addressing capability enables input and output at any spot of a video screen, therefore an entire screen may be utilized at any given time. A non-screen oriented program usually behaves as if employing a paper-based terminal and the input or output is sequentially printed down the screen (paper). It is more difficult to implement menus and other human-engineering techniques with this type of program and terminal.

A program may divide a video terminal screen into multiple windows, each a miniature of the screen. One window may be used for displaying the menu options and accepting the user's selection. The other window may be used for displaying on-line help, if desired. Thus, the user may access on-line help while a window for the current operation remains intact. Multiple windows can establish a vivid dialogue between the user and the system, and allow the user to utilize multiple terminal screens virtually while physically he has only one.

1.5.2. JUSTIFICATION FOR DEVELOPING SUPERBIB

The author attempted to adapt the above human engineering techniques for one bibliographic system,

Unix-bib. Unix-bib is chosen over BIBLIO and LIBHCOR for the following reasons:

- (1) It was suggested by an expert Unix-bib user, the project commissioner, Dr. Barr. He personally used Unix-bib and felt that it should have been improved in the areas of user-friendliness.
- (2) The author is interested in programming in the Unix environment and in the C language. Unix provides the most powerful programming environment compared to any other operating system currently in the market. Unix has a rich collection of software tools. By using these software tools it is likely that a software developer does not have to write his programs from scratch.
- (3) A Vax-11/750 computer, running under the Unix operating system, provides broad services to the faculty and students in this Department. In addition, UM has acquired a Vax-11/785 computer which is running under Unix. The Vax computers are likely to become major teaching tools in this University. A human engineered Unix bibliographic system might be utilized by a good number of faculty members and students on this campus, if it had been developed successfully.
- (4) Unix is gaining popularity nationwide and it is available in both mainframes and microcomputers. Most universities and colleges have Unix systems. Business executives are beginning to recognize the power of Unix.

CHAPTER 2

ANALYSIS

The development of any medium to large-sized software project involves three phases, analysis, design, and implementation. Among these the analysis phase is the most important. The systems analyst must prepare a precise document prescribing what has to be done to solve the problem at hand. In the case of replacement of an existing system, a systems analyst studies the current system, identifies the problems (if any), proposes a solution to the problem (if applicable), and identifies the requirements of the new system. In the case where there is no current system, the analyst skips the first three steps, and only identifies the requirements of a proposed system. In either instance, the user is to verify the document prepared by the analyst before the analysis phase is completed.

The methodology of systems analysis, the analysis of the current Unix bibliographic system (Unix-bib) and the analysis of a new Unix bibliographic system (SuperBIB) are discussed below.

2.1. FUNDAMENTALS OF ANALYSIS

2.1.1. WHAT IS AN ANALYSIS ?

An analysis is the study of a problem which is to be solved, prior to taking any action. In computer science, analysis refers to the procedures adopted to understand a problem. During an analysis, the context and the requirements of the solution to the problem are identified.

2.1.2. DIFFICULTIES OF ANALYSIS

Analysis is not easy. The changing nature of the user's needs makes the analysis even more difficult.

Interpersonal skills are very important to the systems analyst. An analyst has to understand the languages of the software user and the software designer, because he acts as a bridge between the two. The analyst has to communicate well, analyze systems appropriately, and write quality specification documents as well.

Traditionally, the systems analyst writes pages of text to describe his understanding of the existing system (if any), the problem, the solution, and the requirements of the proposed system. This document, the system specification document, usually is quite long and difficult to

grasp quickly.

2.1.3. WHAT IS STRUCTURED ANALYSIS ?

Structured Analysis is a set of techniques which analyze the system graphically, and supplement the graphs with documents written in a subset of English called Structured English. Structured Analysis techniques allow the analyst to perform his duties more easily, due to the fact that graphic representations of the system are easier to establish, and only a minimum amount of writing is required. In addition, the specification documents prepared by using Structured Analysis techniques are easy to understand, therefore Structured Analysis is a better analysis tool than the traditional text-only techniques.

According to Structured Analysis and System Analysis by De Marco [DeMarco,1978], Structured Analysis is the use of Data Flow Diagrams [2-1], a Data Dictionary and a Transform Description (Minispecs, or mini-specifications) in Structured English to build a specification document, Structured Specification. As De Marco points out:

[2-1] See Figures 2-1 to 2-4 for examples.

- (1) A Data Flow Diagram (DFD) is a graphic representation of a system. A Data Flow Diagram portrays the system in terms of its components, with all interfaces among the components indicated.
- (2) The Data Dictionary (DD) is a set of definitions of data flows and files. The Data Dictionary provides a single place to look up definitions of terms.
- (3) The Transform Description is the statement describing the policy that governs transformation of input data flow(s) into output data flow(s) at a given primitive process.
- (4) Structured English is a subset of English with limited syntax, limited vocabulary, and an indentation convention to call attention to logical blocking.

2.1.4. ELEMENTS OF THE DATA FLOW DIAGRAM

According to De Marco, Data Flow Diagrams are made up of the following basic elements:

- (1) flows of data, represented by named vectors, -->,
- (2) processes, represented by circles or "bubbles",
- (3) files, represented by short straight lines,
- (4) data sources (originators) and information sinks (receivers), represented by boxes which are outside of the domain of the system under study [2-2].

[2-2] The USER, BIBLIO, LIBHCOR, and WLN of Figure 2-3 are the data originators and information receivers for SuperBIB and these are not in the domain of the SuperBIB system.

2.1.5. THE DERIVATION OF THE DATA FLOW DIAGRAM

According to De Marco, Data Flow Diagrams are derived by using the following techniques.

- (1) Identify all net system inputs (the data coming from the originator), or all net system outputs (the information going into the receiver) and draw them in the outer part of the diagram.
- (2) Identify other data flows between data originators and information receivers; work the way from inputs to outputs, if possible. Otherwise, work backwards from outputs to inputs, or from the middle out.
- (3) Label all data flows among processes.
- (4) Label the processes in terms of their inputs and outputs.

2.2. AN OVERVIEW OF UNIX-BIB

A Unix-bib reference file is a collection of reference entries. A reference entry is a collection of all entry elements for a particular publication. An entry element may contain a part of a citation (an author's name, a title, a publication date, etc.), an annotation (an abstract, or comments), or other information. The entry elements are prefixed with a percent sign (%), a corresponding key-letter and a space. A sample is listed in Table 2-1.

Table 2-1 A Sample Unix-bib Reference File

COMMENTS	FILE AS SEEN ON SCREEN
	.
	.
	.
	%A M. Bishop
	%A L. Snyder
ref. <	%T The Transfer of Information
entry <	%J Proceedings of the 7th SOSP
	%P 45-54
ref. <	-- %D 1979
file <	
	%E R.A. DeMillo
entry ---->	%E D.P. Dobkin
element	%E A.K. Jones
	%E R.J. Lipton
	%T Foundations of Computation
	%I ACPRESS
	%D 1978
	^
key-letter ---	
	.
	.
	.

Note that one blank line separates each reference entry.

Currently Unix-bib recognizes the key-letters listed in Table 2-2.

Table 2-2 Key-letters Recognized by Unix-bib

Key-letter	Description
A	Author's name
B	Title of book containing an article which the user is interested in
C	City of publication
D	Publication date
E	Editor's name
F	Caption
G	Government (NTIS) ordering number
I	Issuer (publisher)
J	Journal name
K	Search words (Key-words)
N	Issue number
O	Other information
P	Page numbers
R	Technical report name
S	Series title
T	Title
V	Volume number
W	Nearest location where a copy of the publication can be found
X	Annotation

In the scope of this thesis project, five Unix programs are associated with the Unix-bib operations. These programs are a Unix text editor(vi, ed, etc.), Invert, Roffbib, Lookbib and Sortbib. The text editor, Invert and Roffbib must be called (in this order) to produce a bibliography appropriately. The text editor is used to edit a reference file. Invert [2-3] is used to create, or

update an inverted index of the reference file for quick search. Roffbib is used to format the bibliography. Optionally, Lookbib [2-4] and Sortbib may be used. Lookbib is used to search for any particular bibliographic information; Sortbib is used to sort the bibliographic information.

To illustrate how Unix-bib produces a bibliography, a sample terminal session is given in Table 2-3.

A sample of the bibliography generated by using Roffbib is illustrated in Table 2-4.

2.3. STRUCTURED ANALYSIS OF UNIX-BIB

The goal of Structured Analysis of Unix-bib is to analyze, to document, and to verify the current operations of the Unix bibliographic systems. The Unix-bib operations are analyzed, and documented in Physical and Logical Data Flow Diagrams. These Data Flow Diagrams were verified by Dr. Barr, an expert Unix-bib user.

[2-3] 'Indxbib' is a Unix program similar to 'Invert'.

[2-4] 'Lookup' is a Unix program similar to 'Lookbib'.

Table 2-3 A Sample Terminal Session of Unix-bib Operations

User's Inputs	Comments
vi reference_filename	
[i]	
%A M. Bishop	
%A L. Snyder	
%T The Transfer of Information	CREATE A
%J Proceedings of the 7th SOSP	REFERENCE FILE
%P 45-54	USING VI EDITOR
%D 1979	
<ESC>	
[ZZ]	
invert reference_filename	CREATE, OR UPDATE AN INVERTED INDEX FOR REFERENCE FILE
sortbib reference_filename	SORT REFERENCE ENTRIES
roffbib reference_filename	FORMAT BIBLIOGRAPHY

Note that [i], <ESC> and [ZZ] are vi editor commands. To enter insert mode, type 'i'. To exit insert mode, press ESC key. To exit vi editor, type 'ZZ'.

Table 2-4 A Sample Bibliography Generated by Unix-bib

=====

BIBLIOGRAPHY

Bishop M. and L. Snyder, The Transfer of Information,
 Proceedings of the 7th SOSP, 1979, 45-54.

·
 ·
 ·

=====

2.3.1. PHYSICAL DATA FLOW DIAGRAM OF UNIX-BIB

Through literature research, interviews with an expert Unix-bib user, the person who commissioned this project, Dr. J. Barr, the author learned how the current Unix bibliographic system works. The author tried to see the Unix-bib operations from the viewpoint of the data, instead of the viewpoint of any human being. This helped the author to derive the data flow diagrams for Unix-bib. To make the project commissioner, Dr. Barr, understand the diagrams easily, the author used the terms that he had used. Therefore, the data flow diagrams are full of Unix commands and filenames, i.e. ' vi reference_filename'. These physical checkpoints helped Dr. Barr and the author to relate the diagrams to the real operations of Unix-bib. Since most of these checkpoints are physical in nature,

this diagram is called the Physical Data Flow Diagram of Unix-bib (Figure 2-1).

The initial analysis of Unix-bib was done when the author walked through the Physical DFD of Unix-bib with Dr. Barr, then playing the role of an expert user, and he accepted it as an accurate representation of his mode of operation. The next step was to derive a Logical DFD of Unix-bib.

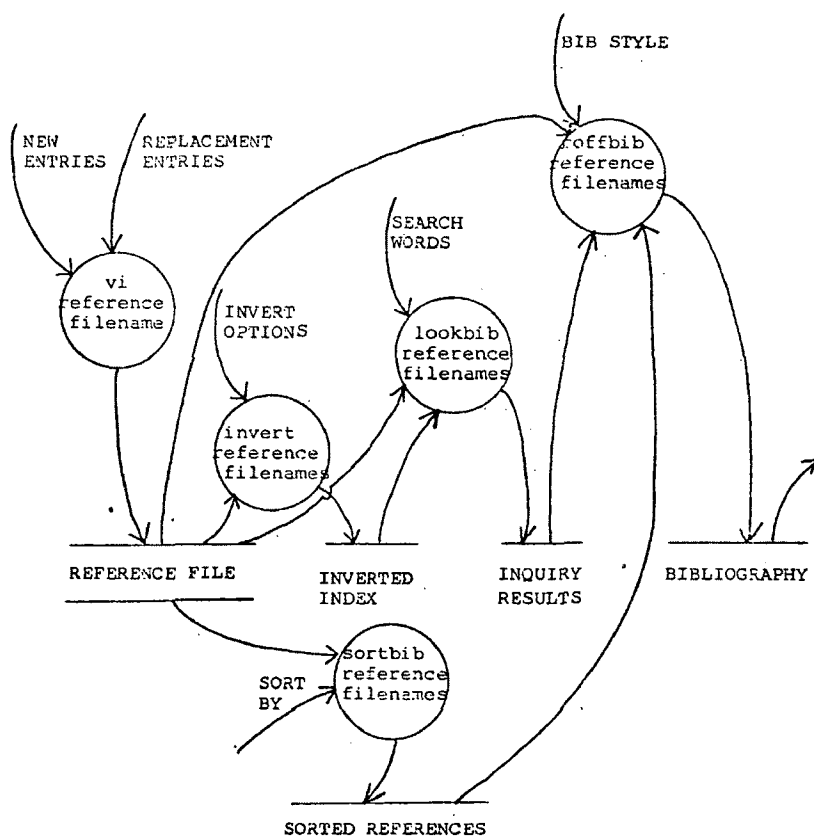


Figure 2-1. Physical Data Flow Diagram of Unix-bib.

2.3.2. LOGICAL DATA FLOW DIAGRAM OF UNIX-BIB

To derive a Logical Data Flow Diagram, the author removed all physical checkpoints and replaced each one with its logical counterpart. The goal was to generalize the Physical DFD, and to divorce the objectives of the operations from the methods of carrying them out. As an example, the 'vi reference_filename' of the Physical DFD was changed to 'edit a reference file' in the Logical DFD. This step was done when the Logical Data Flow Diagram of Unix-bib was drawn, walked through and verified by the project commissioner-expert user. Figure 2-2 is a Logical DFD for Unix-bib which is a transformation of the Unix-bib Physical DFD.

2.4. A SUMMARY OF THE SUPERBIB REQUIREMENTS

The requirements of SuperBIB were identified through literature research, the system analysis done to that point, and interviews with an expert Unix-bib user, Dr. J. Barr, and representatives of the potential SuperBIB users, Professors R. Walton, R. Field, L. Frey, H. Fritz, and R. Dhesi. In summary, Dr. Barr and the potential SuperBIB users wanted SuperBIB to do these things:

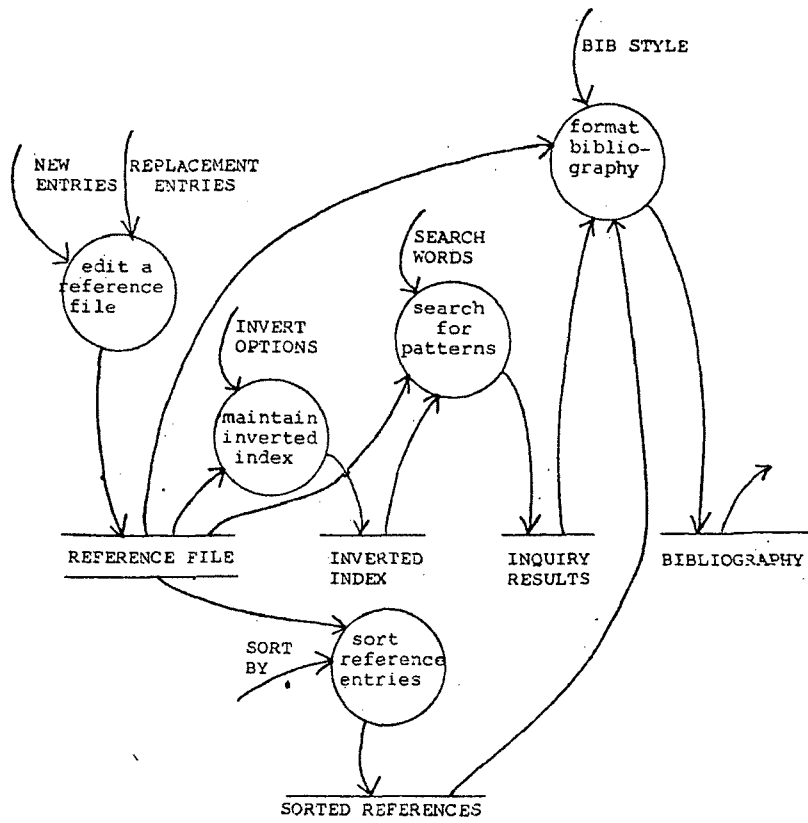


Figure 2-2 Logical Data Flow Diagram of Unix-bib.

- (1) Accept citations (authors' names, titles, publication dates, etc.), annotations (abstracts, comments), and other information (library call numbers, etc.) in a human-engineering (user-friendly) fashion. Ideally, SuperBIB should be menu-driven, screen-oriented, and employ windows,
- (2) Inquire (search and retrieve) for any particular reference entry which meets certain criteria. The user should be able to search for authors' names, titles, publication dates, other search words (keywords), or a combination of these. (All entries matching the above search criteria would be retrieved. The capabilities of wildcard searching and selection of those entries matching a part of search criteria are desired, but not required.)
- (3) Print the reference information in a few basic formats [2-5]. (The capabilities of producing the bibliography in a variety of formats are desired, but not required.)
- (4) Modify the reference information using a text editor, if desired. (The capability of modifying without using an editor is desired, but not required.)
- (5) Sort bibliographic information according to the senior author's last name, and publication date. (The capabilities of sorting entries by other criteria are desired, but not required.)

2.5. STRUCTURED ANALYSIS OF SUPERBIB

The goal of Structured Analysis of SuperBIB is to identify the requirements of SuperBIB, to document the requirements predominantly in a graphic form, and to verify the documents with the representatives of the

[2-5] To produce the bibliography in a variety of formats, working knowledge of 'Nroff' and 'Bib' are required.

potential SuperBIB user.

The Data Flow Diagrams are graphic, partitioned and multidimensional. It is easy to understand DFDs compared to a long text. The DFDs display major partitions of the system first, then go down to the details. The DFDs emphasize the flow of data, but de-emphasize the control information such as executive components, decision-making or repetition, due to the fact that data flows are stable and control information may be not.

The SuperBIB requirements were analyzed and documented in the Physical and Logical Data Flow Diagrams that are promoted by De Marco. The Logical DFDs illustrate the functionality of SuperBIB and the Physical DFD presents a portion of these functionalities that are to be automated in this thesis project.

2.5.1. LOGICAL DATA FLOW DIAGRAMS OF SUPERBIB

In theory, at the analysis stage, an analyst is purely to describe what has to be done, not concerning himself at all with how it will be accomplished. At the stage of deriving the Logical DFDs, he supposedly does not even distinguish between those that will be automated and those that will not.

Due to the complexity of this project, a leveled set of Logical Data Flow Diagrams was developed. A leveled set of Data Flow Diagrams is composed of a hierarchy of DFDs. According to De Marco, a leveled set of DFDs is made up of a top, a bottom and a middle. The top is a single diagram called the Context Diagram. The bottom consists of a set of unpartitioned bubbles, called functional primitives. The middle is everything else.

2.5.1.1. CONTEXT DIAGRAMS

Figure 2-3 is a Context Diagram of SuperBIB. The goal of deriving a Context Diagram is to declare the domain of this project. The domain of this project is illustrated by a circle, labeled SuperBIB. The net input of SuperBIB is the user's bibliographic input (USER'S INPUT), such as citations, and the net output is the SuperBIB output (SUPERBIB'S OUTPUT), such as the bibliog-

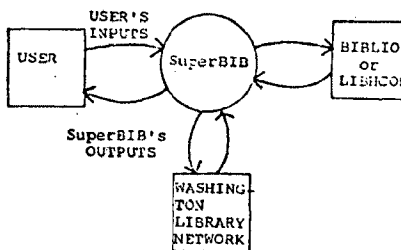


Figure 2-3 Context Diagram of SuperBIB.

raphy, inquiry results, or sorted reference entries. BIBLIO, LIBHCOR, or WASHINGTON LIBRARY NETWORK (WLN), represented by boxes, are not in the domain of this project.

2.5.1.2. LEVEL-0 DATA FLOW DIAGRAM OF SUPERBIB

Figure 2-4 is a Logical Data Flow Diagram (Level-0, Diagram-0) of SuperBIB. The primary requirement of SuperBIB is to accept bibliographic information, and to produce a reference file which meets the requirements of the Unix bibliographic system (Unix-bib). A process, ADD REFERENCE ENTRIES, is to accomplish this task. ADD REFERENCE ENTRIES is to accept, transform and store new reference entries (NEW ENTRIES) in a reference file (REFERENCE FILE).

SuperBIB should search for reference entries that match the search words supplied by the user. The search words may be authors' last names, words from titles, publication dates, other search words, or a combination of these. A process, INQUIRE REFERENCE ENTRIES, is to accomplish this task. INQUIRE REFERENCE ENTRIES gets reference entries from REFERENCE FILE, or SORTED REFERENCE FILE. It accepts user's search words, and searches for reference entries matching the words. It maintains a file (INQUIRY-RESULT FILE) which holds the matched reference entries.

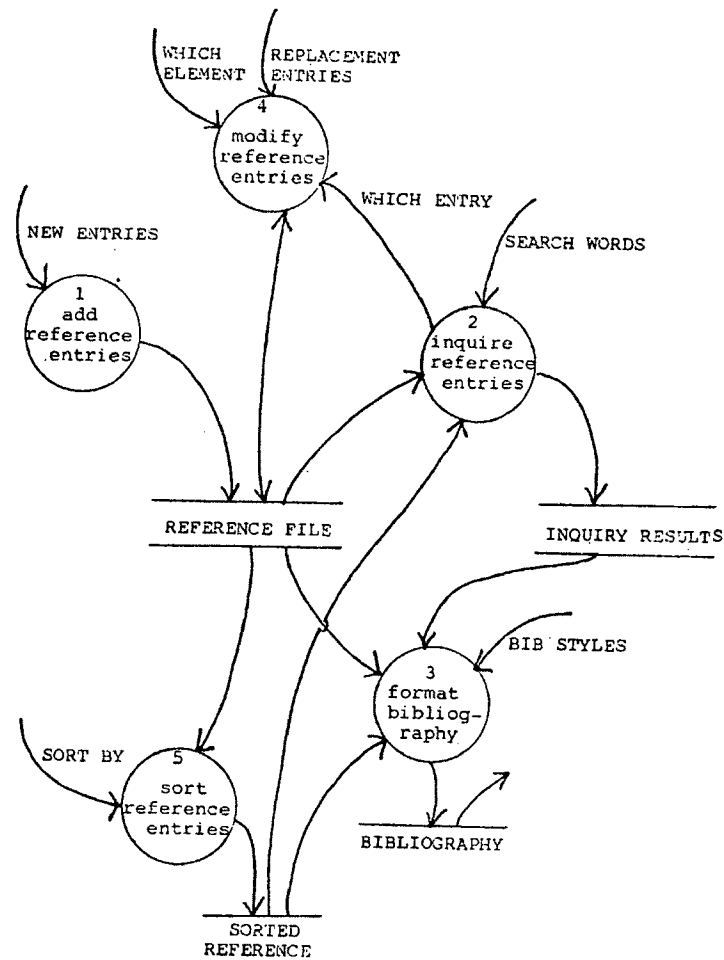


Figure 2-4 The Logical Data Flow Diagram of SuperBIB (Level-0, Diagram-0).

It may return pointers (markers, WHICH ENTRY) that point to entries matching all search words, if applicable.

An ideal bibliographic system should produce a bibliography in various formats. Process **FORMAT BIBLIOGRAPHY** is to accomplish this task to a certain degree. **FORMAT BIBLIOGRAPHY** gets reference entries from **REFERENCE FILE**, **SORTED REFERENCE FILE**, or **INQUIRY-RESULT FILE**. It gets

the user's choice for bibliographic style (BIB STYLE), and maintains a file, BIBLIOGRAPHY, which holds the formatted bibliographic information. The bibliography may be displayed on the terminal, printed on a printer, or stored in a file.

SuperBIB should allow the user to modify entries. A process, MODIFY REFERENCE ENTRIES, is to accomplish this task. For a given entry which is marked by WHICH ENTRY, MODIFY REFERENCE ENTRIES retrieves all, or a part, of entry elements according to a user's choice (WHICH ELEMENT), and accepts replacement reference entries (REPLACEMENT ENTRIES).

SuperBIB should sort the reference entries by the authors' last names and publication dates. SORT REFERENCE ENTRIES is to accomplish this task. SORT REFERENCE ENTRIES gets sorting criteria (SORT BY), sorts the reference entries in the reference file, and maintains a sorted reference file (SORTED REFERENCE). The sorted entries may be routed to a terminal screen, a printer queue, or a file of the user's choice. Each subprocess of the level-0 DFDs is discussed below.

2.5.1.3. ADD REFERENCE ENTRIES

When adding a reference entry to a reference file, the user may want to add a regular book entry, an article entry from a journal, an article entry from a book, a technical report entry, an article entry from conference proceedings, a compiled book entry, a multi-volume series entry, etc.

Figure 2-5A is a Data Flow Diagram (Level-1, Diagram-1) of ADD REFERENCE ENTRIES. Since there are seven major types of publication, there are seven subprocesses in ADD REFERENCE ENTRIES. These subprocesses are:

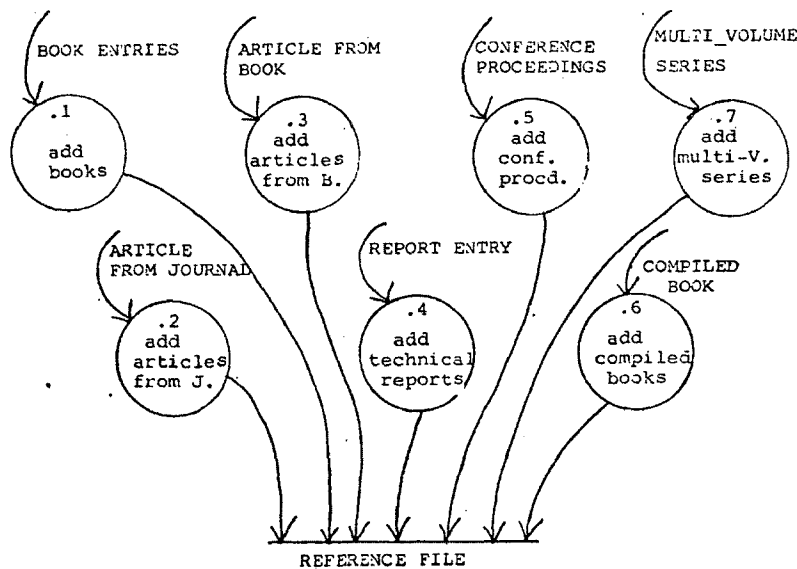


Figure 2-5A Logical Data Flow Diagram of ADD REFERENCE ENTRIES (Level-1, Diagram-1).

- (1) ADD BOOK, accepts, formats and stores elements for a book entry in the reference file.
- (2) ADD ARTICLE FROM JOURNAL, accepts, formats and stores an article entry from a journal.
- (3) ADD ARTICLE FROM BOOK, accepts, formats and stores an article entry from a book.
- (4) ADD TECHNICAL REPORT, accepts, formats and stores elements for a technical report (or M.S. thesis) entry.
- (5) ADD CONFERENCE PROCEEDINGS, accepts, formats, and stores elements of a conference proceedings entry,
- (6) ADD COMPILED BOOK, accepts, formats, and stores elements of a book entry which is compiled by editor(s).
- (7) ADD MULTI-VOLUME SERIES, accepts, formats, and stores elements of a multi-volume series entry.

The lower-level Data Flow Diagrams of ADD REFERENCE ENTRIES (Figures 2-5B to 2-5K) are listed in Appendix A.

2.5.1.4. INQUIRE REFERENCE ENTRIES

Figure 2-6 is a Data Flow Diagram (Level-1, Diagram-2) of INQUIRE REFERENCE ENTRIES. SuperBIB should be able to search for and retrieve entries in the reference file. INQUIRE REFERENCE ENTRIES is to accomplish this task. INQUIRE REFERENCE ENTRIES may be subdivided into three subprocesses, MAINTAIN INVERTED INDEX, PARSE SEARCH WORDS and RETRIEVE MATCHING ENTRIES.

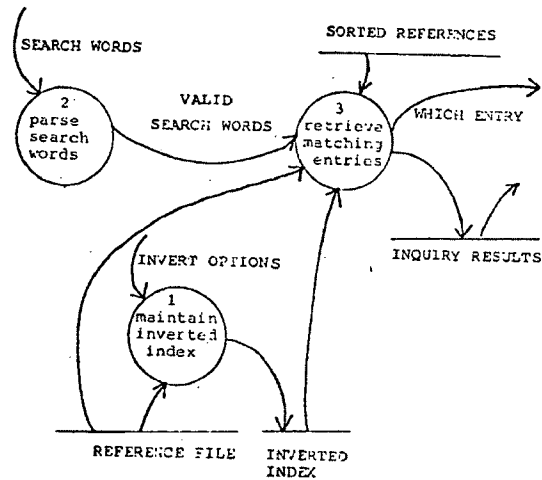


Figure 2-6 Logical Data Flow Diagram of INQUIRE REFERENCE ENTRIES (Level-1, Diagram-2).

PARSE SEARCH WORDS parses the search words supplied by the user, and returns valid search words, if applicable. For quick search, INVERT REFERENCE FILE gets invert options (i.e. truncating search words to six characters, etc.), maintains an inverted index for the reference file. RETRIEVE MATCHING ENTRIES utilizes REFERENCE FILE, INVERTED INDEX, or SORTED REFERENCE FILE. It retrieves reference entries that match all search words, and maintains a file, INQUIRY-RESULT FILE, which holds the entries searched and selected. It also returns pointers (WHICH ENTRY) that point to entries matching the search words, if applicable.

2.5.1.5. FORMAT BIBLIOGRAPHY

Figure 2-7 is a Data Flow Diagram (Level-1, Diagram-3) of FORMAT BIBLIOGRAPHY. SuperBIB formats entries into the bibliography in a few basic formats. It also routes the bibliography to the terminal screen, a printer queue, or a file of the user's choice. FORMAT BIBLIOGRAPHY is to accomplish this task. FORMAT BIBLIOGRAPHY may be subdivided into two subprocesses. These subprocesses are FORMAT ENTRIES and ROUTE FORMATTED ENTRIES. FORMAT ENTRIES gets reference entries from REFERENCE FILE, SORTED REFERENCE FILE, or INQUIRY-RESULT FILE, gets the user's choice for bibliographic style (BIB STYLE), and formats the reference entries to bibliographic entries (FORMATTED ENTRIES). ROUTE FMT ENTRIES routes the formatted entries to a terminal screen, a printer queue, or a file of the

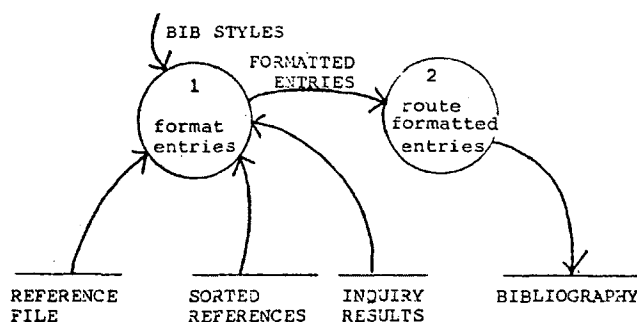


Figure 2-7 Logical Data Flow Diagram of FORMAT BIBLIOGRAPHY (Level-1, Diagram-3).

user's choice.

2.5.1.6. MODIFY REFERENCE ENTRIES

Figure 2-8A is a Data Flow Diagram (Level-1, Diagram-4) of MODIFY REFERENCE ENTRIES. SuperBIB allows the user to modify (alter) a reference file, if he desires. MODIFY REFERENCE ENTRIES is to accomplish this task. MODIFY REFERENCE ENTRIES may be subdivided into seven subprocesses [2-6]. These subprocesses are:

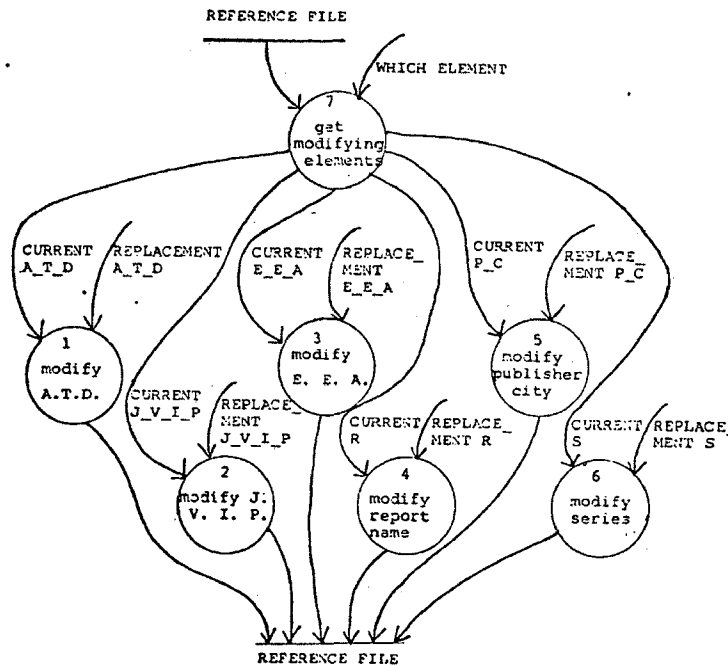


Figure 2-8A Logical Data Flow Diagram of MODIFY REFERENCE ENTRIES (Level-1, Diagram-4).

- (1) GET MODIFYING ELEMENTS, gets the entry elements that are to be modified, for a given entry (WHICH ENTRY), and according to the user's choice (WHICH ELEMENT),
- (2) MODIFY AUTHOR/TITLE/DATE, modifies authors' names, titles, publication dates.
- (3) MODIFY JOURNAL/VOLUME/ISSUE/PAGE, modifies journal names, volume numbers, issue numbers, page numbers.
- (4) MODIFY EDITOR/COMPILED BOOK/ ARTICLE FROM BOOK, modifies editors' names, compiled book titles, article entries from books.
- (5) MODIFY REPORT NAME, modifies technical report names.
- (6) MODIFY PUBLISHER/CITY, modifies publishers, publishing cities.
- (7) MODIFY SERIES, modifies multi-volume series.

The lower-level DFDs of MODIFY REFERENCE ENTRIES (Figures 2-8B to 2-8E) are listed in Appendix A.

2.5.1.7. SORT REFERENCE ENTRIES

Figure 2-9 is a Data Flow Diagram (Level-1, Diagram-5) of SORT REFERENCE ENTRIES. SuperBIB sorts the reference entries. It also routes the bibliography to the

[2-6] Note that this analysis applies to both modifying methods - modifying by invoking text editor, or by answering questions. In both modifying methods, MODIFY AUTHOR/TITLE/DATE marks the current elements as unused, and gets the replacement element.

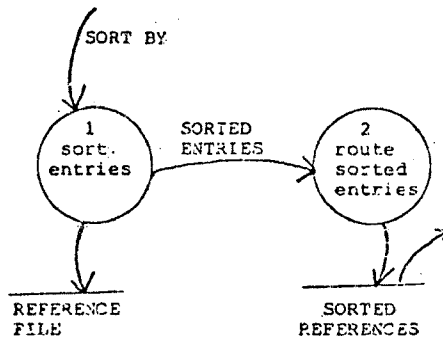


Figure 2-9 Logical Data Flow Diagram of SORT REFERENCE ENTRIES (Level-1, Diagram-5).

terminal, a printer queue, or a file of the user's choice. SORT REFERENCE ENTRIES is to accomplish this task. SORT REFERENCE ENTRIES may be subdivided into two subprocesses. These subprocesses are SORT ENTRIES, and ROUTE SORTED ENTRIES. SORT ENTRIES gets sorting criteria (SORT BY), sorts the reference entries. ROUTE SORTED ENTRIES routes the sorted reference entries to a terminal screen, a printer queue, or a file of the user's choice.

2.5.2. PHYSICAL DATA FLOW DIAGRAM OF SUPERBIB

Once some physical constraints have been added, or the area to be automated is determined, the Level-1 Logical DFDs of SuperBIB is turned into a Physical DFD. Briefly, the SuperBIB functionalities which are not automated are listed below.

- (1) The capability to accept more than one paragraph of annotations is not implemented.
- (2) Only the entries that contain all of the search words will be retrieved. The wildcard searching and search for entries that satisfy a portion of the search words are not implemented.
- (3) The bibliography in a few basic formats is provided [2-7].
- (4) The capabilities of modifying a reference file using ex or edit editor, or modifying without invoking a text editor (by questions and answers) is not developed.
- (5) Sorting the reference entries by other sorting criteria beyond the senior author's last name and publication dates, is not developed.

2.5.3. DATA DICTIONARY PROCESSOR

After the DFDs of SuperBIB were done, the ISDOS [2-8] Problem Statement Language/Problem Statement Analyzer (PSL/PSA) was used to produce supporting documents, such as a Data Dictionary and Transform Descriptions (Min-

[2-7] To produce a bibliography with a variety of other formats, and to have better searching capabilities, refer to "A Unix Bibliographic Database Facility" by Budd, and "Writing Papers with Nroff Using -Me" by Allman, and the Unix Programmer's Manual. In summary, if the user creates (or maintains) a reference file, inverts (or indxbib) the file, creates (or maintains) an Nroff text file with imprecise citations, invokes Bib command, the user will obtain the full power of the Unix bibliographic system.

ispecs).

Manual procedures to maintain a Data Dictionary are error-prone, redundant, repetitious and time-consuming. A program managing the DD is called a Data Dictionary Processor. The ISDOS Data Dictionary Processor allows its user to describe the data flow and process in an English-like language called Problem Statement Language (PSL). Then these descriptions are analyzed for completeness and consistency by the Problem Statement Analyzer (PSA).

The supporting documents for SuperBIB were done using PSL under the System Encyclopedia Manager (SEM) in the Unix environment. Every bubble in the Data Flow Diagram is a PROCESS in PSL/PSA. Every PSL/PSA PROCESS was described by a short description, an algorithm (if applicable), the input and the output. These may serve as a Transform Description (Minispecs) for the primitive processes. The Data Dictionary was compiled using DESCRIPTION for each ELEMENT, ENTITY, INPUT, OUTPUT and SET of the SuperBIB system where SET was used to define files, INPUT and OUTPUT were used to define the net data flow (the input coming from the data originator box, or the output going into

[2-8] ISDOS is the name of a project conducted at the University of Michigan by Dr. Daniel Techroew.

the data receiver box), and ENTITY was used to define the complex data flow which consists of the elementary component, ELEMENT. INTERFACE was used to define the data originator or receiver (the boxes in DFD).

Two PSL/PSA reports were produced, the Formatted Statements Report and the Name Selection Report. The Formatted Statements Report provides us with the Data Dictionary and Transform Descriptions (Minispecs). The Name Selection Report gives us a listing of all data items. The Formatted Statements Report in partitioned format is not available because TRACE-KEY in the current installation of SEM does not work.

The management of the Data Dictionary and Transform Description for a medium to large-sized software project is not easy. The ISDOS PSL/PSA reports provide the Data Dictionary and Transform Description in an efficient way. The ISDOS PSL/PSA enters only non-redundant information into its database. It is relatively easy to change an analysis document generated by a Data Dictionary Processor (such as PSL/PSA) during the life cycle of software development, due to the fact that the information stored in the PSL/PSA database is not redundant and one modification made to the database changes all related information.

The consistency checking, cross-reference listing and alias control are the other benefits of using PSL/PSA.

However, ISDOS PSL/PSA is not ideal. Its costs are high. It requires a certain amount of initial training in order to use it. It generates an enormous number of reports and sometimes it is hard to utilize the reports. The author feels that the ISDOS PSL/PSA Data Dictionary Processor may be a necessity to manage a large-sized software development, although its effectiveness in this thesis project was not significant.

2.6. TEMPLATES

Each type of reference entry contains a slightly different set of information. For example, a regular book entry may contain authors' names, a title, a publication date, a publisher, a publishing city, an abstract, comments, search words (key-words), etc. However, an article entry from a journal may contain a journal title, a volume number, an issue number, page numbers, plus all the information for a regular book entry. In other words, each type of publication must have its own template. The templates of all publication types are listed below:

(1) The template of a regular book is:

Author's Name
Title
Publisher
City
Date
Search Words
Others
Annotation

(2) The template of an article from a journal is:

Author
Title
Journal Title
Volume Number
Issue Number
Page Number
Date
Search Words
Others
Annotation

(3) The template of an article from a book is:

Author's Name
Title
Book Title
Editor's Name
Page Number
Publisher
City
Date
Search Words
Others
Annotation

(4) The template of a technical report is:

Author's Name
Title
Report Title
Publisher
City
Date
Search Words
Others
Annotation

(5) The template of an article from a conference proceedings is:

Author's Name
Title
Journal Title
Page Number
Date
Search Words
Others

(6) The template of a compiled book is:

Editor's Name
Book Title
Publisher
Date

(7) The template of a multi-volume series is:

Author's Name
Title
Book Title
Editor's Name
Page Number
Series Title
Volume Number
Publisher
City
Date

2.7. ANALYSIS OF HUMAN-MACHINE INTERACTION

Due to the fact that this project was to develop a preprocessor (user interface) for Unix-bib, Structured Analysis of SuperBIB was complemented by an analysis in the human-machine interaction area. The analysis of the human-machine interaction in this thesis project was done by developing a menu tree, a hierarchy of the menus, and a draft of the SuperBIB User's Guide [2-9].

2.7.1. MENU TREE

Figure 2-10 illustrates a hierarchy of SuperBIB menus. A brief description of these menus is listed below.

[2-9] A copy of SuperBIB User's Guide is in Appendix B.

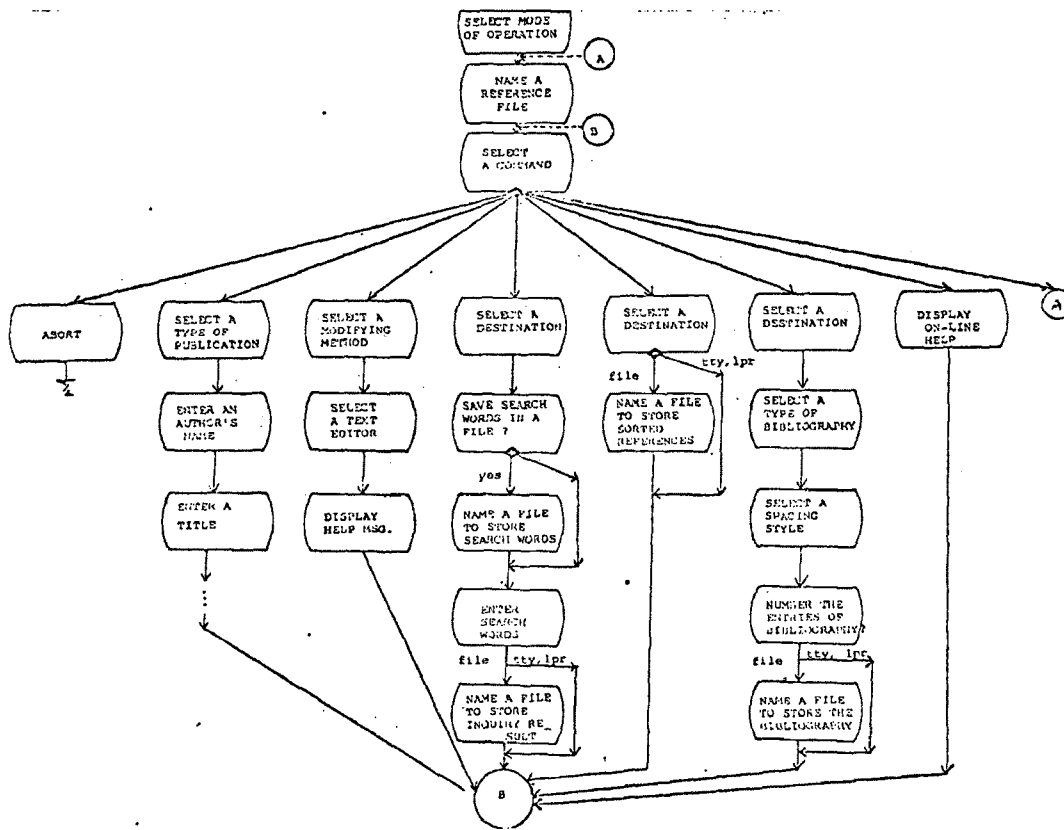


Figure 2-10 The SuperBIB Menu Tree

- (1) SELECT A MODE OF OPERATION. Separate modes of operation for the novice and sophisticated users are desired (but not required). In this way the sophisticated users would not be bothered by excessive information, and the novice user can get a smooth start in the SuperBIB operation. (Note that a novice mode of operation has higher priority over the sophisticated one.)
- (2) NAME A REFERENCE FILE. The user would like to name a reference file early in the SuperBIB operation and keep using this file until the user invokes a different reference file, or a sorted reference file is created [2-10].
- (3) SELECT A COMMAND, allows the user to select one of the following commands:
 - ADD, adds reference entries to the reference file.
 - INQ, inquires of the reference entries.
 - PRINT, formats and prints the bibliography.
 - EDIT, modifies the reference file.
 - SORT, sorts the reference entries.
 - INVOKE A NEW REFERENCE FILE.
 - HELP, prints on-line help.
 - EXIT, exits SuperBIB.
- (4) SELECT A PUBLICATION TYPE, allows the user to select one of the following publication types.
 - BOOK,
 - ARTICLE FROM JOURNAL,
 - ARTICLE FROM BOOK,
 - TECHNICAL REPORT,
 - ARTICLE FROM CONFERENCE PROCEEDINGS,
 - COMPILED BOOK,
 - MULTI-VOLUME SERIES,
 - OTHERS, for unpublished materials.

[2-10] Reference Filename is Sorted Reference Filename if the user maintained a file holding the sorted reference entries. Select INVOKE A REFERENCE FILE at the SELECT A COMMAND menu to change it, if the user desires.

- (5) ENTER AN AUTHOR'S NAME, allows the user to enter an author's name.
- (6) MORE AUTHORS' NAMES? This is to allow the user to enter multiple authors' names. (Note that the menus for entering other elements are similar, and are omitted.)
- (7) SELECT A DESTINATION, allows the user to select a destination for routing the inquiry results, bibliography, or sorted reference entries.
- (8) NAME A FILE TO STORE SEARCH WORDS, if applicable.
- (9) TYPE SEARCH WORDS. Separate keys with spaces.
- (10) SEARCH-WORD FILE AVAILABLE?
- (11) NAME YOUR SEARCH-WORD FILE.
- (12) NAME A FILE TO STORE THE INQUIRY RESULTS, if applicable.
- (13) SELECT A BIBLIOGRAPHY TYPE, selects the type of the bibliography, such as regular or annotated bibliographies,
- (14) SELECT A SPACING STYLE, selects a spacing style, such as single spaced, or double spaced, etc.
- (15) NUMBER THE BIBLIOGRAPHY, gives the user an option to number the bibliography,
- (16) NAME A FILE TO STORE THE BIBLIOGRAPHY, if applicable.
- (17) SELECT A MODIFYING METHOD modifies a reference file by using a text editor, or by answering questions.
- (18) SELECT AN EDITOR, selects an editor to modify the reference file.
- (19) DISPLAY HELP MESSAGE ABOUT EDITORS, displays brief help messages about how to use vi, ed editors.
- (20) DISPLAY MESSAGE ABOUT SORT KEYS, informs the user what kind of sort keys he may use, etc.

- (21) NAME A FILE TO STORE SORTED ENTRIES, if applicable.
- (22) PRINT ON-LINE HELP MESSAGES, such as 'To advance the cursor, press space bar'.
- (23) EXIT PREMATURELY, terminates the SuperBIB program if the user misuses the system, such as does nothing but keep naming reference files. (Note that this has a low priority.)

2.7.2. SAMPLES OF SUPERBIB MENUS

Samples of SuperBIB menus were illustrated in Tables 2-5 and 2-6.

The menu for NAME A REFERENCE FILE is illustrated below:

Table 2-5 The NAME A REFERENCE FILE Menu

```

=====
*****
* SuperBIB-Super Bibliographic System *
* * * * * * * * * * * * * * * * * * * *
* <<< NAME A REFERENCE FILE. >>> *
* * * * * * * * * * * * * * * * * * * *
* REF. FILE : [ ] *
* ----- *
* * * * * * * * * * * * * * * * * * * *
*****

```

Note that 'SuperBIB - Super BIBliographic System' is a system greeting. A symbol, [], indicates the cursor position when the menu first appeared. The dashed line indicates the maximum length of the inputs.

```
=====
```

The menu for SELECT A COMMAND is illustrated below:

Table 2-6 SELECT A COMMAND Menu

```

=====
*****
* SuperBIB-SUPER BIBliographic System *
*                                     *
*   <<<  SELECT A COMMAND.  >>>     *
*                                     *
*       [ ]  ADD ?                   *
*            INQUIRE ?               *
*            PRINT ?                  *
*            EDIT?                    *
*            SORT?                    *
*            INVOKE A REFERENCE FILE? *
*            HELP ?                   *
*            EXIT ?                   *
*                                     *
* To advance cursor, press space bar. *
* To select a command, press RETURN key.*
*                                     *
* ADD INQUIRE PRINT EDIT SORT INVOKE HELP *
*****

```

Note that the last line on the menu is the contents of a status report.

2.8. PROBLEMS ENCOUNTERED

Relatively minor problems were encountered in the derivation of Data Flow Diagrams. The author had difficulties in excluding the control information, such as executive components, etc., in the derivation of DFDs for SuperBIB.

Two other problems were encountered in the analysis phase for SuperBIB. One of these problems derived from the fact that the project commissioner and the author perceived the Unix-bib system differently. The author thought SuperBIB not only would be a preprocessor for producing a Unix reference file, as the project commissioner requested, but also a preprocessor for Bib, which is a bibliographic preprocessor for Nroff, a Unix typesetting and text processing system.

Another problem was that the analyst, the author, did not take into account the fact that predominantly this project was to develop a preprocessor (user interface) for Unix-bib. She did not precisely specify what the user wanted in the aspect of human-machine interaction of SuperBIB, although the functionality of SuperBIB was specified quite completely.

2.8.1. PROBLEMS OF DERIVING DATA FLOW DIAGRAMS

As De Marco pointed out, trivial error paths, initializations and terminations are not supposed to be in the DFD. During the development of the DFD, a systems analyst should not consider control information, such as executive process, decision-making or repetition. However, the attempt to exclude control information in the

derivation of SuperBIB DFDs was not very successful because the author was used to flow-charts and procedural thinking. The author had to revise the DFD several times to do so.

2.8.2. DIFFERENT VIEWS OF UNIX-BIB

At the early analysis stage, the author discovered a user-contributed (non-standard) Unix bibliographic preprocessor, Addbib, which was unknown previously to the project commissioner, due to the fact that Unix has an enormous number of user-contributed software tools. Addbib prompts its user for input leading the user to create a reference file, or to append reference entries into an existing reference file. To do so, the knowledge of the rigid format of Unix-bib reference files is not required. Addbib is quite user-friendly, despite the fact that it cannot accept annotations, (although its documentation claimed it would). Since the author knew of the existence of Addbib, and other software tools, such as Invert, (or Indxbib), Lookupbib, (or Lookup), Roffbib and Sortbib, and the quality of these software tools generally meet the requirements of the proposed SuperBIB system, she thought a large portion of work was already done, and she should not duplicate it. She assumed wrongly that this project

was to develop a user interface for Bib, a bibliographic preprocessor for Nroff, since this seemed to be the only useful feature which is unavailable in a human engineered (user friendly) fashion.

Nroff is a text processing facility available on the Unix operating system. According to "Writing Papers with NROFF Using -Me" by E. Allman, Nroff reads an input text file prepared by its user, and returns a formatted text suitable for publication. The input consists of text, and requests, which give instructions to the Nroff program telling how to format the text. According to "A UNIX Bibliographic Database Facility" by T. Budd,

- (1) Bib is a preprocessor to the Nroff (or Troff) typesetting systems. Bib is a program for collecting and formatting citation strings in documents. It takes two inputs: a Nroff text document and a reference file. Imprecise (incomplete) citation strings in the text document are replaced by more conventional citation ones. The appropriate references are selected from the reference file, and commands are generated to format both traditional citation strings and the stand-alone bibliography.
- (2) An imprecise citation string is a list of words surrounded by "[" and ".". Words (which are truncated to six letters) in the imprecise citation string are matched against reference entries in the reference file. If an entry is found that matches all words, that reference is retrieved to be included in the bibliography and the imprecise citation string of text document is replaced by a traditional one.

For example, an imprecise citation string of "[. brooks mythical man-month .]" retrieves the book entitled The Mythical Man-month by F. Brooks from the reference file. A more conventional citation string, such as [1], will replace this imprecise citation in the text document, and this book entry will be included in the bibliography.

The author's misunderstanding about the scope of this project went undetected until she had spent fruitless weeks trying to derive a set of satisfactory Logical DFDs for SuperBIB. The author's language barrier was perhaps one contributing factor to the prolonged misunderstanding, since English is her second language. Problems with the terse terminology in the Unix documents were another factor. For example, a crucial term, FORMAT, meant different things to the project commissioner and the author. When the project commissioner mentioned FORMAT, he meant, 'to format a reference file which meets the requirements of the Unix bibliographic systems'. When the author mentioned FORMAT, she meant, 'to format an Nroff text file so that precise citation strings and a stand-alone bibliography may be produced'.

But, neither the language barrier nor the misunderstanding of these particular terms accounts completely for

the author's problem. In fact, such disagreement about existing systems is common for software projects because of the complexity of existing software systems, inadequate documentation, unstructured source code, etc.

At the end of the design stage the author was surprised to find out that only a binary code version of Addbib is available in this Department, since Addbib is not standard Unix software. The lack of source code meant that necessary modifications on Addbib were impossible. The author did not know Unix well enough, lacked experience, and was too confident to check the availability of Addbib source code at an early analysis stage. If the author had done so, she might have discovered her misunderstanding much earlier.

2.8.3. SOLUTIONS TO DIFFERENT-VIEW PROBLEM

At the early stages of Unix-bib analysis, the author tried to understand the current Unix-bib operations. However, she overlooked the need to document the current operations in Physical and Logical DFDs as De Marco suggested. In fact she did not document the operation in any presentable (pleasant) form, and did not verify these documents with the expert user-project commissioner when it should be done. If the author had documented Unix-bib

as De Marco suggested at the early stage of analysis, and walked through the documents with the expert user-project commissioner, the different views of Unix-bib would have been detected before any attempt was made to develop the Logical DFDs for SuperBIB.

2.8.4. CHARACTERISTICS OF A USER INTERFACE PROJECT

Structured Analysis techniques did a good job in analyzing the functionality of SuperBIB. However, to analyze a predominantly user-interface software project completely, an analysis in the human-machine interaction area must be done.

De Marco's Data Flow Diagram techniques do not analyze the human-machine interaction very well, due to the fact that the presence of control information, such as executive modules, decision-making and repetition, are necessary. The analysis of human-machine interaction may be done by developing certain documents such as a menu tree, a hierarchy of menus, or a draft of a User's Guide. The menu tree for this thesis project was not formally developed during the analysis stage when it is most appropriate. Although a skeleton of the menu tree was developed during the prototyping stage of the design stage, this menu tree was not verified by user

representatives, and the user feedback was not utilized when it should have been.

2.8.5. CONCLUSIONS

The author is convinced that the analysis phase is the most important phase in the development of the software project. She agrees that De Marco's methods are valuable for analyzing the functionality of any software engineering project, except the simplest ones. A system analyst must document the current operation, and representatives of the potential users must verify, and approve the documentation before starting the analysis of a proposed system.

In the case of a predominantly user interface project, De Marco's Structured Analysis must be complemented by an analysis in the human-machine interaction area. A statement of "SuperBIB must be menu-driven, screen oriented and employ windows" is not precise enough for the system designer to architect the system without pondering what the user really wants in the human-machine interaction area. Based on the experience of this thesis project, the author found that the menu tree and a draft of the User's Guide had served well in this respect.

CHAPTER 3

DESIGN

The design phase of software development is almost as important as the analysis phase. After a systems analyst has identified what has to be done to solve the problem at hand, a systems designer must determine how the solution can be implemented. When the software design phase begins, the systems analyst has already completed a precise document prescribing what has to be done to solve the problem. By studying the document, a systems designer understands the problem to be solved, an outline of a solution, the requirements of a new system, and the basic structure of the new system. To reduce the complexity, a systems designer partitions the new system into modules. The interfaces among the modules are also established in this phase. Once the analysis and design phases are done properly, the implementation is straightforward.

The fundamentals of software design, the principles of a good software design, Structured Design methodology, architectural design and prototype for SuperBIB, and a review of the SuperBIB design are discussed in this chapter.

3.1. FUNDAMENTALS OF SOFTWARE DESIGN

3.1.1. WHAT IS DESIGN?

According to Design Methods by J. C. Jones, design is a case of decision making in the face of uncertainty, with high penalties for error [Jones,1983]. Software design is a simulation of what a software developer wants to do before he does it. To feel confident in the final result, as many simulations as necessary may be performed.

As G. Bergland pointed out in "Tutorial: Software Design Strategies" [Bergland,1981]:

None of the existing software design techniques truly gives a procedure that can be followed step by step, from start to finish, like a recipe in a cookbook. The design techniques represent alternative plans of attack whose success or failure is (in large measure) determined by the skill and experience of the designer. A design strategy may work well for one class of problems but may fail miserably for another.

3.1.2. SOFTWARE COSTS AND DESIGN

According to Software Engineering Economics by Barry W. Boehm the cost of software is escalating; by 1985 the estimated ratio of software to hardware costs will be nine to one [Boehm,1981]. Because the design of many programs is less than satisfactory, or other reasons, these

programs are difficult to code, debug, test, modify, or maintain, and the life-cycle costs of these programs is high. According to Boehm, 40 % to 60% of software money is spent on the maintenance, modification and continued debugging of production programs. According to Structured Analysis and System Specification by Tom De Marco, for every dollar spent in true development (designing and coding), three dollars are spent in revision (debugging and testing) either before or after delivery [DeMarco, 1978]. Therefore, a good software design is a necessity to control the life-cycle costs in a software engineering project.

3.2. PRINCIPLES OF GOOD SOFTWARE DESIGN

A successful software design matches the structure of the problem which is to be solved. A systems designer is supposed to deal with problems in order of importance instead of in order of execution. A good software design results in a set of small and independent modules. These modules are arranged in a hierarchy with the major modules at the top of the hierarchy and the detailed modules at the bottom.

3.2.1. TOP-DOWN APPROACH

The top-down approach is a variation of Julius Caesar's "divide and conquer" strategy. According to "Top-down Design and Testing" by E. Yourdon, the top-down approach has been referred to as "systematic programming," "stepwise refinement," "levels of abstraction," "functional decomposition," and a variety of other names [Yourdon, 1981]. As Yourdon pointed out, three related, but distinct, aspects of top-down approach are these:

- (1) Top-down design: a design strategy that breaks large, complex problems into smaller, less complex problems - and then decomposes each of those smaller problems into even smaller problems, until the original problem has been expressed as some combination of many small, solvable problems.
- (2) Top-down coding: a strategy of coding high-level, executive modules as soon as they have been designed - and generally before the low-level, detail modules have been designed.
- (3) Top-down testing: a strategy of testing the high-level modules of a system before the low-level modules have been coded - and possibly before they have been designed.

According to Yourdon,

there are two extremes of top-down approach. One is conservative top-down and another is radical top-down. In the case of conservative top-down approach, the software developer designs all level

of modules before he implements any one of them. In the case of radical top-down approach, the developer designs the top level of a system and immediately implements this module before designing any other lower-level modules. Normally a software developer takes a middle-of-the-road stance, depending on particular software project characteristics. If a software developer has a very tight schedule to complete his work, the developer may adopt a method which leans toward the extreme of the radical top-down. If a software developer has to estimate the costs, or schedule for a software project accurately, he is likely to adopt a method which leans toward the extreme of the conservative top-down. Normally a software developer may design 50% to 75% of modules before he implements any of these.

A bottom-up approach would be a practice where the software developer works on all of the bottom-level modules first, then the intermediate-level modules. The upper-level modules are worked out last. Normally the top-down approach is better than the bottom-up because the top-down approach encourages the software developer to work out the important modules first. By using the top-down approach the complexity of the problem is reduced and the user's needs are likely to be better served. However, to ensure the feasibility of development in critical modules, lower-level modules may be developed before the higher-level modules.

One important thing about the top-down approach is that the software developer does not start coding once the

analysis phase is over. The developer has to do a certain amount of design before he can start to implement any module, even in the case of a radical top-down approach.

3.2.2. TOP-DOWN DESIGN

As G. Bergland pointed out in Tutorial: Software Design Strategies [Bergland, 1981]:

Top-down design encourages the software developer to start a design simply by defining one "super" module that will solve the whole problem and then implementing that module with less and less abstract semi-super modules. Sooner or later, the developer gets down to modules that will actually execute the tasks. This procedure is usually called Functional Decomposition, meaning that the main function is decomposed into successively simpler and simpler components. Alternatively, one can think of performing Stepwise Refinement, meaning that the solution is successively refined into more and more detailed explanations of how that solution is to be brought about. In either case, the parallel objective is to identify reusable functional modules wherever possible.

Beyond the above functional decomposition, the modules have to be relatively independent. Coupling, namely the quantity and complexity of data passed between modules, has to be minimized; and cohesion, namely the binding among the program statements within a module, has to be maximized. And these modules have to be dealt with in order of importance, not in the order of execution sequence.

3.2.3. EASE OF EXPANSION AND CONTRACTION

The ability of a software system to expand and contract as the user's needs evolve is an indication of good software design. As D. Parnas pointed out in "Designing Software for Ease of Extension and Contraction" [Parnas, 1981],

When designing a software system, there is a tendency to attempt to develop the software for the problem as if there were only one specific problem and one program to do that job. Instead, one ought to design software in a manner that economically permits new features to be added or unused features to be removed for efficiency.

In summary, the principles of a good software design is to partition the system into smaller, independent, and reusable modules which are arranged in a hierarchy of importance. The high cost of software maintenance can be controlled if software developers practice these principles.

3.3. A SUMMARY OF THE SUPERBIB REQUIREMENTS

In summary, Dr. Barr and the potential SuperBIB users wanted SuperBIB to do these things:

- (1) Accept citations (authors' names, titles, publication dates, etc.), annotations (abstracts, com-

ments), and other information (library call numbers, secondary reference, etc.) in a user-oriented fashion; Ideally, SuperBIB is menu-driven, is screen-oriented, and employs windows.

- (2) Search for and retrieve any particular reference entry matching the search words, such as authors' names, titles, publication dates, other search words (key-words), or a combination of these. Entries which match all the search words will be selected. (Wildcard searching and selecting entries which match a portion of the search words are desired, but not required.)
- (3) Print the bibliography in basic formats. (The capability of formatting a bibliography in a variety of formats is desired, but not required.)
- (4) Modify the reference information using a text editor, if desired. (The capability of modifying without using an editor is desired, but not required.)
- (5) Sort reference entries by the senior author's names and publication dates. (Sorting by other criteria is desired, but not required.)

3.4. STRUCTURED DESIGN AND ITS APPLICATION

There are a number of software design strategies, such as the Structured Design Methodology (Data Flow Design), the Jackson Design Methodology, and the Warnier-Orr Design Methodology, etc. SuperBIB was designed by using the Structured Design methodology, due to the fact that SuperBIB was analyzed by using Data Flow Diagram techniques, and Structured Design is a strategy to convert the Data Flow Diagrams into a suitable design document,

the Structure Chart.

The Structured Design is a set of considerations for making coding, debugging and modification easier, faster, and less expensive by reducing complexity [Yourdon,1979]. According to Structured Analysis and System Specification by T. De Marco [DeMarco,1978], the derivative techniques of Structured Design are Transform Analysis and Transaction Analysis (to be explained). Its refinement techniques are design heuristics, such as coupling and cohesion. The documentation tool for the Structured Design is the Structure Chart.

3.4.1. STRUCTURE CHART

The Structure Chart is used to document the SuperBIB design [3-1]. A Structure Chart illustrates modules of the system, interfaces among modules, and the hierarchy of a system. The three basic elements of the Structure Chart are these:

- (1) The module, represented by a rectangular box with a module name inside.

[3-1] See Figures 3-1, 3-2 for examples of the Structure Chart.

- (2) The module connection, represented by a vector, -->, joining two modules; usually the connections mean one module has called the other.
- (3) The data passed between modules, represented by a short arrow with a circular tail, o-->.

The Structure Chart of design and the Data Flow Diagram of analysis are similar. They both emphasize the partition and the interface. The major difference between a Data Flow Diagram and a Structure Chart is that a structure chart has an executive (a driver) module. In a Data Flow Diagram, there is no executive. In addition, a Data Flow Diagram is a document prepared by a systems analyst; it identifies what has to be done to solve the problem. A Structure Chart is a document prepared by a systems designer; it spell out how the requirements of a new system shall be met.

3.4.2. TRANSFORM ANALYSIS

The Transform Analysis applies to linear Data Flow Diagrams that have clearly identified afferent (input) streams, central processing, and efferent (output) streams. According to Structured Design by E. Yourdon and L. Constantine [Yourdon, 1979], an afferent data element is a high-level element of data that is furthest removed from physical input, but still constitutes input to the

system. In other words, an afferent data element of a data flow diagram is an incoming datum which is in the most sophisticated form of transformation before processing. For example, a reference entry in a reference file is an afferent data element for SuperBIB because a reference entry is an input and it has been transformed thoroughly by prefixing a percent sign (%), an appropriate key-letter, and a space, but it has not been used for processing yet. An efferent data element is a data element that is furthest removed from the physical outputs but still may be termed outgoing. In other words, an efferent data element of a data flow diagram is an outgoing datum which is in the least sophisticated level of transformation into an output.

To design a system using Structured Design techniques, a systems designer converts the Data Flow Diagrams of the analysis phase to the Structure Chart of the design phase. As the author mentioned earlier (Section 3.4.1), the major difference of a DFD and a Structure Chart is that a Structure Chart has an executive module, but a DFD does not. Therefore, it seems logical to find an executive process if the designer wants to convert a DFD to a Structure Chart.

To initiate the conversion, a systems designer needs to determine the central transform, an executive process, of a system. The central transform is the process between the afferent and efferent sides of the Data Flow Diagram. The afferent module is the module on the afferent (input) side, and is concerned with the function of accepting or developing the system input. The efferent module is the module on the efferent (output) side, and is concerned with delivering system output. For example, ADD REFERENCE ENTRIES and MOD REFERENCE ENTRIES are afferent modules, FORMAT BIBLIOGRAPHY is an efferent module, and INQUIRE REFERENCE ENTRIES is the central transform for SuperBIB.

Once the central transform is determined, a systems designer is to determine a top executive (driver) module of the system. The central transform module could be a driver. A systems designer may create a totally new module to be a driver. Once the driver is determined, a systems designer places this driver at the top of a structure hierarchy. The rest of the processes (bubbles) in the level-0 (diagram-0) Data Flow Diagram become immediate subordinates of this driver. A systems designer replaces the bubble of Data Flow Diagrams by the rectangular box of a Structure Chart, due to the fact that a rectangular box represents a process traditionally. He renames some the

rectangular boxes, if applicable. He also identifies the connections (calling and called relationships) between modules. When these are done, a first-cut of the Structure Chart is completed.

Figure 3-1 is the first-cut of the SuperBIB design. The author decided to create a totally new module, SUPER-BIB, to be an executive (a driver) module of the SuperBIB system because the central transform, INQUIRE REFERENCE ENTRIES, does not fit the role of an executive module. To add a reference entry into a reference file, there is no need to retrieve any entry from the file. If INQUIRE REFERENCE ENTRIES was the executive module, the

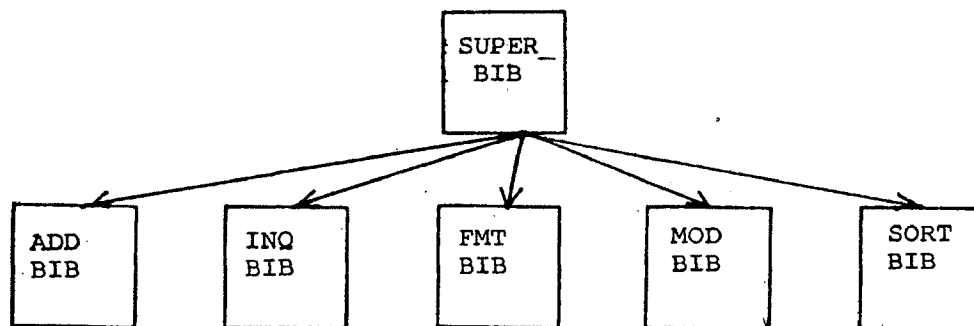


Figure 3-1 The First Cut of the SuperBIB Design.

relationship between ADD REFERENCE ENTRIES and INQUIRE REFERENCE ENTRIES would not exist.

Once the executive module, SUPERBIB, is decided, all level-0 bubbles of the SuperBIB Data Flow Diagrams become the immediate subordinates of SUPERBIB. The author replaced the bubble by a rectangular box. For simplicity and consistency with the current Unix bibliographic documents, ADD REFERENCE ENTRIES was renamed ADD BIB; INQUIRE REFERENCE ENTRIES was renamed INQ BIB; FORMAT BIBLIOGRAPHY was renamed FMT BIB, MODIFY REFERENCE ENTRIES was renamed MOD BIB, etc.

To proceed further in a software design, a systems designer subdivides the afferent and efferent modules further, if applicable.

For reason of clarity, a systems designer may add some symbols representing procedural information; a diamond-shaped symbol indicates a decision-making, and a circular arrow indicates a repetition, etc. [3-2]. For example, the executive module, SUPERBIB, must decide which subordinate module to call, hence there is a diamond-shaped symbol at the bottom of SUPERBIB to indicate this decision-making. A publication entry may be written by author and coauthors, hence ADD AUTHOR's NAME module may

be called more than once for each publication entry. A circular arrow by ADD AUTHOR's NAME module indicates this repetition.

3.4.3. TRANSACTION ANALYSIS

Transaction Analysis is a supplementary technique for Transform Analysis, and is valuable for a system (or sub-system) that processes transactions. The Data Flow Diagram of a transaction system appears to be a network (graph) in its shape, while the Data Flow Diagrams of a transform system tend to be linear.

According to E. Yourdon and L. Constantine a transaction is a stimulus to a system that triggers a set of activities [Yourdon, 1979]. Every transaction carries a tag (code) to indicate its transaction type. By referring to the tag, the system would determine what processing each transaction required. For example, ADD BIB is a system that processes transactions. The user's response regarding the type of publication entry is the stimulus (transaction) of ADD BIB. The types of transaction for ADD BIB is the type of publication entry, such as BOOK, ARTICLE FROM

[3-2] See SUPERBIB and ADD AUTHOR's NAME of Figure 3-2.

JOURNAL, ARTICLE FROM BOOK, etc.

According to Yourdon, four basic types of modules in the Transaction Analysis are these.

- (1) Transaction-center (transaction processor) Module, identifies the type of each transaction, and routes the transaction to an appropriate subordinate. For instance, ADD BIB is a transaction-center module for SuperBIB. (Note that a system may have more than one transaction-center module.)
- (2) Transaction Module, a module which only processes one type of transaction. There are as many transaction modules as there are transaction types. For example, ADD BOOK is a transaction module because it does not identify, or route any type of publications. It processes one type of publication, book entry.
- (3) Action Module, processes a part of work which is to be accomplished by its parent transaction module. For example, ADD AUTHOR'S NAME, and ADD TITLE are two action modules in ADD BOOK.
- (4) Detailed Module, processes a part of work which is to be accomplished by its parent action module. By invoking the same detailed module, similar types of transactions share common codes. For examples, ADD NAMES is a detailed module which is commonly needed in two action modules, ADD AUTHOR'S NAME and ADD EDITOR'S NAME.

The transaction-center module calls each of the transaction modules with no data at all. Each transaction module is responsible for obtaining its own input and delivering its own output.

The fundamental principle of the Transaction Analysis is to separate the various transactions by type and to process each transaction type separately regardless of how similar each transaction might be. The rules are to identify common functions, and to implement the common function in a small and independent module. This principle results in a modifiable and maintainable software system.

The advantage of SuperBIB's design is that the processing for all the publication types is separate, so that any change to the processing for one type of publication will not affect any other type. For example, if the user wants to enter information concerning publisher, or publishing city for an article entry from conference proceedings (this information is not currently available), the system maintainer only needs to invoke two ready-made action modules, ADD PUBLISHER and ADD PUBLISHING CITY. Nothing else needs to be changed.

3.5. ARCHITECTURAL DESIGN FOR SUPERBIB

The design of SuperBIB was primarily derived from the structure of the problem to be solved, and was validated by using the Structured Design techniques. The architectural design of SuperBIB is presented in Figure 3-2.

3.5.1. SUPERBIB

SUPERBIB is an executive (a driver) module for the SuperBIB system (Figure 3-2A). SUPERBIB accomplishes its tasks by invoking its subordinates. The immediate subordinates of SUPERBIB are listed below:

- (1) ADD BIB, adds a reference entry into reference file,
- (2) INQ BIB, inquires of the reference entries for given authors' names, words from titles, publication dates, other search words, or a combination of these.
- (3) FMT BIB, formats the bibliography. The formatted bibliography may be routed to a terminal screen, a printer queue, or a file of the user's choice.
- (4) MOD BIB, modifies reference entries in the reference file by invoking a text editor, vi or ed.
- (5) SORT routines, sort reference entries by the senior author's last names and publication dates [3-3].
- (6) HELP, displays on-line help.

[3-3] Once the user sorted the reference file, the sorted reference file name becomes the new reference filename.

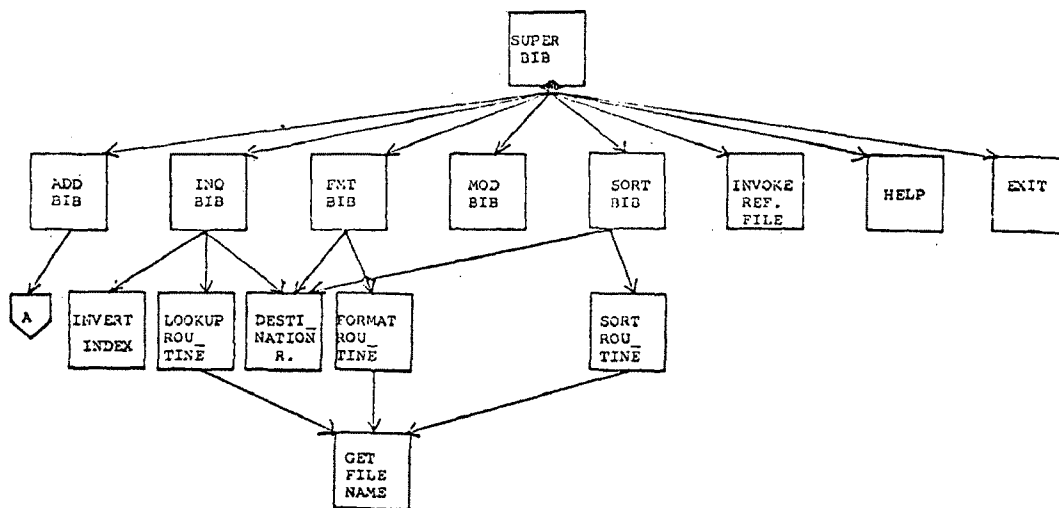


Figure 3-2A Architectural Design of SuperBIB.

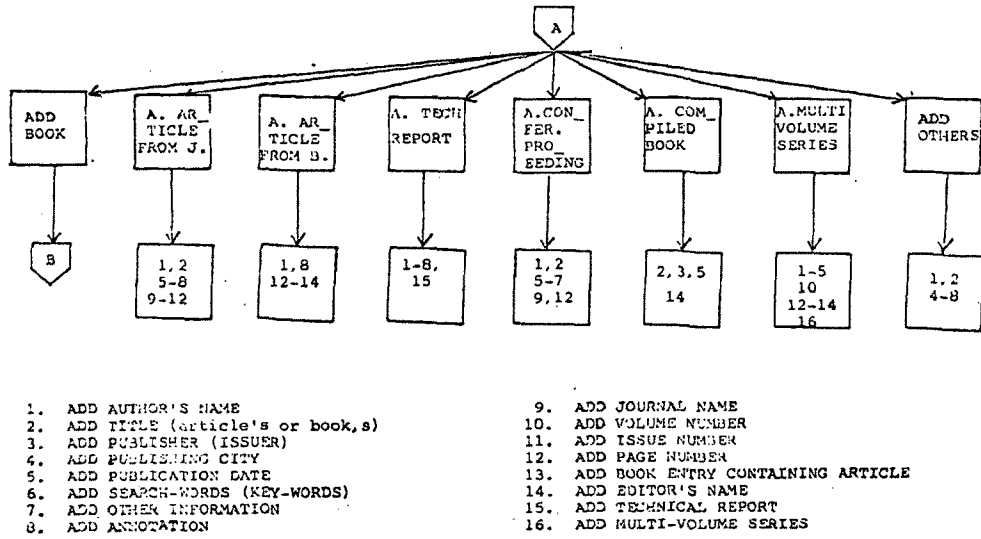


Figure 3-2B Architectural Design of SuperBIB.

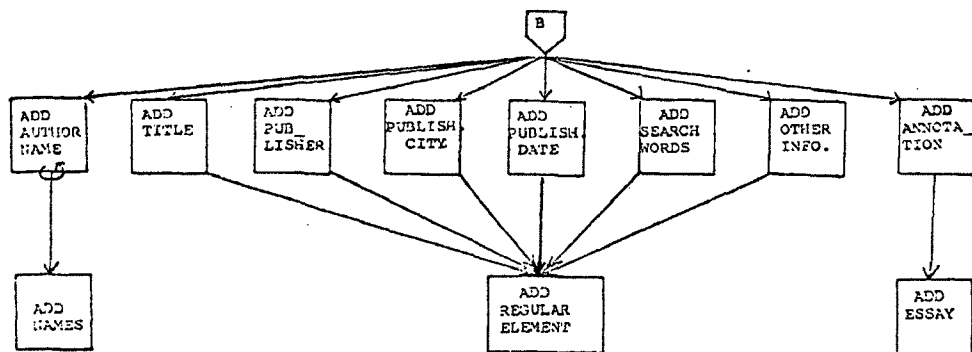


Figure 3-2C Architectural Design of SuperBIB.

3.5.2. ADD BIB

ADD BIB is a transaction center (Figure 3-2B). ADD BIB identifies and routes all types of publications. The immediate subordinates (transaction modules) of ADD BIB are:

- (1) ADD BOOK, adds a book entry into the reference file,
- (2) ADD ARTICLE FROM JOURNAL, adds an article entry from a journal into the file,
- (3) ADD ARTICLE FROM BOOK, adds an article entry from a book into the file,
- (4) ADD TECHNICAL REPORT, adds a technical report entry into the file.
- (5) ADD CONFERENCE PROCEEDINGS, adds an article entry from conference proceedings into the file,
- (6) ADD COMPILED BOOK, adds a compiled book entry into the file,
- (7) ADD MULTI-VOLUME SERIES, adds a multi-volume series entry into the file.
- (8) ADD OTHERS, adds other bibliographic information such as unpublished materials, etc.

3.5.3. ACTION MODULES OF ADDBIB

Based on the principle of Transaction Analysis, the author created one action module for adding each entry element (Figure 3-2C). The SuperBIB action modules are:

- (1) ADD AUTHOR'S NAME, adds authors' names into a reference file.
- (2) ADD TITLE, adds a title of an article (or a book) into the file. (Note that an article entry from a book has two titles, an article title and a book title. Invoke ADD TITLE to add the article title, and ADD BOOK TITLE to add the book title.)
- (3) ADD PUBLICATION DATE, adds a date of publication.
- (4) ADD PUBLISHER, adds the publisher information.
- (5) ADD PUBLISHING CITY, adds the city of the publication.
- (6) ADD JOURNAL NAME, adds the name (title) of a journal.
- (7) ADD VOLUME NUMBER, adds the volume number of a journal.
- (8) ADD ISSUE NUMBER, adds the issue number for a particular volume.
- (9) ADD PAGE NUMBER, adds page numbers of an article.
- (10) ADD BOOK TITLE, adds a title of the book containing an article which the user is interested in. (See ADD TITLE.)
- (11) ADD EDITOR'S NAME, adds editors' names.
- (12) ADD SERIES, adds the name of a multi-volume series.
- (13) ADD SEARCH WORDS, adds other search words (keywords) beyond the authors' names, words from titles, and publication dates.
- (14) ADD OTHER INFORMATION, adds other information such as secondary references, or library call numbers, etc.
- (15) ADD ANNOTATION, adds the abstract or comments of the publication.

3.5.4. DETAILED MODULES

A set of three detailed (bottom-level) modules, ADD NAMES, ADD ANNOTATION, ADD REGULAR ELEMENT, were developed in this project (See Figure 3-2C). ADD NAMES adds authors' (or editors') names into a reference file. In the cases of multiple authors or editors, ADD NAMES calls itself recursively. ADD ANNOTATION adds an annotation (an abstract, comments) to the reference file. ADD REGULAR ELEMENT processes the inputs for all types of reference elements, except names and annotations.

3.5.5. NOTES FROM DESIGNER

The input for an annotation has to be the last one in a reference entry in the current Unix bibliographic system although no written documents have said so. The annotation may be only one paragraph long although the documentation says differently. To obtain more than one paragraph of the annotation input, a blank line may be used to separate the paragraphs of annotation. The author decided not to do so because this would increase the number of reference entries in the file incorrectly.

The current unix bibliographic system has adequate inquiry, printing (formatting) and sorting facilities,

therefore the designer decided to invoke these facilities by using a Unix command, System, instead of developing these features from scratch.

Pressing an ESCAPE key brings the system to the parent menu (or module) of the current one. If the user presses the ESCAPE key while entering the input for reference entry elements, SuperBIB escapes to the menu of SELECT A PUBLICATION TYPE.

For simplicity of design and implementation, the user's input for a reference entry element is written into a reference file directly. No buffer, or temporary storage, is provided. This implies that the user needs to edit the reference file if he has pressed the ESCAPE key while entering the input for the reference entry elements.

The window variables, the reference file, the files storing the search words, and the inquiry result, and formatted bibliography are global variables in this thesis project.

3.6. PSEUDO CODE OF SUPERBIB

3.6.1. SUPERBIB

The pseudo code of the executive module, SUPERBIB, is listed below.

- (1) Set up five windows, The characteristics of these windows are illustrated in Table 3-1.

TABLE 3-1 The Proposed Window Layout

WINDOW No.	SIZE (lines)	PURPOSE
1	1	Display greetings
2	10	Display menus, or questions from SuperBIB
3	1	Separate Windows #2 and #4
4	11	Prompt the user, and read the inputs
5	1	Report status of SuperBIB

- (2) Display a greeting 'SuperBIB - Super Bibliographic System' at window #1.
- (3) At window #5 display the contents of status report, such as 'ADD INQUIRE PRINT EDIT SORT FILE HELP',
- (4) At window #2 display a menu containing the choice of ADD, INQUIRE, PRINT, EDIT, SORT, INVOKE NEW REFERENCE FILE, HELP, and EXIT.
- (5) Allow the user to select one command from the menu on window #2,
- (6) Update the status report by highlighting an appropriate command name,
- (7) Invoke the appropriate lower-level modules to execute the command.
- (8) Exit SuperBIB if the user selected EXIT command. Otherwise, return to the appropriate point of execution in SuperBIB.

3.6.2. ADD BIB

The pseudo code of a representative transaction center module, ADD BIB, is listed below.

- (1) Display a menu containing the information of publication types - BOOK, ARTICLE FROM JOURNAL, ARTICLE FROM BOOK, TECHNICAL REPORT, CONFERENCE PROCEEDINGS, COMPILED BOOK, MULTIPLE-VOLUME SERIES, OTHERS.
- (2) Display a new status report containing the publication type.
- (3) Allow the user to select one of the above publication types,
- (4) Update the status report.
- (5) According to the publication type selected, invoke one of the following modules: ADD BOOK, ADD ARTICLE FROM JOURNAL ADD ARTICLE FROM BOOK, ADD TECHNICAL REPORT, ADD CONFERENCE PROCEEDINGS, ADD COMPILED BOOK, ADD MULTI-VOLUME SERIES, ADD OTHERS.
- (6) Escape to SELECT A COMMAND menu, if the escape switch is on.

3.6.3. ADD BOOK

ADD BOOK adds a regular book entry to the reference file. The pseudo code of ADD BOOK (a representative of transaction module) is listed below.

- (1) Invoke ADD AUTHOR's NAME. Escape to SELECT A PUBLICATION TYPE menu, if escape switch is on.
- (2) Invoke ADD TITLE. Escape to SELECT A PUBLICATION TYPE menu, if escape switch is on.
- (3) Invoke ADD PUBLISHER. Escape to SELECT A PUBLICATION TYPE menu, if escape switch is on.
- (4) Invoke ADD CITY. Escape to SELECT A PUBLICATION TYPE menu, if escape switch is on.
- (5) Invoke ADD PUBLICATION DATE. Escape to SELECT A PUBLICATION TYPE menu, if escape switch is on.
- (6) Invoke ADD KEY-WORDS. Escape to SELECT A PUBLICATION TYPE menu, if escape switch is on.
- (7) Invoke ADD OTHER INFORMATION. Escape to SELECT A PUBLICATION TYPE menu, if escape switch is on.
- (8) Invoke ADD ABSTRACT. Escape to SELECT A PUBLICATION TYPE menu, if escape switch is on.

(Due to their similarities, the algorithms of the rest of the transaction modules are omitted.)

3.6.4. ADD AUTHOR's NAME

The pseudo code of a representative action module, ADD AUTHOR's NAME, is described below.

- (1) Prepare the appropriate strings for prompting the user, i.e., '<<< ENTER AN AUTHOR'S NAME. >>>'.
,
- (2) Invoke ADD NAMES with the appropriate prompt strings,

3.6.5. ADD NAMES

The pseudo code of a representative of detailed modules, ADD NAMES, is described below.

- (1) Prompt the user for input.
- (2) Read the user's response.
- (3) Exit ADD NAMES if the escape switch is on.
- (4) If the user's response is not null, prefix the user's response with a percent sign (%), an appropriate key-letter, and a space, then write the user's response to the Unix-bib reference file. (Refer to Table 3-2 for appropriate key-letters recognized by SuperBIB.)
- (5) Call ADD NAMES recursively in case of multiple authors or editors.

Table 3-2 Key-letters Recognized by SuperBIB

Key_letters	Description
A	Author's name
B	Book title
C	Location (city name, etc.) of the publisher
D	Publication date
E	Editor's name
I	Issuer (publisher)
J	Journal name
K	Search words, key-words
N	Issue number
O	Secondary references, library call numbers, etc.
P	Page numbers
R	Technical report name
S	Series title
T	Title
V	Volume number
X	Annotation

- (1) Invoke UPDATE INDEX FILE to maintain an inverted index file.
- (2) Determine the destination of the inquiry results.
- (3) Name a file to hold inquiry results, if applicable.
- (4) Create a file to hold search words, such as authors' last names, words from titles, publication dates, or other search words, or a combination of these.
- (5) Call Lookbib, to search a reference file for reference entries that match the search words.
- (6) Route the search results to the terminal, a printer queue, or a file. Note that the ability to invoke Bib, or Nroff is not required.

3.6.7. FMT BIB

The pseudo code of FMT BIB is listed below:

- (1) Determine the destination of the formatted bibliography.
- (2) Name a file to hold the formatted bibliography, if applicable.
- (3) Determine the style of the bibliography, such as regular or annotated, single or double spaced, numbered or un-numbered.
- (4) Invoke Roffbib to format a bibliography.
- (5) Route the bibliography to a terminal, to a file, or to a printer queue.

3.6.8. MOD BIB

MOD BIB calls system editors, vi or ed, by using Unix's system command. Note that the ability to invoke other editors, and the ability of modifying by answering a series of questions are not required.

3.6.9. HELP

HELP provides the user with on-line help. A sample of proposed on-line help messages is listed below.

- (1) Reference file name : _____
- (2) Search-word file name : _____
- (3) Inquiry-result file name : _____
- (4) Bibliography file name : _____
- (5) Sorted file name : _____
- (6) To edit, press DELETE, or CTRL-u.
- (7) to advance the cursor, press SPACE BAR
- (8) to select a command, press RETURN key
- (9) To escape to parent levels, press ESC keys.

3.7. PROTOTYPE

The SuperBIB design was further advanced by developing a prototype. Prototyping is building a model of a

complex system before building the final version of it. Prototyping is not only a valuable tool for a software designer, but also a valuable tool for systems analysts and programmers. However, it may lead to problems if not used properly.

As P. Freeman suggested in his Tutorial on Software Design Techniques, prototyping can be used in the following areas [Freeman, 1983].

- (1) Prototyping is to build a subsystem which is substantially like the final system. By placing the subsystem into service, feedback of users is collected, and the subsystem is studied. Based on knowledge of the prototype, a complete system is developed, and the prototype is replaced. Traditionally, the main purpose of prototyping is to collect the feedback of potential users, and to finalize the design of the desired system. Some prototyping primarily is to provide the users with quick service although it may not be full service.
- (2) Prototyping can be used at each stage of development to make decisions. At the analysis stage, an analyst may build a prototype to get feedback from potential users on proposed specifications. At the design stage, a designer may prototype several different designs to determine whether they work, which one works, or which is the most efficient one. During implementation, the programmer may prototype alternative algorithms to study their properties.

The reasons for developing a prototype in this project were to collect user feedback, and to finalize the

design for SuperBIB.

The prototype of ADDBIB ran well in less than a week and valuable feedback was used to finalize the system. The prototype of ADDBIB allowed the user to enter all entry elements but the annotation. It did not have editing abilities, such as character deletion and line deletion. The user had to type the input correctly. Once he pressed a key, there was no way to modify it except invoking a system editor at the Shell (monitor) level. There were no on-line help facilities either.

The prototype was presented to potential users and their comments were collected. The major suggestions were these:

- (1) A more satisfactory way to obtain the reference filename.
- (2) A more reasonable sequence, and a more understandable wording of templates (prompts for inputs).
- (3) A more intelligent status report was added to inform the user of the publication type.

3.8. DISCUSSIONS

3.8.1. DESIGN PRINCIPLE

On the basis of this project, the author agrees that the principles of a good software design are correct and should be followed closely.

The author designed the SuperBIB system using the principles of a good software design. SuperBIB is composed of relatively small and independent modules, and these modules are arranged in a hierarchy of importance, instead of execution sequence. The executive module, SUPERBIB, is a "super" module that theoretically would solve the whole problem by invoking subordinate modules. For example, ADD BIB adds reference entries into a reference file. Most of the common functions were identified and designed as independent modules so that these modules may be re-used. The changing needs of the user may be served better because SuperBIB was designed with ease of expansion and contraction in mind.

3.8.2. STRUCTURED DESIGN TECHNIQUES

The author followed closely the techniques of Structured Design (Transform Analysis, Transaction Analysis, etc.,) but she did not find that these design methods were particularly useful in this thesis project. SuperBIB could

be designed by merely using the principles of good software design, and some educated common sense. In her opinion, the effort spent in designing SuperBIB using rigid Structured Design methods did not pay off very well.

The SuperBIB design did result in a relatively clean structure, the development was relatively trouble-free, and the potential maintenance costs should be minimized. The author agrees that these rigid Structured Design techniques are useful in validating SuperBIB design although the effect of using rigid Structured Design techniques in this project was not significant. Perhaps this was due to the fact that the size of this project was relatively small. Only one software developer was involved. If the size of a software project is too large for one software developer to work on, a team of developers may be assigned to the project. A small team may be composed of two software developers. A large team may have over two thousand software developers. For a project of this size, the complexity of the problem is very great. To partition the large system appropriately, and to wisely assign modules to software development teams, perhaps the Structured Design techniques become a necessity.

3.8.3. MENU TREE AND USER'S GUIDE

The author found that a good way to communicate with the user in a predominantly user-interface project is to develop a menu tree, a hierarchy of menus, and by developing a draft of the user's guide. Due to the fact that developing a menu tree requires less labor and costs less, the author recommends that a menu-tree should be developed first, then a rough draft of a user's guide should be developed. These jobs should have been done by the analyst in the analysis stage, however the designer should cover these jobs if the analyst has not done so. In either case, the document should be verified by the representatives of the user.

In this thesis project, the author did a certain amount of work in menu-tree area shortly before she prototyped the system, but the document was not in any presentable (pleasant) format and she did not verify ("walk through") it with the user. During the prototyping stage, crucial decisions were made by guessing the users' needs (or preferences) and hoping that the users would like these guesses.

3.8.4. PSEUDO CODE

The author regrets that most of the pseudo code for SuperBIB was not done in a presentable (pleasant) form [3-4]. The author documented some modules in pseudo-code before prototyping. But these documents were not in any presentable form, therefore the author did not get too much benefit out of these.

The author believes that it is advantageous to document everything in a presentable form. Preparing a presentable document in itself gives the software developer a second chance to think the whole thing over. The presentable form of documents can be recognized next morning, and beyond. Furthermore, a presentable document equips the software developer with a medium to communicate with other software developers, and the user, if necessary.

The author believes that the prototypes could have been neater if she documented the pseudo code in a presentable form, walked through it, and referred to it during development. While doing prototyping, the author

[3-4] The pseudo-code was not formally documented until the time of writing this chapter.

found herself more than occasionally forgetting to refer to design documentations because she was not well-organized during the course of project development, and she was trying to do too many things at the same time. She thought that she remembered the design well, and the design documents were not all in a pleasant (or presentable) form, therefore sometimes she did not refer to the design documents when she should have.

3.8.5. PROTOTYPING

The author believes that it is usually too expensive to prototype a system, therefore it should be avoided, if possible. A menu-tree and preliminary copy of the user's guide would be a better alternative, and can usually do the job well. For instance, if the user does not like to name a reference file before selecting a command, it is very easy to make the changes in a menu-tree. It will take a little longer to revise the user's guide. Definitely it will take much longer to modify, compile, debug, and test the code.

In this thesis project, prototyping seems to be justified because the author was a novice Unix and C-language programmer and it is the designer's responsibility to ensure that her design works (at least in one way). It

seems appropriate that the author prototyped the system in order to finalize her design.

3.8.6. ANALYSIS PHASE IS MOST IMPORTANT

Based on the experience from developing this thesis project, the author believes that it is absolutely true that the analysis phase is the most important phase in the software life-cycle. If the analyst failed to prescribe precisely the human-machine aspect of the system, it is likely that the designer might not design this part. This would leave the programmer (or coder) to perform the analyst's and designer's duties while coding (or prototyping) the system. And the user's needs are not likely to be served well in this way.

Due to the fact that in this project the analyst, the author, did not analyze the human-machine interaction in precise terms during the analysis phase, therefore the designer, the author, did not formally design the human-machine interaction (at least not in a presentable form). Also she did not walk through her design in the human-machine interaction area with the user. To make the matter worse, the pseudo-code was not done in a presentable form either. The author felt that she was like a coder who was forced to perform the analyst's job, and was

forced to make design decisions. The result is that the shortcomings of the first release of SuperBIB were apparent. To remedy this problem, the author had to iterate the analysis, design (redesign), and implementation of SuperBIB.

CHAPTER 4

IMPLEMENTATION

Implementation (coding and testing) is relatively straightforward once the analysis and design have been done properly.

The principles of good software implementation, the techniques of structured programming, top-down coding and top-down testing, the implementation of Superbib, and the problems encountered are discussed below.

4.1. PRINCIPLES OF GOOD SOFTWARE IMPLEMENTATION

4.1.1. TOP-DOWN IMPLEMENTATION

Top-down implementation is a strategy of coding and testing high-level, executive modules as soon as these modules have been designed, and generally before the low-level, detail modules have been designed, and definitely before these are coded or tested.

The bottom-up approach is a practice where the software developer works on the bottom-level modules first, then the intermediate modules. The upper-level modules are developed last. In a bottom-up development,

the major conditions of the system are usually not known until the integration phase, when there is not enough time to deal with the problems which arise when the modules are brought together. In the top-down approach, the major interfaces of the system are tested at an early stage, and problems can be solved while there is still time to do so. Usually, the top-down approach is better than the bottom-up, but sometimes critical modules should be developed first, even if these modules are lower in the hierarchy of a design tree (Structure Chart). Top-down development has positive effects on reliability, and, indirectly, on the programmer's productivity.

According to Yourdon, the advantages of top-down implementation are these:

- (1) Major interfaces are exercised (tested) at the early implementation phase. The top-down approach exercises the major interfaces at an early stage, hence reducing the chance of discovering major design flaws. In the bottom-up approach, major interfaces usually are not tested until the very end. If design flaws were discovered at the end of development, major recoding is needed while the deadline problem is too sensitive to deal with.
- (2) Users can see a working demonstration of the system at the early stage of development. The software developer can demonstrate a skeleton version of the system to the potential user at an early stage before he has wasted a great deal of time coding from inaccurate analysis or design

documentation.

- (3) Debugging is easier. Debugging is easier in a top-down implementation because the top-down implementation tends to be incremental in nature. A software developer usually adds one new module to an existing skeleton of the system. If the new system misbehaves, the developer knows the problem must be located in the new module, or in the interface between the new module and the rest of the system. The developer may even remove the new module, reinsert the stub (dummy module) and try to debug it. Note that in the top-down implementation, the lower-level modules that have not been coded are substituted by a stub (dummy module), which is relatively inexpensive compared to a test driver of a bottom-up implementation.
- (4) Top-down implementation eliminates the need for a test driver, since the existing skeleton system can serve as a test driver. At bottom-up implementation, a test driver, which is more expensive to code, is required.

4.1.2. STRUCTURED PROGRAMMING

Structured Programming was first mentioned in "Notes on Structured Programming," by E.W. Dijkstra. The lack of a precise definition of Structured Programming has confused many software developers. General characterizations of Structured Programming are these:

- (1) Structured Programming includes top-down development.
- (2) Structured Programming means to program using disciplined structures, such as sequence, conditional statements (IF THEN ELSE, CASE, SWITCH), and repetition statements (DO WHILE, DO UNTIL).

- (3) Structured Programming is programming with the controlled use of GOTO statements, but it should be characterized by the presence of structure, not by the absence of GOTOs.

Structured Programming does not mean a software developer absolutely cannot use GOTOs, it means to implement software in a simple, structured and disciplined way, so that the software can be produced faster, cheaper, and the product will be more reliable, and maintainable.

4.2. IMPLEMENTATION ENVIRONMENT FOR SUPERBIB

4.2.1. UNIX PROGRAMMING ENVIRONMENT

According to M. Waite, in 1983, around the United States there were more than 8000 Unix installations, supporting over 200,000 users [Waite]. Why is Unix so popular? Why do so many software developers prefer to do their work in Unix?

The reasons are these:

- (1) Unix is adaptable. Unlike other operating systems (which are written in Assembly Language) Unix is written in a relative high-level language, the C language. It is not easy to transfer an operating system, (or programs written for it), to other types of computers if the operating system is written in Assembly language, due to the fact that the assembly languages of different computers are significantly different.

However, it is easy to transfer Unix, its utility programs, or its application programs, to other types of computers because most of Unix software is written in a relatively high level language, the C language. As long as the host-computer-to-be has a C compiler, the transfer could be done relatively easily.

- (2) The Unix Shell is a programmable command interpreter. It accepts the commands typed on the user's terminal, and invokes appropriate programs to execute the command. Most operating systems have command interpreters which are not programmable. The programmability of the Unix Shell provides a great power to the software developer.
- (3) Unix has a library of software tools. These tools are relative simple; they do one thing and do it well. These tools can be connected (piped) together to do complex tasks. Due to the fact that the Unix Shell is programmable and the ability to pipe the output of one software tool to be the input of another tool, software developers often avoid programming from scratch.
- (4) Unix provides many major programming languages, such as FORTRAN, COBOL, Pascal, C and LISP. Some other languages, such as APL, LOGO, RPG, and RP-1 are available on specific Unix systems. Programming debugging tools are also available.
- (5) Unix provides Programmer's Workbench for large software development. Programmer's Workbench is a package of Unix tools which simplifies the program transfer between computers. It also records all the changes made to the programs so that any earlier version of a program may be recreated at any given time. It also allows programs to be tested in part as they are developed.

4.2.2. SOFTWARE PROJECT MANAGEMENT SYSTEM

The Unix Software Project Management System (SPMS) techniques were used to manage the Superbib development

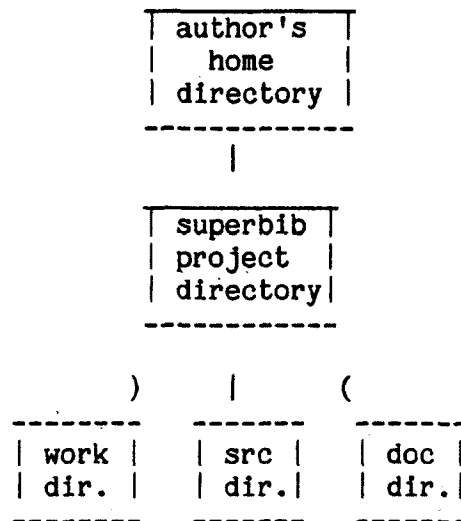
[Nicklin, 1983]. SPMS is a system for the management of medium to large-scale software systems. SPMS provides a number of commands which simplify tasks associated with program development and maintenance in the Unix environment. Changing directories in a regular Unix directory tree requires the user to remember the precise position of the directories in the tree. To move to a brother-or-sister directory, first it is required to move up to the parent level then move down to the directories desired. In the SPMS directory tree, changes of directories are straightforward. It is not required to move up to the parent directory in order to move to brother-or-sister directories. Therefore, the precise location of the directories in the directory tree does not need to be specified or memorized.

An SPMS project directory was built during Superbib's development. The directory tree is illustrated in Table 4-1.

Note that SRC DIR is a directory for storing all source-code files, DOC DIR is a directory for storing all documents for the SuperBIB project, and WORK DIR is a working directory for the author to develop the SuperBIB software. Once a source file, or document had been com-

Table 4-1 SPMS Project Directory Tree for SuperBIB

=====



=====

pleted, it was copied into appropriate directories for security reasons.

4.2.3. CHOICE OF PROGRAMMING LANGUAGE

Superbib was coded in the C language. The primary reason was the author's intention to use a package of library functions, Screen Updating and Cursor Movement (Window Package) [Arnold]. Window Package, written in the C language, may be used to move the cursor from any place to another on a terminal screen. It may be used to get

inputs from the terminal in a screen-oriented fashion. This package allows the C programmer to do the most common type of screen-oriented and window functions with ease.

The C language is a programming language developed by D. Ritchie at Bell Labs around 1972. It is an offspring of Algol-60. The C language is a relatively low-level language that allows the programmer to access machine and operating system features. Thus, a program written in the C language may achieve maximum computing efficiency. It is also a relatively high-level language that can hide many of the details of the computer architecture. Thus, a program written in the C language promotes the programming (or the programmer's) productivity. Programs in the C language run fast and take little storage space. The ability to build a complex program out of simple elements is a great strength of the C language.

The C language is the most basic tool of Unix. Most Unix software was coded in the C language. Nearly all software tools supplied with Unix were written in C. The Unix environment supports various programming languages, such as Pascal, APL, FORTRAN, etc. However, a program written in the C language interfaces better with the Unix programming environment.

The disadvantages of the C language is that the programs written in C are too compact to understand at first glance. Perhaps a better choice of the programming language for this project would have been Modula-2, due to the fact that the programs written in Modula-2 can have better modularity.

4.3. SUPERBIB IMPLEMENTATION

Superbib implementation was done by using Top-down coding and testing, and Structured Programming techniques. An executive module, Main, could add reference entries, inquire reference files, produce a bibliography, modify reference entries, and sort reference entries. SUPERBIB does these by invoking a number of lower-level modules, such as ADD BIB, INQUIRE BIB, FORMAT BIB, MOD BIB, and SORT BIB. These modules accomplish their tasks by invoking lower-level modules, such as ADD JOURNAL ARTICLE, ADD TECHNICAL REPORT, etc. These modules accomplish their tasks by invoking lower-level modules, such as ADD AUTHOR's NAME, ADD TITLE, etc. These modules accomplish their tasks by invoking the three bottom-level modules, ADD NAMES, ADD ANNOTATION, and ADD REGULAR ELEMENT. ADD NAMES adds the names of the authors (or editors). ADD ANNOTATION adds the inputs of an abstract, or readers'

comments. ADD REGULAR ELEMENT adds the rest of the elements.

These modules are arranged hierarchically in the order of importance. The executive module was coded and tested first. A stub (dummy module) was used whenever a higher-level module calls a lower-level module which has not been implemented. The modules were coded in a disciplined way--using sequences, conditional and repetition statements. A few GOTOs were used in a controlled way. For example, GOTOs were used to exit the help facility and return to the command menu, or to escape to a parent module if the user wants to exit a current module prematurely.

4.3.1. SUPERBIB SOURCE FILES AND MODULES

The SuperBIB source-code files and its modules are presented in Table 4-2.

4.3.2. HEADER FILE

A header file, bib.h, contains the declaration of global variables, definitions of symbolic constants, etc. bib.h was included in the executive module, main during the compilation.

Table 4-2 The SuperBIB Source-code
Files and Modules.

File	Modules Defined in The File
bib.h	--
superbib.c	main
getfile.c	get_file
read_string.c	getstring
addbib.c	add_ref add_conference_proceedings add_article_from_book add_multi_volume_series add_edited_book add_article_from_journal add_technical_report add_any_type add_author add_city add_keywords add_others add_abstract add_page add_editor add_art_in_book_title add_series_name add_volume add_journal_name add_number add_report add_regular_element add_names add_essay
askbib.c	search_ref lookup_routine
fmtbib.c	print_bib roffbib_routine
modbib.c	modify_ref
sortbib.c	sort_ref sort_routine
help_msg.c	print_help
exitbib.c	die
invertf.c	invert_file
destination.c	destination_routine

4.3.3. NAME A REFERENCE FILE

A module, Get file in Getfile.c, is to name a reference file [4-1]. Primarily get file does this task by invoking getstring in read string.c.

4.3.4. SUPERBIB'S EXECUTIVE MODULE

An executive module, main in superbib.c, is developed to drive the whole SuperBIB system.

The algorithm of the executive module is listed below.

- (1) Include all SuperBIB's source-code files listed in Table 4-2.
- (2) Set up four windows (windows #1,2,4,5). Note that window #3 was deleted to make more room on the screen for window #2.
- (3) Display Superbib greeting 'SuperBIB - Super Bibliographic System' at window #1,
- (4) Display the contents of status report, such as 'ADD INQUIRE PRINT EDIT SORT INVOKE NEW REFERENCE FILE HELP' at window #5
- (5) Invoke get file, to name a reference file, if it is the first cycle of SuperBIB execution,

[4-1] Names appended with '.c' indicate source-code files written in the C-language.

- (6) Display a menu, SELECT A COMMAND, for a choice of commands - ADD, INQUIRE, PRINT, EDIT, SORT, INVOKE NEW REFERENCE FILE, HELP, and EXIT.
- (7) Allow the user to move the cursor up and down the menus by pressing the space bar; to select a command by pressing the RETURN key,
- (8) Update the status report once the user has made a selection,
- (9) According to the user's selection, invoke one of the following modules in addbib.c:

add ref,
to add entry elements into the file,
search ref,
to search entries in the file,
print bib,
to print the bibliography,
modify ref,
to edit the file,
sort ref,
to sort entries in the file,
get file,
to get a reference filename,
print help,
to get on-line help.

- (10) Invoke die, to exit SuperBIB if the user selected EXIT. Return to SELECT A COMMAND menu, if the user selected ADD, HELP, or INVOKE A REFERENCE FILE. Otherwise return to WIN (window) SETUP to re-setup the windows. (Note that commands of EDIT, INQUIRE, and PRINT would destroy the window setup, therefore it should be reset.)

4.3.5. ADD REFERENCE ENTRIES

The module add ref in addbib.c, adds citations (author names, titles, publication dates, etc.), annotations (abstracts, comments), and other information to a

reference file. The algorithm of add ref is listed below.

- (1) Display a menu containing all types of publication entry - BOOK, ARTICLE FROM JOURNAL, ARTICLE FROM BOOK, TECHNICAL REPORT, CONFERENCE PROCEEDINGS, COMPILED BOOK, MULTI-VOLUME SERIES, and OTHERS.
- (2) Allow the user to press the space bar to advance the cursor, to press the RETURN key to select a publication type, or to press the ESCAPE key for an escape to the parent menu, SELECT A COMMAND.
- (3) Update the status report by highlighting ADD and the publication type selected. The publication types are abbreviated as follows: BOOK, ARTICLE/J (article from a journal), ARTICLE/B (article from a book), REPORT (technical report), COMPILED (compiled book), SERIES (multiple volume series), OTHERS (unpublished).
- (4) According to the publication type selected, invoke one of the following modules in addbib.c:

add regular book,
to add a book entry,
add journal,
to add an article entry from a journal,
add article in book,
to add an article entry from a book,
add technical report,
to add a technical report entry,
add conference proceedings,
to add a conference proceedings entry,
add edited book,
to add a compiled book entry,
add multi volume series,
to add multi volume series entry,
add any type,
to add unpublished information.

- (5) Escape to SELECT A COMMAND menu if the escape switch is on. (Note that the switch is on, if the user wants to exit certain modules prematurely.)

4.3.6. ADD REGULAR BOOK ENTRIES

A module, add regular book, in addbib.c, adds a regular book entry to a reference file. The algorithm of add regular book is listed below.

- (1) Invoke add author to add an author's name to the file. Escape to SELECT A PUBLICATION TYPE menu, if the escape flag is on.
- (2) Invoke add title to add a title entry to the file. Escape to SELECT A PUBLICATION TYPE menu, if the escape flag is on.
- (3) Invoke add publisher, to add a publisher entry to the file. Escape to SELECT A PUBLICATION TYPE menu, if the escape flag is on.
- (4) Invoke add city, to add a publishing city to the file. Escape to SELECT A PUBLICATION TYPE menu, if the escape flag is on.
- (5) Invoke add publication date to add a date of publication to the file. Escape to SELECT A PUBLICATION TYPE menu, if the escape flag is on.
- (6) Invoke add keywords to add additional search words to the file. Escape to SELECT A PUBLICATION TYPE menu, if the escape flag is on.
- (7) Invoke add others to add reader's comments, secondary references, library catalogue numbers, or other information into the file. Escape to SELECT A PUBLICATION TYPE menu, if the escape flag is on.
- (8) Invoke add abstract to add an abstract entry to the file. Escape to SELECT A PUBLICATION TYPE menu, if the escape flag is on.
- (9) Write a blank line to the reference file to separate each publication entry.

Note that due to the strong similarity, the algorithms of the following modules are omitted:

add journal,
add article in book,
add technical report,
add conference proceedings,
add edited book,
add multi volume series,
add any type.

4.3.7. ADD AUTHOR NAMES

The module add author, in addbib.c is to add authors' names to the file. The algorithm of add author is listed below.

(1) Initialize the following strings for prompting:

```
prompt1 : <<< ENTER AN AUTHOR'S NAME. >>>
prompt2 : AUTHOR
prompt3 : <<< MORE AUTHOR'S NAMES? >>>
```

(2) Invoke add names with the above prompt strings and a key-letter, A, as actual parameters. For example, 'add_names(prompt1, prompt2, 'A', prompt3)' is a legal function call.

4.3.7.1. ADD NAMES

A module, add names, in addbib.c, adds the authors' (or editors') names into the reference file. The algorithm of add names is listed below.

- (1) Signal the user by prompting '<<< ENTER AN AUTHOR'S NAME. >>>', `prompt1`.
- (2) Display instructions, such as 'Enter given name first.', and 'To bypass, press RETURN.', etc.
- (3) Display the prompt (`prompt2`) for inputs at the next available line position (`line_pos`) of window #4. (Note that up to ten lines of the bibliographic information can be displayed at window #4.)
- (4) Read the input in by invoking `getstring`, of `read string.c`.
- (5) If the escape flag is on, exit this module. Otherwise do the following steps:
- (6) Skip leading spaces.
- (7) If the input is not null, prefix the input string with a percent sign (%), a key-letter (A or E) and a space. Write these strings to the reference file.
- (8) Get ready for next element inputs by writing a new-line symbol to the file.
- (9) Call `ADD NAMES` recursively, if there are multiple authors' names.

Note that due to their similarity, the algorithms of `add essay`, and `add regular element` are omitted.

4.3.8. SEARCH REFERENCES

A module, `search ref`, in `askbib.c`, allows the user to inquire the reference file, The user may search for: authors' or co-authors' last names, words from the title, other search words (key-words), or a combination of the above.

The algorithm of search ref is listed below.

- (1) Display instructions about inquiry.
- (2) Invoke invert file, to create, (or update) an inverted index of the reference file. This can be done by using ' system ("invert reference_filename"); '.
- (3) Determine the destination of the inquiry result. The possible destinations are the terminal screen, printer, or file.
- (4) Create a file to hold search words.
- (5) Determine the name of the file to store the inquiry result, if the user has indicated that the destination is a file.
- (6) Invoke Lookbib to search for the search words (key-words) by using ' system ("lookup reference_file_name"); '.
- (7) Route the inquiry result to the destination specified.

4.3.9. PRINT BIBLIOGRAPHY

A module, print bib in fmtbib.c, prints the stand-alone bibliography. The bibliography can be routed to a terminal screen, a printer, or a specified file. This module does the following:

- (1) Display instructions about printing the stand-alone bibliography.
- (2) Determine the destinations of the stand-alone bibliography. The possible destination are the ter-

minal, the printer, or a file.

- (3) Determine the name of the file to store the bibliography, if the destination is to be a file.
- (4) Determine the style of the bibliography, such as single or double spaced, etc.
- (5) By using ' system ("roffbib reference_filename") ', invoke Roffbib to produce the bibliography.
- (6) Route the bibliography to the destination specified.

4.3.10. MODIFY REFERENCE FILE

A module, modify ref, in modbib.c, allows the user to modify the reference file by invoking an editor. Modification of the reference file by answering questions is not implemented. The algorithm of modify ref is listed below.

- (1) Determine the method of modifying - by invoking editors, or questions & answers.
- (2) If the user chose to invoke an editor, determine which editor the user wants - vi, ed. Otherwise print appropriate messages.
- (3) Invoke the editor by using the Unix System command.

4.3.11. SORT REFERENCES

The algorithms of sort ref in sortbib.c is listed below.

- (1) Display instructions about sorting the reference entries,
- (2) Determine the destinations of the sorted reference entries. The possible destinations are the terminal screen, the printer queue, or a file of the user's choice.
- (3) Determine the name of the file to store the sorted reference entries, if the destination is to be a file.
- (4) By using `'system ("sortbib reference_filename");'`, invoke Sortbib to sort the references.

4.3.12. EXIT SUPERBIB SYSTEM

A module, die, in exitbib.c, terminates Superbib properly by calling endwin from Screen Package.

4.3.13. INDEX FILE OF THE BIBLIOGRAPHIC DATABASE

A module, invert file, in invertf.c, creates (or updates) an inverted index for the reference file by using `'system ("invert reference_filename");'`.

4.3.14. ON-LINE HELP

A module, print help, in help msg.c, displays on-line help messages.

4.4. PROBLEMS ENCOUNTERED

The implementation for ADD BIB (addbib.c) went smoothly, due to the fact that this function is completely decomposed according to the structure of the problem. Very few bugs were encountered in the development of ADD BIB. The only problem encountered in the ADD BIB development was the lack of editing capabilities, i.e., character deletion and line deletion in Window Package. The author adapted a small module, GET STRING, from "Comprehensive Data Collection System" by J. Scott Mulligan and Murali Veeramoney to read a string from the terminal and do the deletions, if necessary.

The author wanted to invoke the editor in one of the windows (window #2), and leave the rest of the windows intact. The idea was to provide the user with on-line help, status report, and greeting, while he is using the system editor. However, the idea did not go through because of the complexity of implementing this feature. In order to do so, it was required to open a process with the editing command as an argument by using Popen, read and write the contents of the information that the editing session encountered by using fgets and wprintw, and then close the process by using pclose.

A syntax error in the declaration of pointer variables, such as using ' char *var = "XYZ"; ', while the correct format is ' char *var; ', and attempts to use the above variable, var, as an array, were detected by neither the C compiler nor lint, a program that reports potential errors of the user's application programs. Despite the above misdeclaration and misuse, the program would execute properly most of the time. But the programs would quit working correctly at certain points of execution. The author suspects that the C compiler did not consider the variable, var, as a pointer since it had not been declared correctly. The author also suspects that although she had not allocated space for the variable, var, while using it as an array, the compiler found space for the variable, and the program would execute fine. But the space ran out at a certain point of execution, and the programs quit working properly.

The author suspected that the misdeclaration and misuse caused troubles in the implementation of INQUIRY BIB, the search and select routine of SuperBIB. The author could not redirect the inquiry results to a file of the user's choice, while she had successfully redirected the output of any other Unix command to files.

Some mysterious inconsistencies of the performance of Window Package, such as irregular cursor movements, or characters left over from previous screen images, after clearing up, were suspected to be related to the misdeclaration and misuse.

4.5. CONCLUSIONS

Based on the implementation of this thesis project the author feels that the principles of good implementation are correct. Using the top-down coding, top-down testing and structured programming techniques did result in relatively reliable and maintainable software. The software developed by practicing these good implementation principles does result in a relatively short development time, hence the life-cycle costs are controlled.

The author agrees that software should be developed as tools. These software tools are relatively simple. A tool does only one thing, but it does it well. A large software system should be developed by using appropriate software tools. Programming from scratch should be avoided unless no tools are available. In this sense, the author agrees that Unix is a very good programming environment because of the large collection of software tools in Unix.

The author agrees that SPMS helped her to manage the development of the SuperBIB project. SPMS served as a project librarian for her. Once she had completed something she checked the product into an appropriate SPMS project directory. The materials in that directory can be checked if desired, but the check-in is controlled carefully to avoid potential hardship, such as overwriting a file accidentally. The author did not utilize the full power of SPMS, due to her limited understanding of this subject. The management of this thesis project would have been even easier, if she had known SPMS better.

CHAPTER 5

CONCLUSION

The SuperBIB project achieved its goal of making a bibliographic system (Unix-bib) more useful by adapting human-engineering techniques that are commonly applied to microcomputer software to a mainframe environment.

This chapter discusses the adaptation of human-engineering techniques to the mainframe environment, benefits and problems of the adaptation, the software engineering techniques adopted, and directions future work on this project might take.

5.1. HUMAN-ENGINEERING

5.1.1. WHAT ARE HUMAN-ENGINEERING TECHNIQUES?

Human-engineering techniques are the techniques that harmonize the communication between human being and computer. Usually, these refer to the menus, windows, reverse videos, etc. The menu provides all of the options the user has at one point of the execution, therefore it is easier for the user to utilize the computer. Windows provide a dialogue between the computer and the person who

uses it, therefore the computer appears to be more friendly. The reverse video highlights certain things, and brings the user's attention to important points, therefore it is desirable.

5.1.2. WHY NOT IN MAINFRAME?

These human-engineering techniques have generally not been applied to mainframe software because:

- (1) historically, mainframe software was batch-oriented,
- (2) historically, mainframe software was based on the old-fashioned, paper terminals. The older paper-based terminals lack the direct cursor addressing capabilities, etc., therefore screen-oriented features cannot be implemented.
- (3) the mainframe user is often expected to be more sophisticated.

The extra overhead encountered by executing human-engineered software may be a problem in a busy, or overloaded mainframe environment. A wide variety of terminals (or personal computers) that mainframe software may encounter is another problem, because different types of terminals often have different settings (escape sequences). The software developer may have to write a driver program for each type of terminal (or personal com-

puter) that may communicate with the screen-oriented mainframe software. The microcomputer (personal computer) software does not have this problem because it only deals with one type of terminal.

5.1.3. THEORY AND HYPOTHESIS

The author's theory and hypothesis are correct. She theorized that people still use manual methods to prepare and maintain their bibliographic information primarily because the current automated systems are not easy to use. Hypothetically, by adapting human-engineering techniques, programs can be made easier to use, hence more widely used.

5.1.4. TRANSITION TO MAINFRAME

The author successfully adapted the human-engineering software techniques to a mainframe environment, the VAX 750/780/785 running under Berkeley Unix 4.2. The human-engineered version of Unix-bib is named SuperBIB. The problems encountered during the transition are discussed below.

5.1.4.1. A VARIETY OF TERMINALS

Several common types of terminal, such as H19, VT100, and Z29, are compatible with SuperBIB. One very rare type of terminal, ADDS-Regent25, is partially compatible. Vt52 terminals, now a rather rare terminal on this campus, is not compatible. The rest of the terminal types are untested or unconfirmed.

The author did not experience problems with H19 or Z29 terminals during the project development. She had slight problems with the VT100 terminals. It seems that the problem disappeared when she eliminated an illegal piece of syntax which had escaped the detection of both the C compiler and lint, a program to report potential errors in application programs. These program development tools are apparently flawed; the VT100 terminal, so far as she can tell, works perfectly with SuperBIB.

SuperBIB should function appropriately if the user has his '.login' file set up correctly, and uses common UM terminals, such as the H19, VT100, or Z29. Setting up the terminal automatically is accomplished by including "set term= '/etc/ttype' " in the user's ".login" file. However, this only sets up a few common types of terminals, such as H19, Z29, and VT100, on the University of Montana

campus.

A few users may have to specify the type of their terminals by typing 'setenv TERM XYZ' before invoking SuperBIB, if they use a rare type of terminal, or their '.login' files do not set the terminal type automatically. For instance, to set up Advanced Digital Data System's (ADDS) Regent25 terminal, a rare type of terminal at UM, the user may type 'setenv TERM regent25'.

For other types of terminals, the user must determine the name that Unix is using for his type of terminal by using 'fgrep XYZ /etc/termcap' command, where XYZ is a guessed terminal name that might be used in the Unix terminal capability database, Termcap. For instance, type 'fgrep reg /etc/termcap' to determine Termcap's name for Regent terminals. To set up a TRS-80 personal computer to appropriate capabilities, type 'fgrep trs /etc/termcap' on the screen to confirm the Termcap name that Unix is using for referring to TRS-80 personal computers. If the name, trs, is confirmed, then type 'setenv TERM trs' to set up the correct mode for a TRS-80 PC.

5.1.4.2. REVERSE VIDEO

SuperBIB uses reverse video to indicate the current command in use. Certain older types of CRT terminals, such as ADDS Regent25 terminals, do not have reverse video capabilities, therefore this feature does not work at those terminals.

5.1.4.3. 300-BAUD LINKS

The execution speed of SuperBIB was quite slow on 300-Baud terminals. The slowdown was especially obvious while SuperBIB drew boxes on the screen. Furthermore, Arnold's Window Package does not have the capability to change only the contents inside boxes, while leaving the boxes intact. To speed the system up while operating SuperBIB from 300-Baud terminals, the author decided not to draw boxes at all.

5.1.4.4. LACK OF EDITING POWER

The author adopted Arnold's Screen Updating and Cursor Movement Package (Window Package) to do the common type of window and screen-oriented operations. The Window Package successfully interfaced with the Unix environment and the bibliographic software tools, such as Roffbib, etc. However, the Window Package lacks certain built-in

editing capabilities. To delete a character, Window Package must execute Delch (); to delete a line, it must execute Deleteln (). As the Window Package stands now, the user cannot edit while entering the input from a terminal. The author had to adapt a small module to do the editing while taking input from a terminal. The editing capabilities must be restored again while invoking Lookbib, the bibliographic search routine, where the user supplies the input from terminals.

The author suspects that while invoking the ed editor, the normal ed editing capabilities are lost. The user is instructed to press 'CONTROL-c' and a period (if necessary) for editing. The editing capabilities for the vi editor remain intact.

5.1.5. MORE USEFUL?

SuperBIB is more useful compared with the bare version of the Unix bibliographic system, Unix-bib. SuperBIB has been accepted by both naive and more sophisticated users. Three users, one of them having no experience with computers at all, were invited to make SuperBIB acceptance tests. After fifteen minutes briefing, they were able to use SuperBIB well, experiencing only minor confusions. They all expressed interest in using SuperBIB to prepare

bibliographic information in the future.

The author has casually introduced SuperBIB to a number of graduate students in the Computer Science Department. Their reactions to SuperBIB are very positive.

SuperBIB is likely to become one of the user contributed software tools in the '/usr/mnt/um/student/bin' area on VAX-11/750 computer at the Computer Science Department at the University of Montana.

5.2. VALUE OF SOFTWARE ENGINEERING METHODS

Based on the experience obtained from doing this project, the author agrees that a quality software implementation depends on a good software design, and a good design depends on an adequate analysis, although the converse of these propositions may not be true. A good analysis may not be transformed into a good design because the designer may not follow the principles and methods of good design. A good design may not be transformed into a quality software product because the programmer (or coder) may not follow the principles and methods of good software implementation.

5.2.1. ANALYSIS

De Marco's Structured Analysis methods are good at analyzing the functional aspects of a system, but fall short at specifying precisely the human-engineering (or user-interface) aspects of the system.

The Structured Analysis methods were used to analyze SuperBIB functionally. The Structured Analysis methods and supporting documents did help the author to understand the problem at hand, and to identify what had to be done. First, the Physical and Logical Data Flow Diagrams (DFDs) of the current Unix bibliographic system, Unix-bib, were developed to reflect the current operations in Unix-bib. Then a set of Logical Data Flow Diagrams of the newly proposed bibliographic system, SuperBIB, were derived to identify functional requirements in SuperBIB.

De Marco suggests that, during a complete life-cycle of the analysis stage, we analyze systems without overloading the analysts with the control information, such as the executive components, decision-making and looping. The author agrees that it is a good practice to bypass the control information during the early analysis phase because the amount of information the analyst has to deal with is too great to grasp in the first attempt. The

analyst has to start from somewhere and make progress from there. But bypassing control information during a complete life-cycle of an analysis stage was proved to be incompatible with a predominant user-interface software project. In a predominant user-interface project, the most important thing is to understand what the user wants in the aspect of human-machine interaction, not the functionality of the system. To analyze the human-machine interaction, to harmonize the machine with the human being who will use the machine, control information is a necessity. Therefore, De Marco's Structured Analysis methods are inadequate to analyze a predominant user-interface software project.

A potential solution would be to develop a menu-tree, a hierarchy of menus, and a draft of a user's guide. Based on this thesis project, the author found that a menu tree, and a rough draft of the user's guide were very helpful in analyzing precisely the user's needs in the area of human-machine interactions. The author believes that the analyst should start with deriving a menu tree, then proceed to develop a draft of the user's guide, and verify (walk through) both documents with the users.

5.2.2. DESIGN

The author agrees that the Structured Design method used in the SuperBIB project was adequate to design a system architecturally, although the effect was not significant in this project due to the small size of the project. The author believes that the architectural design has to be supplemented by developing pseudo-code in a presentable form, and the documents should be "walked through" by software developers, including the designer himself.

Developing a prototype is expensive, although it is beneficial. The author feels that prototyping is a good method to collect user feedback, and to validate the design, etc. However prototyping should be used in a disciplined way, otherwise it may become a not-well-planned coding session in the name of prototyping.

The author agrees that user-oriented design, a design method that is based on the observation and analysis of the user's reaction to a proposed software design, is the best way to design an easy-to-use user interface for a given functionality [Good,1984]. Iterating the design based on the user's reaction to the initial design has been proven to be a nice way to design a user interface.

5.2.3. IMPLEMENTATION

The top-down coding and testing proved to be well-suited in software development. Based on this thesis project, the author believes that the top-down approach is better than the bottom-up in the aspect of collecting the user feedback which leads to successful software development. She also agrees that the Structured Programming techniques are a necessity to produce reliable and maintainable software, and to control the life-cycle costs of the software.

Due to the fact that the author had practiced the top-down approach, she was able to demonstrate the skeleton versions of SuperBIB to the representatives of the users at an early stage of development. With the help of project director, Dr. A. Wright, and the fact that the user feedback indicated room for improvement in the first release of SuperBIB, the author realized that her work in the analysis stage was not precise enough in the area of human-machine interaction.

An analysis of the human-machine interaction area was conducted, and verified by the representatives of the potential users. A re-design of SuperBIB, mainly in the areas of INQUIRY, PRINT, and SORT, was done. The

implementation of the re-design was completed in about ten working days, due to the fact that the author had practiced structured programming techniques which make the programs easy to understand, modify, and maintain. The author estimated that over fourteen hundred out of twenty-six hundred lines of original source-code were either added, deleted or modified during the modification [5-2].

In summary, if the author had not practiced the top-down approach and collected the user feedback at an early stage of development, she might not have realized that her work in the analysis stage was not precise enough in the area of human-machine interaction when there was still time to fix the flaw. If the author had not practiced the Structured Programming techniques, she might have had difficulty modifying a relatively large portion of source-code quickly.

5.3. FUTURE WORK

[5-2] The second release of SuperBIB contains little over 3,300 lines of source-code in the C language. These figures include generous comment lines.

5.3.1. MINOR ENHANCEMENTS

Editing without using a text editor (editing by answering questions) would be necessary to serve those users who do not wish to learn any text editor. Entering search words without naming a file, and searching for reference entries that match a part of the search words is desirable. Currently, SuperBIB is aimed at serving the novice users. The development of a sophisticated mode of operation for SuperBIB would be a plus to its overall performance.

5.3.2. MAJOR ENHANCEMENTS

Preprocessors for Nroff and Bib, interfaces between SuperBIB and other systems, such as BIBLIO, LIBHCOR, and the Washington Library Network (WLN) would be major enhancements to SuperBIB.

Nroff, a text processor, is a prerequisite to Bib, a sophisticated bibliographic preprocessor for Nroff in the Unix environment [5-3]. Therefore, an Nroff preprocessor

[5-3] To serve those users who do not wish to learn the Nroff text processor, SuperBIB was built on simplified bibliographic tools, such as Roffbib, Lookbib, Sortbib, and Invert. SuperBIB has nothing to do with Bib, a sophisticated bibliographic preprocessor for Nroff.

would be a prerequisite to a sophisticated human-engineered (user-friendly) bibliographic system in the Unix environment. By using imprecise (incomplete) citation strings in a text file, Bib can retrieve appropriate entries from the reference file, to produce the stand-alone bibliography, and to produce the precise (traditional) citation strings for the text file. Bib recognizes some standard abbreviations, such as the months, certain journal titles, conference names, and certain publishers. For instance, the user may type 'WILEY' for 'John Wiley & Sons', or 'CACM' for 'Communications of the ACM', or 'JAN' for 'January', etc.

Once preprocessors for Nroff and Bib are available, the user could:

- (1) format the stand-alone bibliography in various formats (to be explained),
- (2) retrieve entries from the reference file by using imprecise (incomplete) citation strings,
- (3) abbreviate authors' (or editors') first names,
- (4) reverse the first names and last names for all, or a part, of authors' (or editors') names,
- (5) footnote references,
- (6) hyphenate adjacent references,
- (7) sort by other elements beyond authors' last names, and publication dates.

According to Budd, a summary of various formats of the bibliography generated by using Bib and Nroff are these:

- (1) astrophysical journal style,
- (2) footnoted citations,
- (3) Hanson Normal Forms,
- (4) listref default,
- (5) open alphabetic,
- (6) open numeric,
- (7) SP and E Journal style,
- (8) standard alphabetic,
- (9) standard numeric,
- (10) standard sorted numeric,
- (11) superscripted numeric style.

5.3.3. INTERFACES WITH BIBLIO, LIBHCOR AND WLN

Interfaces between SuperBIB and BIBLIO, (or LIBHCOR), would be beneficial to certain users who are currently using these systems. Links between SuperBIB and the Washington Library Network (WLN) would be beneficial to transfer information between a personal reference file in Unix and a WNL database system through the IBM-PC.

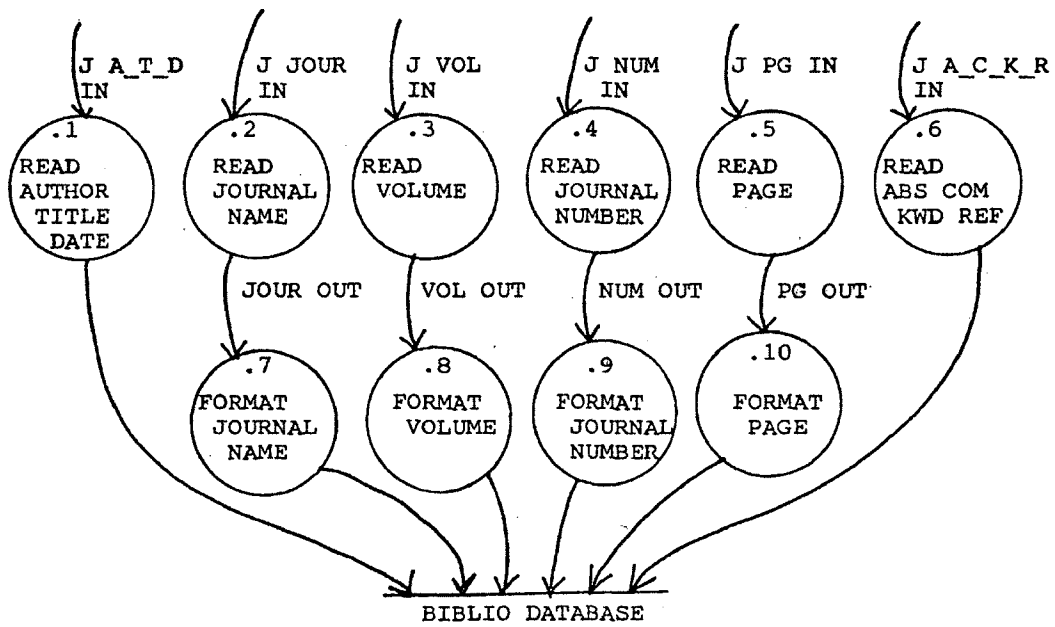


Figure 2-5B ADD JOURNAL ARTICLE (Diagram 1.2)

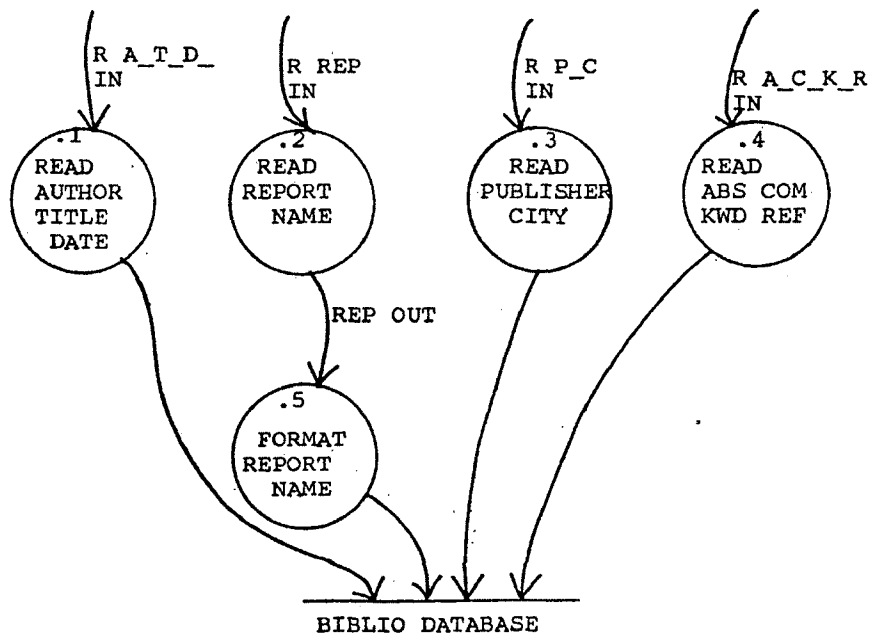


Figure 2-5C ADD TECHNICAL REPORT (Diagram 1.3)

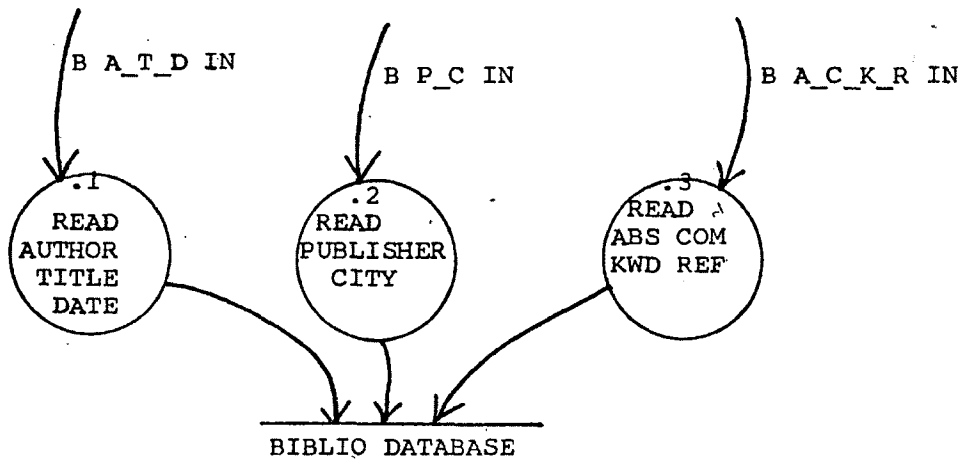


Figure 2-5D ADD BOOK (Diagram 1.1.1)

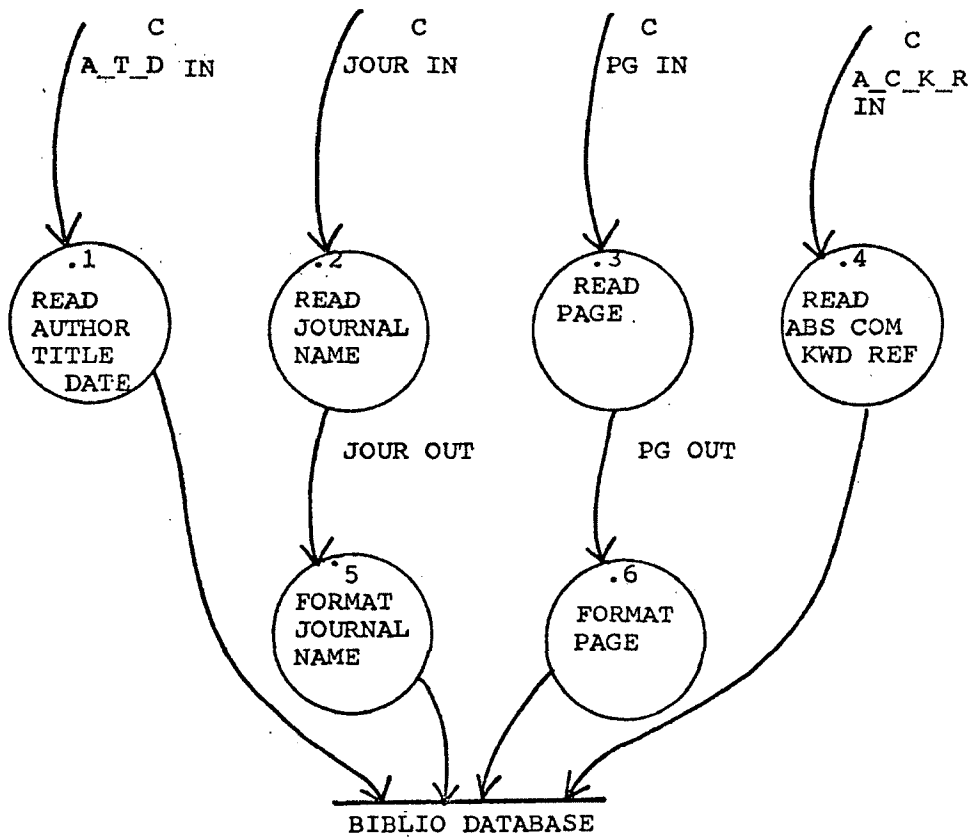


Figure 2-5E ADD CONFERENCE PROCEEDINGS (Diagram 1.1.2)

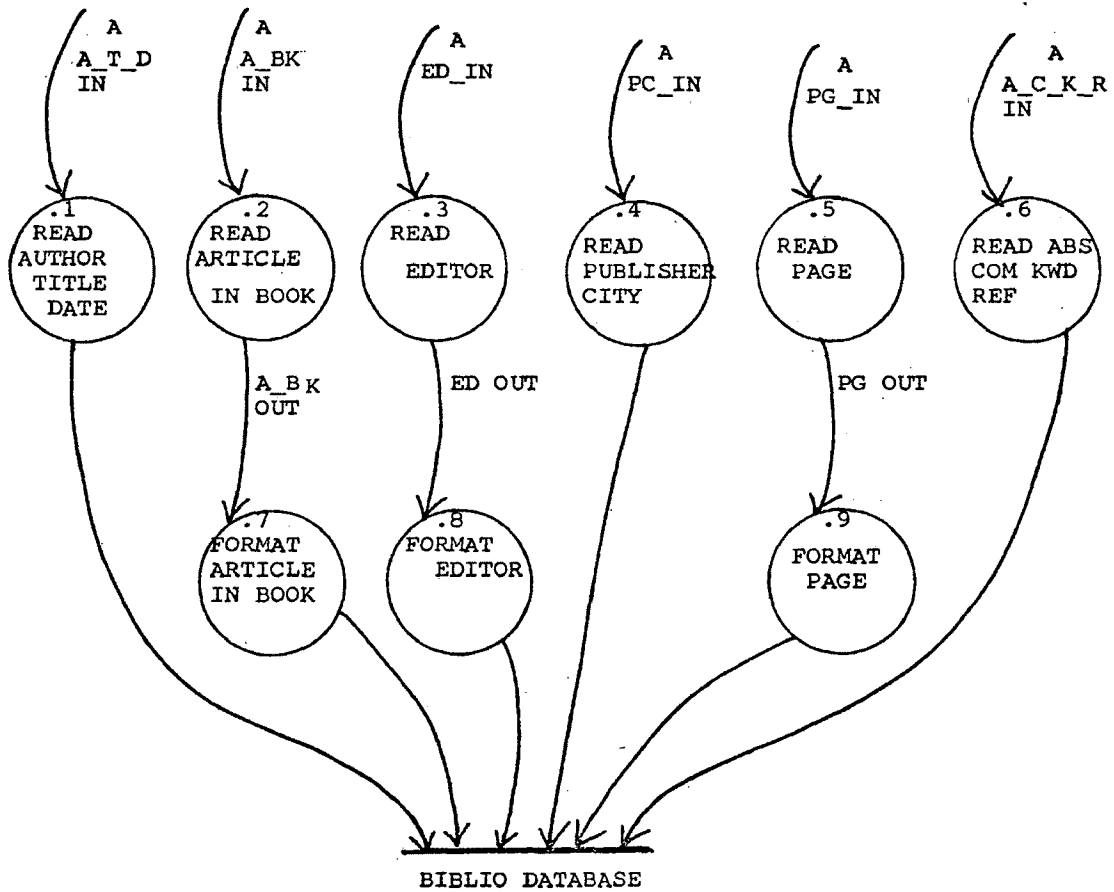


Figure 2-5F ADD ARTICLE IN BOOK (Diagram 1.1.3)

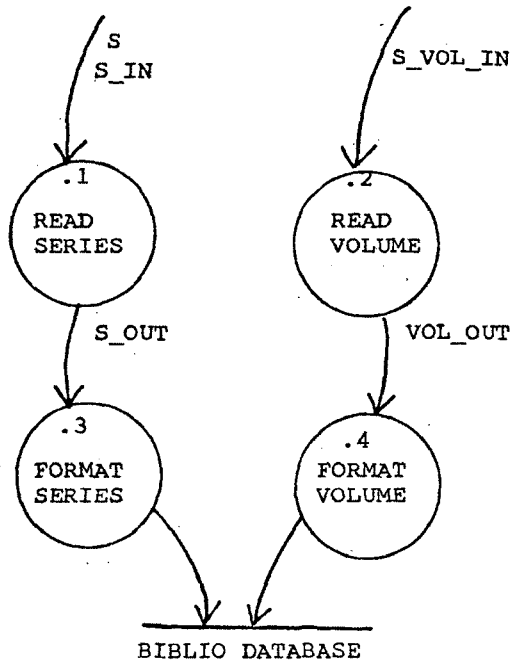


Figure 2-5G ADD MULTI-VOLUME SERIES (Diagram 1.1.4)

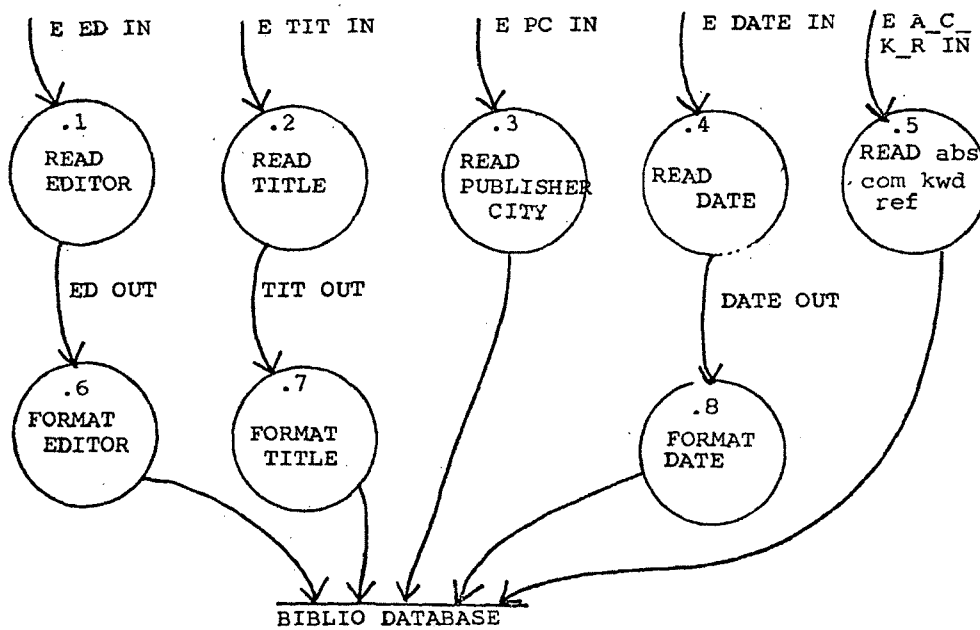


Figure 2-5H ADD EDITED BOOK (Diagram 1.1.5)

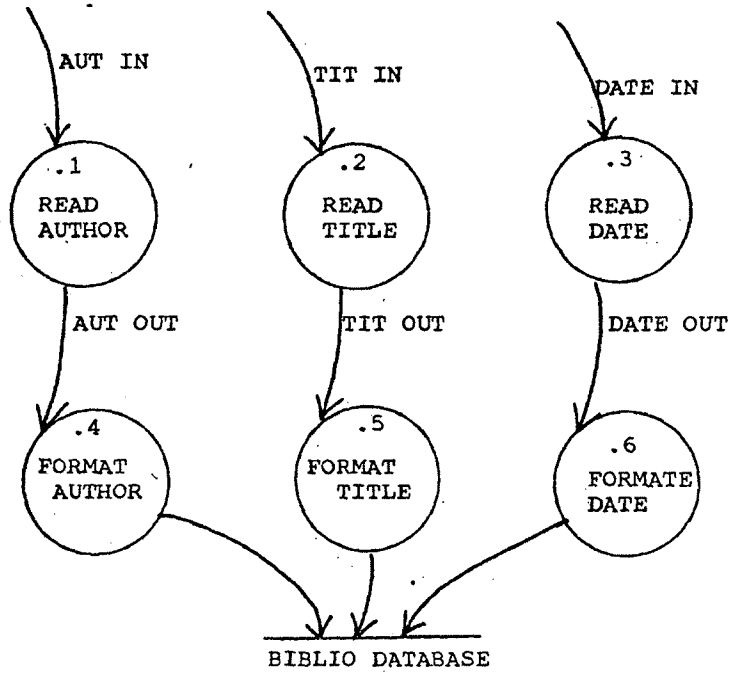


Figure 2-5I READ AUTHOR TITLE DATE (Diagram 1.1.1.1)

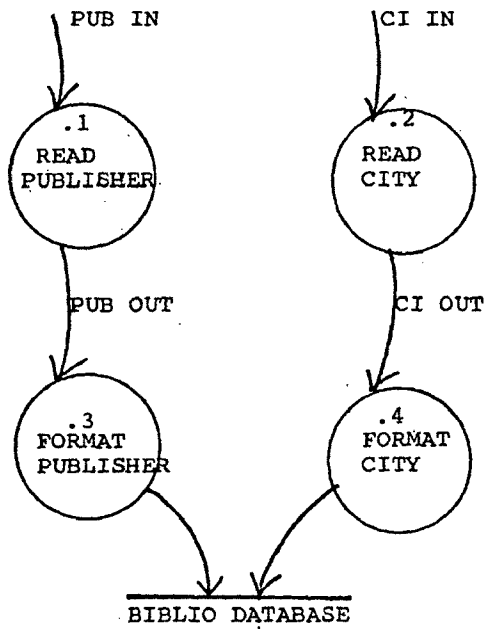


Figure 2-5J READ PUBLISHER CITY (Diagram 1.1.1.2)

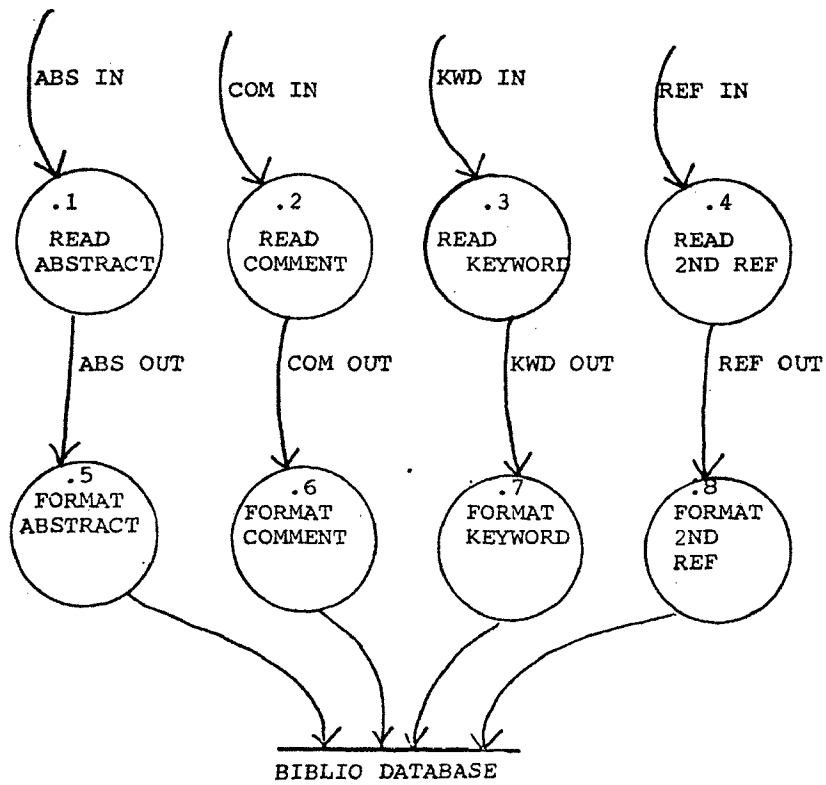


Figure 2-5K READ ABS COM KWD REF (DIAGRAM 1.1.1.3)

APPENDIX A-6

THE USER'S GUIDE
FOR
SUPERBIB -
SUPER BIBLIOGRAPHIC SYSTEM

by

A.Y. (Michelle) Liao

University of Montana
Missoula, Montana

TABLE OF CONTENTS

INTRODUCTION	153
BACKGROUND INFORMATION	155
FEATURES AND LIMITATIONS	158
AN OVERVIEW OF OPERATIONS	161
ADD REFERENCE ENTRIES	168
INQUIRE REFERENCE FILE	175
PRINT BIBLIOGRAPHIES	180
MODIFY REFERENCE ENTRIES	184
SORT REFERENCE ENTRIES	187

1. INTRODUCTION

Superbib, is a set of superior programs (preprocessor, user interface) for the Unix Bibliographic System, Unix-bib. SuperBIB, was developed to enable researchers, educators, writers, and members of the general public to easily prepare bibliographies.

The preparation of a bibliography is a perfect application for the computer, since the computer is good at recording, searching and sorting information. An automated bibliographic system can accept, search, print, or sort bibliographic information with a minimum of effort. The bibliographic information stored in the computer may be easily modified and camera-ready bibliographies may be generated at a touch of the finger.

Superbib is easy to use, due to the fact that it is menu-driven, screen-oriented and employs windows. SuperBIB provides a vivid dialogue between the user and itself. SuperBIB provides adequate on-line help and the user neither memorizes the command names, nor refers to this User's Guide constantly.

SuperBIB displays option menus on the user's terminal. To select a command, position the cursor next to the option desired using the space bar, then press the RETURN key.

Superbib subdivides a terminal screen into four windows. The first window displays greetings. The second window displays menus and allows the user to select commands. The third window prompts the user for information inputs, and accepts the inputs. The last window reports the status of SuperBIB. For example, a highlighted ADD indicates that Superbib is in the process of adding reference entries into a reference file.

2. BACKGROUND INFORMATION

2.1. DEFINITIONS

The reference file, and its components are defined below:

- (1) An entry element may be a citation (an author's name, a title, or a publication date, etc.), or an annotation (an abstract or comments).
- (2) A collection of all entry elements for a particular publication is called a reference entry.
- (3) A collection of reference entries is called a reference file.

2.2. A SAMPLE REFERENCE FILE

A sample reference file, and its components is illustrated in Table 2-1.

TABLE 2-1 A Sample Reference File and its Components.

COMMENTS	FILE AS SEEN ON SCREEN
	.
	.
	.
	%A M. Bishop
	%A L. Snyder
ref. <	%T The Transfer of Information
entry <	%J Proceedings of the 7th SOSP
	%P 45-54
ref. <	-- %D 1979
file <	
	%E R.A. DeMillo
reference ->	%E D.P. Dobkin
element	%E A.K. Jones
	%E R.J. Lipton
	%T Foundations of Secure Computation
	%I ACPRESS
	%D 1978
	^
key-letter ---	
	.
	.
	.

=====
 Note that a blank line separates each reference entry.

2.3. KEY-LETTERS

Currently Superbib recognizes the key-letters listed in Table 2-2.

TABLE 2-2 Key-letters Recognized by SuperBIB

Key-letter	Description
A	Author's name
B	Title of book containing article
C	City of publication
D	Publication date
E	Editor(s) name
I	Issuer (publisher)
J	Journal name
K	Search words (Key-words)
N	Issue number
O	Other information
P	Pages of article
R	Technical report name
S	Series title
T	Title
V	Volume number
X	Annotation

3. FEATURES AND LIMITATIONS

3.1. FEATURES

SuperBIB accepts, inquires, prints, alters, and sorts the bibliographic information. The SuperBIB features are discussed below.

- (1) Superbib accepts reference entry elements, such as citations (authors' names, titles, publication dates, etc.), annotations (abstracts and comments) in a friendly fashion. It stores this information in a file of the user's choice. Up to 3,500 characters of information may be accepted for an annotation (abstracts or comments). Up to 64 characters of the information may be accepted for other reference entry elements.
- (2) SuperBIB may search for and select reference entries which match all search words, i.e., authors' (or editors') last names, words from titles, publication dates, other search words (key-words), or a combination of these. All reference entries meeting all the search criteria will be selected.
- (3) SuperBIB may print regular and annotated bibliographies, either numbered or un-numbered, and either single- or double-spaced. In addition, a doubled-spaced citation may be printed along with a single-spaced annotation. SuperBIB may display a bibliography on the terminal screen, print it on a printer, or store it in a file of the user's choice.
- (4) SuperBIB may invoke vi, (or ed) text editor of the Unix environment to alter the reference file.

- (5) SuperBIB sorts bibliographic information by the senior authors' last names, and publication dates.
- (6) SuperBIB provides adequate on-line help. The current status of the Superbib execution is intact while the help message is displayed. A sample on-line help is illustrated in Table 3-1:

TABLE 3-1 A Sample SuperBIB On-line Help Message.

=====

Reference File : _____
Search-word File : _____
Inquiry-result File : _____
Bibliography File : _____
Sorted reference File : _____

To delete character, press DELETE.
To delete a line, press CTRL-u.
To advance the cursor, press SPACE BAR.
To select a command, press the RETURN key.
To escape to parent menus, press ESC keys.

=====

Note that Search-word File is a file holding search words. Inquiry-result File is a file holding the entries that match the search words. Bibliography File is a file holding the formatted bibliography.

3.2. LIMITATIONS

The limitations of SuperBIB are listed below:

- (1) Only one paragraph of annotations (abstracts, or comments) may be entered although up to 3500 characters may be accepted. (The documents of the Unix bibliographic system say that more than one paragraph of the annotations may be entered, but this claim is questionable.) SuperBIB displays a prompt, 'ANNOTATION', for both abstract and comment inputs. Abstracts and comments do not have separate prompts. Extracting and inserting the annotation (abstracts and comments) into a text file is not accommodated.
- (2) Reference entries may be selected, if these entries match all of the search words, i.e. the authors' (or editors') last names, words from titles, publication dates, other search words (key-words) and a combination of these. Entries matching a part of the search words cannot be selected in the current installation of SuperBIB. Wildcard searching is not accommodated.
- (3) Limited bibliographic styles may be printed. Printing a bibliography in other styles is not accommodated [1].
- (4) Altering bibliographic information without invoking a text editor is not accommodated.
- (5) Sorting bibliographic information by criteria other than the senior authors' last names and publication dates is not accommodated.

[1] Working knowledge of 'Nroff' and 'Bib' are required to acquire full service from the Unix bibliographic system.

4. AN OVERVIEW

The SuperBIB terminal screen of is subdivided into four windows. The first window, one-line in size, displays greetings. The second window, 11-lines in size, displays menus and accepts the user's selections. The third window, 11-lines in size, prompts the user for information, accepts input, and displays help messages (if applicable). The last window, 1-line in size, reports the status of Superbib. For example, a highlighted ADD indicates the user is adding a reference entry into the reference file.

Superbib displays menus on the terminal. To select a command, position the cursor next to the command desired by pressing the space bar, then press the RETURN key.

4.1. HOW TO INVOKE SUPERBIB

First log into a VAX computer on which SuperBIB is installed, then type 'sb' in lower case. If the user invokes SuperBIB successfully, a NAME A REFERENCE FILE menu will be displayed on the terminal screen (Table 4-1).

TABLE 4-1 The NAME A REFERENCE FILE Menu

```

*****
* SuperBIB - SUPER Bibliographic System *
*                                         *
*   <<< NAME A REFERENCE FILE. >>>   *
*                                         *
*   To terminate input, press RETURN.   *
*   To edit, press DELETE or CTRL-u.   *
*                                         *
* REF. FILE: [ ] _____             *
*                                         *
*****

```

Note that the symbol, [], indicates the cursor position when the menu first appears.

4.2. HOW TO EXIT SUPERBIB

To exit SuperBIB gracefully, select the EXIT option from the SELECT A COMMAND menu (to be explained in Section 4.4). If the user is not currently at this menu, he may press the ESC (escape) key repeatedly (if necessary) to reach this menu, then select the EXIT option.

4.3. HOW TO RECOVER FROM MISTAKES

Before the user presses the RETURN key to terminate his input, he may press the DELETE key to delete one character at a time; or he may type CONTROL-u to delete a line. After he has pressed the RETURN key, the EDIT

option from the SELECT A COMMAND menu may be used to correct the information in the reference file.

The user may press the ESC key to exit menus which he wishes to abort. He may press the ESC key repeatedly to go back to the SELECT A COMMAND menu. If the user presses the ESC key when adding an entry element, SuperBIB escapes to its parent level menu. Invoke an editor to delete this information, if desired. However, reference entry element information entered prior to ESC termination will be entered into the reference file.

4.4. SELECT A COMMAND MENU

The menu of SELECT A COMMAND is illustrated in Table 4-2.

TABLE 4-2 The SELECT A COMMAND Menu

```

*****
* SuperBIB - Super Bibliographic System *
*
* <<< SELECT A COMMAND. >>> *
*
*      [ ]  ADD ? *
*           INQUIRE ? *
*           PRINT ? *
*           EDIT ? *
*           SORT ? *
*           INVOKE NEW REFERENCE FILE? *
*           HELP ? *
*           EXIT ? *
*
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
*
*****

```

Note that the last line on the menu shows the contents of a status report.

The SELECT A COMMAND MENU is discussed below.

- (1) ADD, adds reference entries to a reference file of the user's choice,
- (2) INQUIRE, selects the reference entries matching all search words, i.e. names, words from titles, publication dates, key-words, or a combination of these,

- (3) PRINT, prints the different styles of regular and annotated bibliographies on the printer, displays it on the terminal, or stores it in a file of the user's choice.
- (4) EDIT, modifies a reference file.
- (5) SORT, sorts the bibliography according to the senior author's last name and publication dates.
- (6) INVOKE NEW REFERENCE FILE, invokes a reference filename for adding, searching, printing, etc.
- (7) HELP, provides on-line help messages.
- (8) EXIT, exit SuperBIB gracefully.

4.5. HOW TO EXIT SUPERBIB TEMPORARILY

The user may type CONTROL-z to return to the Unix Shell level whenever he desires. He may do anything which is acceptable at the Shell level. Then he may type 'fg' to return to SuperBIB.

Some of the Unix commands the user may need are discussed below.

4.6. HOW TO GET A LISTING OF FILES

To get a listing of files (or directories) type 'ls' at the Shell level. Naming your reference files with a standard extension (e.g., 'ref') is highly recommended. For example, the author names her reference file for this

thesis project 'thesis.ref'. In this case the user may type 'ls *.ref' to obtain a listing of all reference files.

4.7. HOW TO COPY FILES

To obtain a duplicate copy of your file, type 'cp your_file_name a_different_file_name' at the Shell level. It is recommended that you obtain a duplicate copy of important files, i.e. the reference files.

4.8. HOW TO REMOVE FILES

The user may name his files with standard names and these files will be removed (deleted) automatically when exiting SuperBIB. A listing of recommended (standard) file names is in Table 4-3.

TABLE 4-3 Recommended File Names

File Type	Recommended File Name
Search words	temp.kwd*
Inquiry results	temp.inq*
Bibliography	temp.bib*
Sorted references	temp.sorted*

Note that the '*' indicates any letters, numbers, or others.

4.9. HOW TO DISPLAY FILES

At the Shell level, type 'more your_file_name' to display the first page of your file. Once you are in the MORE program, press the space bar, or the RETURN key to see more of the display; type 'q' to quit prematurely.

4.10. HOW TO PRINT FILES

To obtain a hard-copy of a file, type 'lpr your_file_name' at the Shell level.

4.11. HOW TO GET A PRINTER-QUEUE STATUS REPORT

To learn the status of your print jobs, type 'lpq' or 'lpq +2' at the Shell level. The 'lpq +2' will display the printer-queue status every two seconds.

4.12. HOW TO LEARN MORE ABOUT UNIX

The user may type 'learn' at the Shell level to learn 'files', 'morefiles', 'macros (text processing)', 'eqn' editor, vi, or the C language. Note that this feature is not available in 'csvax', VAX-11/750 in the Computer Science Department.

5. ADD REFERENCE ENTRIES

5.1. SELECT A PUBLICATION TYPE

The menus of SELECT A PUBLICATION TYPE are illustrated in Table 5-1:

TABLE 5-1 The SELECT A PUBLICATION Menu

```
*****
* SuperBIB - Super Bibliographic System *
* <<< SELECT A PUBLICATION TYPE. >>> *
* [ ] BOOK ? *
* ARTICLE FROM JOURNAL ? *
* ARTICLE FROM BOOK ? *
* TECHNICAL REPORT ? *
* CONFERENCE PROCEEDINGS ? *
* COMPILED BOOK ? *
* MULTI-VOLUME SERIES ? *
* OTHERS ? *
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
* --- *
*****
```

Note that ADD of the status report is highlighted to indicate that the user is in the process of adding an entry to the reference file.

When adding a reference entry, the user needs to specify the type of publication which is to be added. For

the purpose of illustrating more ADD menus, the author assumes that the user wants to add a book entry to the reference file. The operations of adding the author's name of the book entry are described below. The operations for adding other entry elements is similar to those of adding authors' names, and therefore are omitted.

5.2. ENTER AN AUTHOR'S NAME

Snap-shots of menus for ENTER AN AUTHOR'S NAME are illustrated in Tables 5-2A to 5-2E:

TABLE 5-2A The ENTER AN AUTHOR'S NAME Menu
When it First Appeared

```
*****
* SuperBIB - Super Bibliographic System *
*                                         *
*   <<< ENTER AN AUTHOR'S NAME >>>   *
*                                         *
*           Enter given names first.   *
*           To bypass, press RETURN.   *
*                                         *
*                                         *
* AUTHOR [ ]                            *
*   -----                             *
*                                         *
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
* ---                                     *
*****
```

TABLE 5-2B The User Entered an Author's Name.

```

*****
* SuperBIB - Super Bibliographic System *
*                                     *
*   <<< ENTER AN AUTHOR'S NAME >>> *
*                                     *
*           Enter given names first. *
*           To bypass, press RETURN. *
*                                     *
* AUTHOR John Smith                  *
*   -----                          *
*                                     *
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
* ---                                  *
*****
    
```

Note that 'John Smith' is an assumed name of an author.

TABLE 5-2C SuperBIB Asks for the Possibility of Having Multiple Authors

```

*****
* SuperBIB - Super Bibliographic System *
*                                     *
*   <<< MORE AUTHORS' NAMES ? >>> *
*                                     *
*           [ ] No ?                  *
*           Yes ?                      *
*                                     *
* AUTHOR John Smith                  *
*   -----                          *
*                                     *
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
* ---                                  *
*****
    
```

TABLE 5-2D The User Indicated That He Has Multiple Author Names

```

*****
* SuperBIB - Super Bibliographic System *
*                                     *
*   <<<  MORE AUTHORS' NAMES ?  >>> *
*                                     *
*               No ? *
*             [*] Yes ? *
*                                     *
* AUTHOR John Smith *
* ----- *
*                                     *
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
* --- *
*****
    
```

Note that the symbol, [*], indicates the user has selected this option.

TABLE 5-2E SuperBIB Prompts the User For Author Name.

```

*****
* SuperBIB - Super Bibliographic System *
*                                     *
*   <<<  MORE AUTHORS' NAMES ?  >>> *
*                                     *
*               No ? *
*             [*] Yes ? *
*                                     *
* AUTHOR John Smith *
* ----- *
* AUTHOR [ ] *
* ----- *
*                                     *
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
* --- *
*****
    
```

When entering names to the reference file:

- (1) Multiple authors' (or editors') names are accommodated.
- (2) Type the author's first name first, then the last name.
- (3) SuperBIB treats the last word in a name entry as the last name, therefore special care has to be taken for some last names. If the last name is composed of two words, type a back slash between the two words. For example, the user has to type 'De [back slash] Marco' for the name 'De Marco'. If the author's name contains 'Jr.', type a comma before 'Jr.'.

5.3. TEMPLATES

Each type of publication has its own template (prompts for inputs). Prompts and dashed lines will be displayed on the user's terminal. The user is to type the input at the beginning of the dashed line. Leading white space is ignored.

5.3.1. TEMPLATE OF A BOOK ENTRY

Author
Title
Date
Publisher
City
Search word(s)
Annotation

5.3.2. TEMPLATE OF AN ARTICLE FROM A JOURNAL

Journal Title
Volume #
Number
Date
Page #
Author
Book Title
Search Key(s)
Annotation

5.3.3. TEMPLATE OF AN ARTICLE FROM A BOOK

Article Title
Date
Publisher
City
Page #
Author
Book Title
Search Key(s)
Annotation

5.3.4. TEMPLATE OF A TECHNICAL REPORT ENTRY

Report Title
Date
Publisher
City
Author
Article Title
Search Key(s)
Annotation

5.3.5. TEMPLATE OF CONFERENCE PROCEEDINGS

Journal title
Date
Page #
Author
Book Title
Search words(s)

5.3.6. TEMPLATE OF A COMPILED BOOK ENTRY

Editor
Book title
Publisher
Date

5.3.7. TEMPLATE OF A MULTI-VOLUME SERIES

Editor
Article Title
Series Title
Volume #
Page #
Publisher
Date
City
Author
Book Title

6. INQUIRE REFERENCE FILE

The user may search for reference entries matching certain search words, i.e. authors' (or editors') names, words from titles, publication dates, other search words (key-words) and a combination of these. Only the first six characters of the search words are significant. The user may choose to type the first six characters only, if applicable [2]. Search words shorter than three characters are ignored [3]. For example, if the user wants to search for entries that have something to do with the C language, the search word he provides cannot be 'c', due to the fact that any words less than three characters long are ignored. Enter an entry (e.g. 'c-language') as a key-word element, then search for 'c-language' instead.

6.1. SELECT A DESTINATION OF THE INQUIRY

The user may select a destination for the output of the inquiry results. The SELECT A DESTINATION menu is illustrated in Table 6-1.

[2] & [3] These are system defaults and may be changed.

TABLE 6-1 The SELECT A DESTINATION Menu
For Routing Inquiry Results.

```

*****
* SuperBIB - Super Bibliographic System *
*
* <<< SELECT A DESTINATION. >>> *
*
*      [ ] Terminal ? *
*      Printer ? *
*      File ? *
*      HELP ? *
*
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
* ----- *
*****

```

6.2. NAME A SEARCH-WORD FILE

6.2.1. NAME A FILE TO STORE SEARCH WORDS

The user may name a file to store his search words. A filename of 'temp.kwd*', where '*' could be any letter, number, or others, will be deleted automatically when exiting SuperBIB. The NAME A FILE TO STORE SEARCH WORDS menu is illustrated in Table 6-2.

TABLE 6-2 The NAME A FILE TO STORE SEARCH WORDS Menu

```

*****
* SuperBIB - Super Bibliographic System *
* *
* < NAME A FILE TO STORE SEARCH WORDS. > *
* *
* *
* SEARCH-WORD FILE : [ ] _____ *
* *
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
* ----- *
*****

```

Note that a filename of 'temp.kwd*' is recommended if the user wants to delete this file when exiting SuperBIB.

6.2.2. SEARCH-WORD FILE AVAILABLE ?

SuperBIB assumes that the user may have a search-word file available, if he has invoked the INQUIRE command more than once. In this case, the menu of SEARCH-WORD FILE AVAILABLE will be displayed (Table 6-3).

TABLE 6-3 The SEARCH-WORD FILE AVAILABLE Menu

```

*****
* SuperBIB - Super Bibliographic System *
*
* <<< SEARCH-WORD FILE AVAILABLE ? >>> *
*
*           [ ] No ? *
*           Yes ? *
*           HELP ? *
*
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
* ----- *
*****
    
```

Note that this menu appears only if the user has successfully created a search-word file.

6.2.3. NAME THE READY-MADE SEARCH-WORD FILE

TABLE 6-4 The NAME YOUR SEARCH-WORD FILE Menu

```

*****
* SuperBIB - Super Bibliographic System *
*
* <<< NAME YOUR SEARCH-WORD FILE. >>> *
*
* SEARCH-WORD FILE : [ ] _____ *
*
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
* ----- *
*****
    
```

Note that this menu appears only if the user has successfully created a search-word file and wishes to use the same search-word file.

TABLE 6-5 NAME A FILE TO STORE INQUIRY
RESULTS IF DESTINATION IS FILE.

```
*****  
* SuperBIB - Super Bibliographic System *  
* *  
* < NAME A FILE TO STORE INQUIRY RESULTS >*  
* *  
* INQUIRY FILE: [ ] _____ *  
* *  
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *  
* ----- *  
*****
```

Note that a filename of 'temp.inq*' is recommended if the user wants this file deleted when exiting SuperBIB.

7. PRINT BIBLIOGRAPHIES

7.1. SELECT A DESTINATION FOR THE BIBLIOGRAPHY

A bibliography may be printed on the user's terminal screen, or on a printer. The user may store the bibliography in a file of his choice. The SELECT A DESTINATION menu is illustrated in Table 7-1.

TABLE 7-1 The SELECT A DESTINATION Menu
For Printing a Bibliography

```
*****
* SUPERBIB-Super Bibliographic System *
*                                     *
* <<<  SELECT A DESTINATION.  >>>   *
*                                     *
*           [ ] Terminal ?           *
*           Printer ?                 *
*           File ?                     *
*           HELP ?                     *
*                                     *
* ADD INQUIRE PRINT EDIT SORT  EXIT HELP *
*           -----                    *
*****
```

7.2. SELECT BIBLIOGRAPHY STYLES7.2.1. REGULAR OR ANNOTATED

TABLE 7-2 SELECT A BIBLIOGRAPHY TYPE Menu

```

*****
* SUPERBIB-Super Bibliographic System *
*                                     *
* <<< SELECT A BIBLIOGRAPHY TYPE. >>> *
*                                     *
*           [ ] Regular Bibliography ? *
*           Annotated Bibliography ? *
*           HELP ?                     *
*                                     *
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
*           ----- *
*****

```

7.2.2. NUMBER YOUR BIBLIOGRAPHY MENU

TABLE 7-3 The NUMBER YOUR BIBLIOGRAPHY Menu

```

*****
* SUPERBIB-Super Bibliographic System *
*                                     *
* <<< NUMBER YOUR BIBLIOGRAPHY ? >>> *
*                                     *
*           [ ] No ?                   *
*           Yes ?                       *
*           HELP ?                       *
*                                     *
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
*           ----- *
*****

```

7.2.3. SINGLE OR DOUBLE SPACED

TABLE 7-4 The SELECT A SPACING STYLE Menu
For a Regular Bibliography

```
*****
* SUPERBIB-Super Bibliographic System *
*                                     *
* <<< SELECT A SPACING STYLE ? >>> *
*                                     *
*      [ ] Single Spaced ?           *
*      Double Spaced ?               *
*      HELP ?                         *
*                                     *
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
*      ----- *
*****
```

Note that this menu appears for the regular bibliography only. The user may single or double space the citations.

TABLE 7-5 The SELECT A SPACING STYLE Menu
For an Annotated Bibliography

```
*****
* SUPERBIB-Super Bibliographic System *
*                                     *
* <<< SELECT A SPACING STYLE ? >>> *
*                                     *
*      [ ] Single Spaced ?           *
*      Double Spaced ?               *
*      Double & Single Spaced ?     *
*      HELP ?                         *
*                                     *
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
*      ----- *
*****
```

Note that this menu appears for the annotated bibliography only. The user may single or double space the citations, or single space citations and double space annotations.

7.3. NAME A FILE TO STORE THE BIBLIOGRAPHY

The NAME A FILE TO STORE THE BIBLIOGRAPHY menu is illustrated in Table 7-6.

TABLE 7-6 NAME A FILE TO STORE THE
BIBLIOGRAPHY Menu

```

*****
* SUPERBIB-Super Bibliographic System *
* * * * *
* <<< NAME A FILE TO STORE BIB. >>> *
* * * * *
* [ ] *
* ----- *
* * * * *
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
* ----- *
*****

```

Note that a filename of 'temp.bib*' is recommended if the user wants to delete this file automatically when exiting SuperBIB.

8. MODIFY REFERENCE ENTRIES

8.1. SELECT A MODIFYING METHOD

The user may select a method to modify the reference file. Currently, the user may use a text editor ('vi' or 'ed') to modify the reference file. Modifying without invoking an editor is not implemented. The SELECT A MODIFYING METHOD menu is illustrated in Table 8-1.

TABLE 8-1 SELECT A MODIFYING METHOD Menu

```
*****
* SuperBIB - Super Bibliographic System *
* <<< SELECT A MODIFYING METHOD. >>> *
*          [] Text editor ? *
*          Questions & answers ? *
*          HELP ? *
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
*          ---- *
*****
```

8.2. SELECT AN EDITOR

The user may select his favorite editor to modify a reference file. Currently, the user may invoke the vi, or

ed editor. The author suspects that when invoked from SuperBIB, the ed editor may not have its full editing power, due to the fact that SuperBIB is screen-oriented and the ed editor is a line-oriented editor. The SELECT AN EDITOR menu is illustrated in Table 8-2.

TABLE 8-2 The SELECT AN EDITOR Menu

```

*****
* SuperBIB - Super Bibliographic System *
*                                     *
*   <<<  SELECT AN EDITOR.  >>>   *
*                                     *
*           [ ] vi ?                *
*                                     *
*                   ed ?           *
*                                     *
*                   HELP ?         *
*                                     *
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
*                                     *
*                   ----           *
*****

```

8.3. A VI EDITOR REFERENCE GUIDE

The quick reference guide for using the vi editor is presented below.

You may type:

- (1) a 'G' to move cursor to the end of the file; a '#G' to move cursor to the #th line of the file. For example, '10G' moves the cursor to the 10th line from the current line.
- (2) an 'h' to move cursor to your left,
- (3) a 'j', or '+', to move cursor down the screen,
- (4) a 'k', or '-', to move cursor up the screen,
- (5) a 'l' to move cursor to your right,
- (6) an 'x' to delete a character. For example, '5x' deletes five characters from the current character.
- (7) a 'dd' to delete a line. For example, '5dd' deletes five lines from the current line.
- (8) an 'i' to enter insert mode; Press ESC to exit insert mode,
- (9) a 'o' to insert line(s) below the current line, Press ESC to exit insert mode,
- (10) a 'O' to insert line(s) above the current line, Press ESC to exit insert mode,
- (11) a 'CTRL-g' to learn what current line number is,
- (12) a 'ZZ' to exit the vi editor.

9. SORT REFERENCE ENTRIES

The user may sort the reference entries according to the last names of the senior authors, and publication dates.

9.1. SELECT A DESTINATION FOR SORTING

The possible destinations for sorted reference entries are the terminal screen, printer queue, or a file of the user's choice. Route the sorted entries to a file, if the user wants to have the bibliography (or reference file) sorted. The current reference filename will be changed to this sorted reference filename if a sorted reference file is created successfully.

TABLE 9-1 The SELECT A DESTINATION Menu FOR SORTED REFERENCE ENTRIES.

```

*****
* SUPERBIB-Super Bibliographic System *
*                                     *
* <<< SELECT A DESTINATION. >>> *
*                                     *
*           [ ] Terminal ? *
*             Printer ? *
*             File ? *
*             HELP ? *
*                                     *
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
*                                     *
*****
    
```

Note: Select FILE when sorted bibliography or reference entries are desired later.

9.1.1. NAME A FILE TO STORE SORTED ENTRIES

TABLE 9-2 The NAME A FILE TO STORE SORTED REFERENCE ENTRIES Menu

```

*****
* SUPERBIB-Super Bibliographic System *
*                                     *
* <<< NAME A FILE TO STORE SORTED REF.>>> *
*                                     *
* SORTED FILE [ ] *
*           ----- *
*                                     *
* ADD INQUIRE PRINT EDIT SORT EXIT HELP *
*                                     *
*****
    
```

Note that a filename of 'temp.sorted*' is recommended if the user wants to delete this file automatically when exiting SuperBIB.

BIBLIOGRAPHY

1. User's Guide to BIBLIO, a Computerized Bibliographic System, University of Chicago, Chicago, Ill, 1981.
2. Using SEM Version SEM 1.3 under IBM VM/CMS, University of Michigan; Ann Arbor, July 1982.
3. UNIX Programmer's Manual, UNIX TIME-SHARING SYSTEM , 2, Bell Labs, 1983.
4. Allman, E.P., -ME Reference Manual, University of California.
5. Allman, E.P., Writing Papers with NROFF Using -ME, University of California, 1979.
6. Arnold, Kenneth C.R.C., Screen Updating and Cursor Movement Optimization: A Library Package, University of California, Berkeley.
7. Bergland, G.D. and R.D. Gordon, Tutorial: Software Design Strategies, IEEE.
8. Boehm, B.W., Software Engineering Economics, Prentice-Hall, 1981.
9. Budd, Timothy A. and Gary M. Levin, "A UNIX Bibliographic Database Facility," TR 82-1, University of Arizona, Tucson, Arizona, 1982.
10. Burger, Maria and E. Bujdoso, Computerized Bibliography on Homogeneous Chemical Oscillating Reactions (LIBHCOR) Search Manual, University of Montana, Missoula, Montana, 1983.
11. DeMarco, Tom, Structured Analysis and System Specification, Yourdon Inc., New York, New York, March 1978.
12. Good, M.D., J.A. Whiteside, D.R. Wixon, and S.J. Jones, "Building a User-Derived Interface," Communications of the ACM, vol. 27, no. 10, pp. 1032-1043, 1984.
13. Hancock, L. and M. Krieger, The C Primer, McGraw-Hill, 1982.
14. Jones, J.C., "Design Methods," in Tutorial on Software Designing Techniques, ed. A.I. Wasserman, IEEE Computer Society.
15. Kernighan, Brian W. and Dennis M. Ritchie, The C Programming Language, Prentice-hall, Englewood Cliffs, New Jersey, 1978.

16. Kernighan, Brian W. and Rob Pike, The UNIX Programming Environment, Prentice-hall, Englewood Cliffs, New Jersey, 1984.
17. Nicklin, P. J., The SPMS Software Project Management System, University of California, Berkeley, 1983.
18. Page-Jones, Meilir, "Transform Analysis," in Tutorial: Software Design Strategies, IEEE Computer Society.
19. Page-Jones, Meilir, "Transaction Analysis," in Tutorial: Software Design Strategies, IEEE Computer Society.
20. Parnas, D. L., "Designing Software for Ease of Extension and Contraction," in Tutorial: Software Design Strategies, pp. 109-119, IEEE, 1981.
21. Stevens, W.P., G.J. Myers, and L.L. Constantine, "Structured Design," in Tutorial: Software Design Strategies, IEEE Computer Society.
22. Waite, M., UNIX Primer Plus, Howard W. Sams Co..
23. Yourdon, E., "Top-Down Design and Testing," in Tutorial: Software Design Strategies, pp. 57-78, IEEE.
24. Yourdon, Edward and Larry L. Constantine, Structured Design, Prentice-Hall, Englewood Cliffs, New Jersey, 1979.