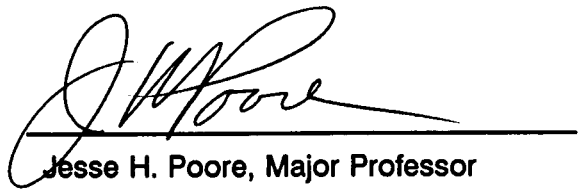
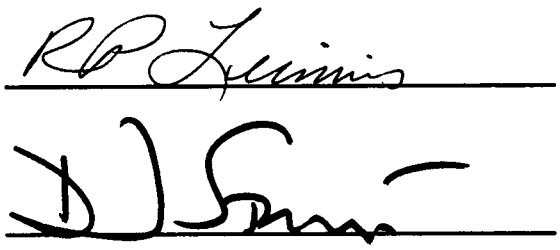
To the Graduate Council:

I am submitting herewith a thesis written by Russell L. Crisp entitled "Database Design for the Cleanroom Software Engineering Environment." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.



Jesse H. Poore, Major Professor

We have read this thesis
and recommend its acceptance:



Accepted for the Council:



Associate Vice Chancellor
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Russell L. Cripe

Date 10/16/92

DATABASE DESIGN

FOR THE

CLEANROOM SOFTWARE ENGINEERING ENVIRONMENT

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Russell L. Crisp

December 1992

ACKNOWLEDGMENTS

I want to thank Dr. Jesse Poore, my major advisor, for affording me the opportunity to be a part of his research team. His ideas and suggestions have been very helpful and are greatly appreciated.

I want to thank Dr. Ronald Leinius and Dr. David Straight for their service on the thesis committee and for their helpful suggestions.

Thanks is also extended to Dr. Carmen Trammell for her assistance and encouragement during the preparation of this thesis. Her thoughtful, humorous demeanor and willingness to help have been very much appreciated.

I want to thank Hailong Mao for his patience and support and for suffering through the many questions I've asked.

I want to thank Letitia Griffey for her editorial suggestions. Her willingness to read this thesis and provide comments have been very helpful.

I want to extend a whole-hearted thanks to Pamela Thomas for her encouragement and generous assistance during the past four and half years, and especially during preparation of this thesis. I am thankful for her friendship and for the contributions she has made to my personal and professional development.

I want to thank Ron Wantland for his understanding and generous support. I am very grateful for his assistance and concern.

I want to thank my parents, Gene and Joyce Crisp, for their encouragement and support. They have always expressed a constant faith in my abilities, and they have taught me to have faith in myself.

I want to thank my wife Sherri, who married me in 1989 with full knowledge that I had several years of graduate study still ahead. Her support and patience over the past four years have been remarkable. I appreciate her more than I could ever put into words.

Finally, I want to thank our unborn child, who unknowingly gave me added incentive to complete this thesis before our expected due date.

ABSTRACT

The primary purpose of this research is to design a database for the Cleanroom Software Engineering Environment system. Eight increments of the system have been implemented using IBM C/2, OS/2, and Presentation Manager. Metrics from design, verification, engineering changes, and certification are needed. This thesis documents an attempt to design a comprehensive database, data entry screens, and reports to collect and maintain metrics from all aspects of Cleanroom development.

This thesis documents the data structures and design of the existing Cleanroom Software Engineering Environment system in order to establish a foundation for incorporating a new database design. References to the existing design are used to provide a clear understanding of the proposed changes.

Based upon a comprehensive search for database software tools meeting the system requirements, Q+E Database Library by Pioneer Software was chosen to aid developers in implementation of the proposed database. Recommendations are made in regard to the use of the Q+E Database Library software.

A database dictionary details proposed changes and additions to the current data structures. The proposed database structures contain all the information required for metric collection and reporting.

Data and processing flow details are presented with the incorporation of the proposed database design. Data entry screens are shown with explanations of all updates and validations. Proposed project development reports have been designed and the data sources for each field are provided.

TABLE OF CONTENTS

1 INTRODUCTION 1
Cleanroom Software Development Method 1
Cleanroom Software Engineering Environment 3
Statement of the Problem 3
Scope of the Thesis 4

2 CURRENT DATA STRUCTURES 6
C Design Language Data 6
C Design Language (CDL) 6
C Code (C) 6
CDL Error File (ERR) 6
Markov Model Data 6
From-To-Frequencies (FTF) 6
From-To-Probabilities (FTP) 7
Markov Usage Model Errors (SER) 7
Usage Model Editor Analytical Results (FRE) 7
Symbols (SYM) 7
Markov Model Notation (MMN) 7
Undefined Symbol Errors (UND) 7
Certification Model Report (CMR) 8
Testing Markov Chain (TMC) 8
Testing Matrix Estimation Report Data (EST) 8
Test Case Sequences (SEQ) 8
Failure Data 9
Testing Failure Data (TST) 9
Time to Failure Data (TTF) 9
Query Manager Database (DB) 9
Query Manager Logical Tables 9
Project Logical Table 10
Version Logical Table 10
Failure Logical Table 10
Engineering Changes Logical Table 10

3 CURRENT SYSTEM PROCESSING AND DATA FLOW 11
CRE Overview 11
Cleanroom Environment 12
Development Assistant 13
Box Structure Method 13
Verification Questions 14
CDL Preprocessor 14
Certification Assistant 16
Usage Model Editor 17
Test Case Generator 19
Testing Markov Chain 20
Certification Model 22

Management Assistant 23
Planning 23
Policies 24
Reports 25
Database 25
Policy Generation 29

4 RESEARCH OF DATABASE SOLUTION 31
Requirements 31
Environment 31
Marketing Considerations 31
Performance 32
Solutions Researched 32
OS/2 Database Manager 32
In-House Designed C Database Product 33
Q+E Database Library 34

5 PROPOSED DATABASE SOLUTION 36
Database Software Solution 36
Database Methodology 36

6 PROPOSED DATA STRUCTURES 38
Project Data 38
Project Table 38
Increment Table 39
Failure Data 39
Version Table 39
Failures Table 40
Engineering Changes Table 41
Project Development Data 42
Verification Table 42
Errors Found Table 43
Verification Attendance Table 43
Hours Worked Table 44
Count Table 45
Totals Table 46
Reliability Model Data 47
Certification Reliability Model File 47
Policy File 48

7 PROPOSED CRE ADDITIONS AND MODIFICATIONS 49
Proposed CRE Overview 49
Project Identifier 50
Project Pull-Down Menu 52
Increment Pull-Down Menu 55
DBS Subsystem Replacement 58
Versions 59
Failures 62

Engineering Changes 66
Verification Sessions Data 68
Verification Errors Found 71
Verification Attendance 74
Hours Worked 76
Counts 79
Totals Table Update 81
CDLP Updates 81
Usage Model Editor Updates 82
Policies Subsystem Development 83
CM Subsystem Modification 87
Testing Markov Chain Table Access 89
File Selection Modifications 90
Reports Subsystem Development 91

8 DATABASE ACCESS METHODOLOGY 92
Installation 92
QELIB User's Guide 92
Data Access 93
Database Connection 93
SQL Statements 93
Data Value Binding 93
Data Value Fetching 95
Closing SQL Statement 96
Disconnecting the Database Driver 96
SQL Select Statements 97
SQL Update Statements 98
SQL Insert Statements 99
SQL Delete Statements 99
Sample Programs 100

9 FUTURE SUGGESTIONS 101
IBM C Set/2 101
SPSS for OS/2 101
Existing Reports 101
QELIB Versions 101

BIBLIOGRAPHY 103

APPENDICES
A PROJECT DEVELOPMENT REPORTS (Figures A.1 - A.11) 106
A.1 Increment Information Report 107
A.2 Version Information Report 108
A.3 Failure Report 109
A.4 Failure Report Continued 110
A.5 Engineering Changes Report 111

A.6 Verification Report 112
A.7 Compilation Report 113
A.8 Execution Report 114
A.9 Hours Worked By Week Ending Date 115
A.10 Hours Worked By Member ID 116
A.11 Verification Sessions Report 117

B PROJECT REPORTS DATA SOURCE 118
Increment Information Report Data Source 119
Version Information Report Data Source 120
Failure Report Data Source 121
Failures Caused by Code Written in Current Increment Data Source 122
Failures Caused by Code Written in Previous Increment Data Source 123
Failures Caused by Code Written in an Engineering Change Data Source 124
Failures Caused by Reused Code Data Source 125
Engineering Changes Report Data Source 126
Verification Report Data Source 127
Compilation Report Data Source 130
Execution Report Data Source 134
Hours Worked By Week Ending Date Data Source 138
Hours Worked By Member ID Data Source 139
Verification Sessions Report Data Source 140

C LIST OF ABBREVIATIONS 141

VITA 143

LIST OF FIGURES

Figure 3.1	Current CRE Data Flow	11
Figure 3.2	Cleanroom Environment Window	12
Figure 3.3	Development Assistant	13
Figure 3.4	CDL Preprocessor Window With Pull-Down Menus	14
Figure 3.5	CDL Preprocessor - Open Window	15
Figure 3.6	Certification Assistant	16
Figure 3.7	Usage Model Editor Window With Pull-Down Menus	17
Figure 3.8	Test Case Generator Window With Pull-Down Menus	19
Figure 3.9	Testing Markov Chain Window With Pull-Down Menus	20
Figure 3.10	Inference Mode Dialogue	21
Figure 3.11	Certification Reliability Model Window With Pull-Down Menus	22
Figure 3.12	Management Assistant	24
Figure 3.13	Certification Database Actions Window	26
Figure 3.14	Add Version(s) Window	27
Figure 3.15	Add Failures(s) Window	28
Figure 3.16	Add Engineering Change(s) Window	28
Figure 3.17	Policy Menu Window	29
Figure 7.1	Proposed CRE Data Flow	49
Figure 7.2	Proposed Cleanroom Environment Window	50
Figure 7.3	Project Identifier Window	51
Figure 7.4	Project Selection Window	52
Figure 7.5	Edit Project Window	53
Figure 7.6	Add Project Window	54
Figure 7.7	Increment Selection Window	55
Figure 7.8	Edit Increment Window	56
Figure 7.9	Add Increment Window	57
Figure 7.10	Database Window	59
Figure 7.11	Version Data Window	60
Figure 7.12	(View/Edit/Add) Version Data Window	61
Figure 7.13	Failures Data Window	62
Figure 7.14	(View/Edit/Add) Failure Data Window	63
Figure 7.15	Engineering Changes Window	66
Figure 7.16	(View/Edit/Add) Engineering Change Data Window	67
Figure 7.17	Verification Sessions Data Window	69
Figure 7.18	(View/Edit/Add) Verification Session Data Window	70
Figure 7.19	Verification Errors Found Data Window	72
Figure 7.20	(View/Edit/Add) Verification Errors Found Window	73
Figure 7.21	Verification Attendance Data Window	74
Figure 7.22	(Edit/Add) Verification Attendance Window	75
Figure 7.23	Hours Worked Data Window	77
Figure 7.24	(Edit/Add) Hours Works Window	78
Figure 7.25	Counts Data	79
Figure 7.26	Policies Window	83
Figure 7.27	Policy Selection Window	84
Figure 7.28	Create Reliability Model File Window	85
Figure 7.29	(Edit/View/Add) Policy Window	86
Figure 7.30	Certification Reliability Model Window	88
Figure 7.31	Certification Reliability Model Selection Window	89
Figure 7.32	Reports Window	91

CHAPTER 1

INTRODUCTION

Cleanroom Software Development Method

The Cleanroom Software Development Method (hereafter referred to as Cleanroom) is a method for team development of software under statistical quality control. Cleanroom is now more than a decade old, and research and refinement continues. It is becoming more popular with software development organizations concerned with improving quality and reducing costs.

The most dramatic principle of Cleanroom is that there is no program debugging or testing prior to independent, statistical, reliability certification testing. This concept may discourage some developers. Developers willing to adhere to the strict rules of Cleanroom are rewarded with measurable reliability and near error-free code.

The name of the method is derived from the semiconductor industry where practices are enforced to prevent defects in design and impurities in process (An Overview of the Cleanroom Software Development Process, p. 2). The Cleanroom process seeks to achieve the same care for perfection as does the semiconductor industry, leading to greater product quality and improved productivity.

Cleanroom has the operational goal of one-hundred percent reliable performance of software. There are six main components of the development process:

- Tightly controlled team development
- The Cleanroom life cycle for incremental development
- Box structure method of analysis and design
- Functional verification
- Statistical testing
- Certification

A brief description of each component is provided below.

Cleanroom requires a team effort in software development. Typically, teams consist of four to eight members. The team leader interacts with the customer, manages strict adherence to Cleanroom, builds teamwork, and is usually the member most knowledgeable about the formal methods of Cleanroom.

Development of a system must occur in an accumulation of separate increments. Each increment is fully developed and certified. The life cycle for each increment is specification, design, development, and certification.

The entire system and each increment is designed using the box structure method. The box structure method provides a hierarchy of black box, state box, and clear box descriptions. The black box includes the highest level of stimulus and response.

Transition rules in the black box provide a state-free and procedure-free description of the function. The state box description introduces stored information as state data. The clear boxes are procedures that may be defined directly from the state boxes or black boxes, depending on the specifics of the module.

Functional verification is performed at regular intervals by the development team. In verification sessions, the teams conduct a detailed theory-based examination of code to verify that the code is correct. All mistakes are corrected and changes are verified.

Utilizing the box structure method, Cleanroom developers design complete and consistent specifications. The specifications serve as a basis for comparison during functional verification. Mathematical comparison of code is performed to determine the correctness of code. "Algebraic concepts are used to reduce the problem of determining compound program correctness to the problem of determining correctness of constituent primes." (Structured Programming, p. 213) Functional verification achieves code correctness through intellectual control, rather than through trial and error debugging.

A separate testing team (other than the developers) should be designated to conduct the statistical testing procedures. Based on an understanding of the usage environment, input sequences are randomly generated from a probability distribution of user inputs. The generated sequences are called test cases and are used to certify the reliability of the software product. Appropriate responses, as understood by the customers, are associated with each test case. Currently, stochastic grammars and Markov chains are used in developing the testing procedures.

After an increment has been coded, and the developers are all in agreement that the code is correct, statistical testing is performed. The results are used in the certification model to calculate the reliability values and to demonstrate that the software meets the required level of reliability. The certification process records all failures (even cosmetic variances from specifications). Failures require that engineering changes be made to the code and that the changed code be verified before statistical testing continues. Changes to the code result in a new version of the software. The certification model uses a Mean Time To Failure (MTTF) benchmark to demonstrate the reliability level of a version of the product. Statistical testing continues until the MTTF level is sufficiently low. Recently, the Testing Markov Chain model has been used in addition to the Certification Model. The Testing Markov Chain uses a stopping criterion based on usage model coverage, as well as the alternative reliability and Mean Time Between Failures (MTBF).

Cleanroom is not easily understood or performed. The process requires extensive training and practice before a developer is proficient. The concepts of the process are not entirely new. Teams have been developing software and performing walk-through trials for many years. However, the ability to provide a statistically valid statement regarding the reliability of software is new. Improvements to the process are consistently being sought, and research is being conducted to improve all aspects of Cleanroom.

Although several significant software systems have been developed using Cleanroom, there is a lack of tools to assist in its implementation. The Cleanroom Software Engineering Environment is an integrated software tool under development for use by

Cleanroom teams and their managers.

Cleanroom Software Engineering Environment

The Cleanroom Software Engineering Environment (CRE) is being designed and written in the Software Quality Research Program at the University of Tennessee. CRE is being developed as a tool to assist any software team in developing systems using Cleanroom.

CRE is funded by IBM Westlake Programming Laboratory, which required that it be written in IBM C/2 using OS/2 with Presentation Manager. Utilizing Cleanroom, each increment is designed, written, documented, and implemented by a team of faculty, graduate, and undergraduate students.

CRE is initiated from an OS/2 main window. Three selections are initially displayed: 1) Development Assistant, 2) Certification Assistant, and 3) Management Assistant. Each of these may be selected by using a mouse and clicking twice on the icon window. Processing continues with subsystem windows displaying the pull-down menus for that subsystem's functions.

This thesis was written after the implementation of the eighth increment of CRE. Metrics associated with the project show 14,556 lines of C Design Language and 11,243 lines of C code in the current implementation of CRE. Development of increments nine and ten occurred during the preparation of this thesis.

Statement of the Problem

The Database subsystem of CRE was designed and implemented using the OS/2 Version 1.3 Database Manager. A database was created using Query Manager (which is a component of Database Manager, and is a menu driven application providing access to Database Services) with four tables - project, versions, failures, and engineering changes. The user accesses the CRE database by clicking on the Database subsystem. This action initiates the Query Manager logon and passes control to the Query Manager menu facility. The user updates the database through the Query Manager menus. The user may run a policy which produces an inter-fail report that may be saved as an ASCII file. The ASCII file may be used to generate input data to any reliability model.

Most data facilities used by CRE do not incorporate the Database subsystem. The database tables are not required in order to access any of the other subsystem functions.

It has been determined that the current approach to the database design does not provide the performance or the amount of information required for CRE. Several problems are noted:

- The Query Manager menu application initiation is slow, requires logon identification, and a password (even though a password is not required to

access other data features of CRE).

- Use of the Query Manager menu application entails loss of control by CRE resulting in a loss of continuity of CRE's interactive features and a limitation to the validation process.

- Using the Query Manager menu application as the only database tool limits access to data by other subsystems.

- CRE is being designed and implemented with data created and used in ASCII files. Users of CRE must provide data by creating several ASCII files before being able to access certain processes provided in CRE. CRE does not provide any means of updating the ASCII files.

- The data currently included in the Query Manager database design are not sufficient for the needs of CRE.

- The data entered into the Query Manager menus are not incorporated into the other functions of CRE. Failure data must be duplicated in other ASCII files for use by CRE. (The Policy menu does offer the ability to create an ASCII file of the failure report. If the user provides the file extension .DB, the file may be used by the Certification Model Report subsystem.)

- The Query Manager menus do not provide a list of existing projects. The user must type the name of the project. The project name is case sensitive, which may create naming problems.

- The user must type the project name for each of the logical tables in the Query Manager menus, even though the project has been previously selected.

- Date fields require a proper format, but the format is not available through help.

- System error messages are presented when the user attempts to add duplicate indexed fields where duplicates are not allowed. The Database Manager error number is given, but the error is not easily determined.

Scope of the Thesis

The purpose of this thesis is to design a replacement Database subsystem for CRE. This design will not only document a replacement of the current Database subsystem, but will impact other functions of CRE. The database subsystems should effectively eliminate the problems listed above. The thesis highlights include:

- Documentation of the current data and control flows for CRE
- Research in determining an appropriate database solution that will meet the requirements of CRE
- Design of proposed data structures for CRE
- Documentation of a data dictionary
- Design of proposed interactive processes of the Database subsystem
- Documentation of the interfaces between existing modules and the database
- Documentation of the C code requirements for database table access

This thesis does not include programming, certification, or implementation of the database interface. The thesis will provide a detailed database design sufficient for an implementation effort.

CHAPTER 2

CURRENT DATA STRUCTURES

CRE accesses twenty-one data structures. These structures include seventeen ASCII files and four Query Manager logical tables. ASCII files have a unique file extension name which identifies the file type. The primary name of the file is user defined. File extension names are shown in parenthesis in the subheadings that follow. This chapter provides a brief description and use of each data structure. The format structure of each ASCII file is detailed in the CRE increment eight documentation.

C Design Language Data

C Design Language (CDL)

The CDL ASCII files are user defined black boxes, state boxes, and clear boxes written in the CDL. There should be one CDL file for each increment of a project.

C Code (C)

The CDL Preprocessor subsystem processes CDL files to generate the C source code files. The C source code files may be compiled by a standard C compiler to generate executable files.

CDL Error File (ERR)

The ERR ASCII files are produced by the CDL Preprocessor subsystem to store information about errors detected in the CDL files.

Markov Model Data

From-To-Frequencies (FTF)

The FTF ASCII file may be created by the user or generated by the Translate MMN option provided in the Usage Model Editor subsystem. The FTF file is used as an input to the Open option in the Usage Model Editor subsystem where it is converted to a From-To-Probability file. The file contains the Markov Usage Model in terms of arc and relative frequency. The first line in the file contains the number of states used in the file. The remaining lines contain a From-State ID, To-State ID, and the frequency count at which the state transition occurs.

From-To-Probabilities (FTP)

The FTP ASCII file is generated by the Usage Model Editor subsystem. The FTP file is used as input to both the Test Case Generator and Testing Markov Chain subsystems. The FTP file contains the Markov Usage Model in terms of arcs and probability: the first line in the file contains the number of states used in the file. The remaining lines contain a From-State ID, To-State ID, and the probability associated with the state transition.

Markov Usage Model Errors (SER)

The SER ASCII file is generated by the Usage Model Editor subsystem during validation of the FTF file. The SER file contains the errors found. The file contains an error description followed by the list of State IDs in error.

Usage Model Editor Analytical Results (FRE)

The FRE ASCII file is generated by the Usage Model Editor subsystem. The file is created and stored in a report format with the project name, source entropy, expected test case length, and expected minimum and maximum tests in the report header. Titled headings are included in the file. There is one line for each State ID identified in the FTF file. Each line of the report file contains a State ID, expected occurrence, actual occurrence, and expected number of sequences.

Symbols (SYM)

The SYM ASCII file may be user defined or generated by the Translate MMN option in the Usage Model Editor subsystem. The file contains a record for each State ID identified in the FTF file. If the SYM file is user defined, each State ID has an associated state label. If the file is generated by the Usage Model Editor subsystem, each line has an additional State ID number field.

Markov Model Notation (MMN)

The MMN ASCII file is user defined. This file uses a special notation to define a Markov Model. The MMN file uses a system of levels to lessen the burden of repeating reoccurring state transitions. The user can create the MMN file instead of the FTF file. The Translate MMN option in the Usage Model Editor subsystem converts the MMN file to an FTF file.

Undefined Symbol Errors (UND)

The UND ASCII file is generated by the Translate MMN option in the Usage Model Editor subsystem. This file contains any errors that occur while converting the MMN file to an

FTF file. Symbols used to identify states in the MMN file that are not defined within the file are listed in the UND file.

Certification Model Report (CMR)

The CMR ASCII file is generated by the Certification Model subsystem. The file is created and stored in report format. The first three lines in the file contain title headings and an underline. The remaining lines contain an independent variable (version number or engineering change), observed MTTF, predicted reliability, predicted MTTF, base MTTF, improvement factor, and correction factor.

If the Time to Failure (TTF) file is the Certification Model subsystem input, the MTTF is calculated by averaging the time to failures (relative time) for the same independent variable. (Relative time for each failure is manually calculated for the TTF file. Several failures may occur within one version of software or Engineering Change.) If the DB file is used as input (instead of the TTF file), the MTTF is calculated by subtracting the failure time (actual time) of the last failure within the independent variable from the last failure for the next independent variable. Only one line is reported for a single independent variable.

Testing Markov Chain (TMC)

The TMC ASCII file is created by the Testing Markov Chain subsystem. Each line of the file contains From-State ID, To-State ID, and frequency representing an arc of the Markov chain. This file is similar to the FTF file used in the Usage Model Editor subsystem.

Testing Matrix Estimation Report Data (EST)

The EST ASCII file is generated from the Testing Markov Chain subsystem. This file is in report format. The first line of the file identifies the report and the project. Following a space line, the next three lines contain title headings and an underline. The remaining lines contain a test case number, failure number, estimated reliability, estimated MTBF, and the estimated discrimination (criteria for termination of testing).

Test Case Sequences (SEQ)

The SEQ ASCII text file is generated by the Test Case Generator subsystem. Using Markov Model data, test cases are randomly generated and stored in the file. Each test case consists of various numbers of States indicating the action to be performed. The SEQ file is to be used by testers to certify software. The first line of the file indicates the number of test cases in the file and identifies the project. The remaining lines contain a test case number followed by a series of actions. The actions are indicated by numbered entries of States and their associated state label (called state transitions). Each test case number contains a series of underscores where the tester can indicate whether the test passed or failed.

Failure Data

Testing Failure Data (TST)

The TST ASCII file is user defined. This data is used as input to the Testing Markov Chain subsystem. The file contains a line for every recorded failure. Each line has a failure number (numbered consecutively), test case number, transition number, and an indication of whether the failure allowed testing to continue or not.

Time to Failure Data (TTF)

The TTF ASCII file is user defined. The Certification Model subsystem accepts either a TTF file or DB file as input data. The TTF file contains failure data in times-to-failure (relative time). Relative time is calculated by subtracting the test case number of the previous failure from the test case number of the current failure. Each line in the file contains an independent variable. The independent variable usually represents the version number or the engineering change number. Multiple lines for the same version number or engineering change number may be listed. Multiple lines of the same independent variable represent multiple failures within the version or engineering change.

Query Manager Database (DB)

The DB ASCII file is the only ASCII file generated by the Database subsystem. In the Database subsystem, the user is given the choice of reporting failures by version number or engineering changes (choice of policy). Information from the logical tables in Query Manager is used to create a report based on the policy chosen by the user. The user can choose to create a DB file from the report. The DB file may be used as input to the Certification Model subsystem instead of the user defined TTF file. The DB file contains an independent variable (version number or engineering change number, according to the policy chosen) and the associated failure time for each failure within the version number or engineering change. Failure time is recorded as the actual time-to-failure, which is the test case number of the failed test case. If more than one failure occurred within an independent variable (version number or engineering change), the same independent variable is listed for each failure.

Query Manager Logical Tables

Query Manager uses a CDB.QPF file that is installed with CRE. The CDB.QPF is a predefined set of logical tables, screens, and procedures prepared by the Cleanroom team using Query Manager. The CDB.QPF file is located in the CDB directory from the root directory. The CDB.QPF file was created by building a Query Manager Procedure

which dynamically determines the flow of execution within the procedure. The CDB.QPF file is a Query Manager object. The Database subsystem calls the CDB.QPF procedure when initiating the Query Manager interface.

When Query Manager creates a new database, it creates a separate directory to store all the associated database objects for reference and update. The first directory for the first database created is named \SQL00001. Directories for databases subsequently created are named \SQL00002, \SQL00003, and so on (Database Manager Programming Guide and Reference p. 1-10). The directories are stored in the root directory. CRE does not access the data in these databases except through the Query Manager facility.

Project Logical Table

The user input for the Project table contains the Project Name, Comments, and Description.

Version Logical Table

The user input for the Version table contains the Project Name, Comments, and Description.

Failure Logical Table

The user input for the Failure table contains the following required fields: Failure Number, Test Number, Test ID, Version Number, Test Date, Name of Tester, and Project Name. Input fields are available for the following non-required data: Test Time, Test Site, and Comments.

Engineering Changes Logical Table

The user input for the Engineering Change table contains the following required fields: Engineering Change Number, Failure Number, Designer, Version Number, and Project Name. Input fields are available for the following non-required data: Severity, EC Classification, Verification Date, Verifiers, Previous EC, and Comments.

CHAPTER 3

CURRENT SYSTEM PROCESSING AND DATA FLOW

CRE Overview

This chapter documents the current data flow, as well as the processing control flow. Figure 3.1 depicts the current CRE data flow. All files have a primary name and a name extension. ASCII file names are displayed as "*.file extension" (the asterisk is used in place of the file name given by the user). The file extension defines the type of file. All files created by CRE use the same user defined primary name of the input file to the subsystem.

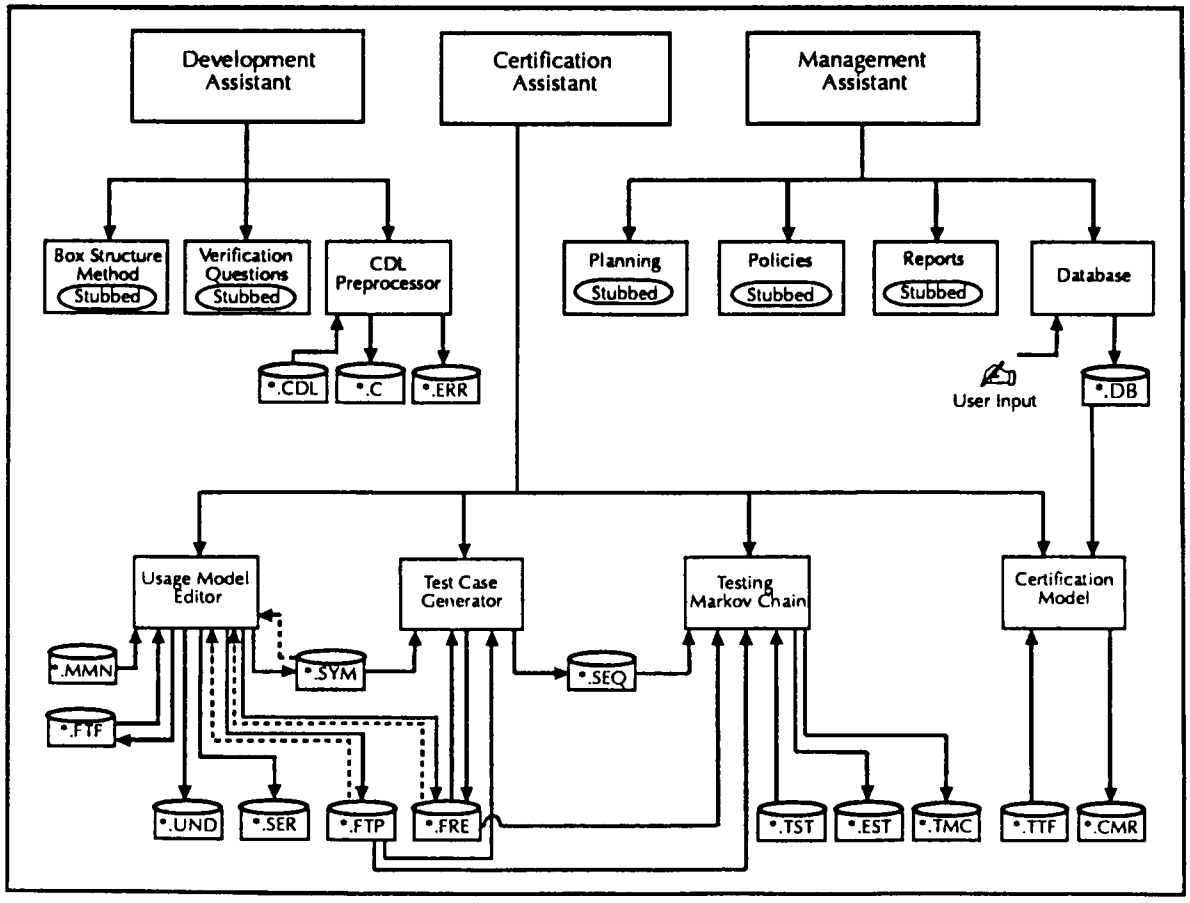


Figure 3.1 Current CRE Data Flow

Cleanroom Environment

CRE (shown in Figure 3.2) consists of four control windows: 1) Cleanroom Environment, 2) Development Assistant, 3) Certification Assistance, and 4) Management Assistant. There are eleven subsystems that are available as functions of these control windows.

The control windows provide a title bar, menu bar, system menu, maximize and minimize icons, and a client area where an icon with the appropriate text is displayed for each of the eleven subsystems. The icon may be displayed without the text through use of the **View** pull-down menu, which allows the user to select the Text On or Text Off options. The control windows may be moved, minimized, maximized, and resized. The **Help** pull-down menu has not been implemented. The **File** pull-down menu displays the **Open** and **Exit** options. The user may exit the window by selecting exit. The user may select one of the subsystems by using the **Open** option if a subsystem has been made current. A subsystem can be made current by clicking once (two clicks open the subsystem). If a subsystem has already been opened, and the user has returned to the Cleanroom Environment window, that subsystem remains current until changed.

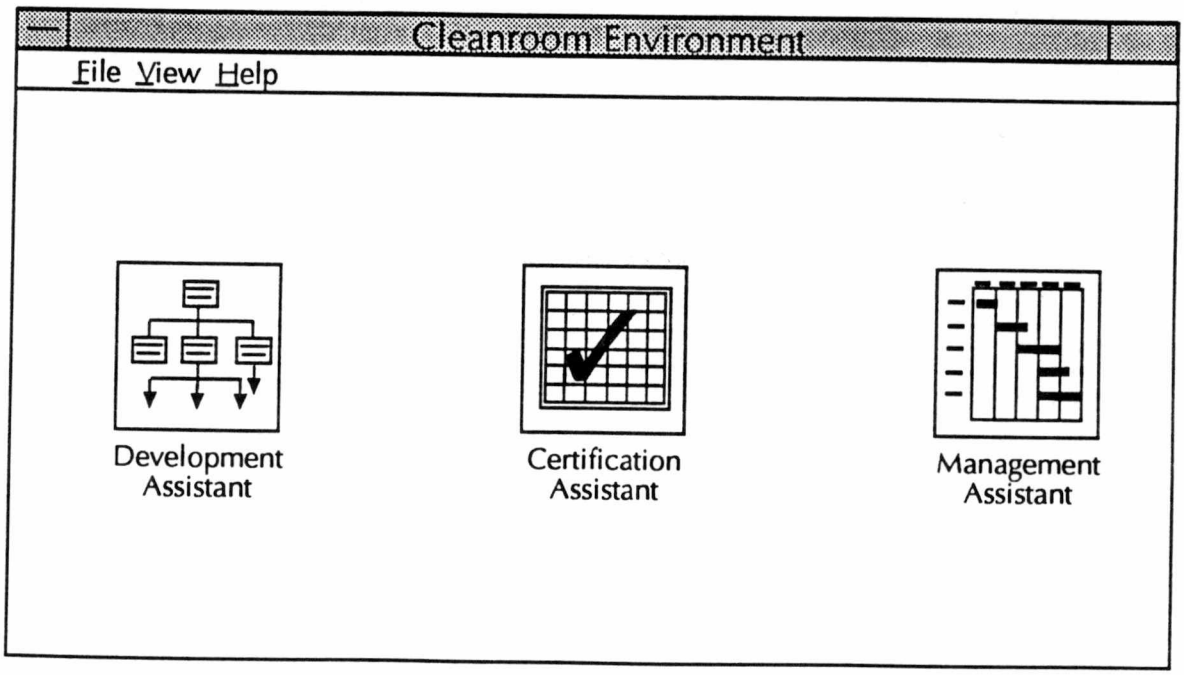


Figure 3.2 Cleanroom Environment Window

Development Assistant

The Development Assistant subsystem provides tools to be used by a Cleanroom development team during the design phase of a project increment. The Box Structure Method, Verification Questions, and CDL Preprocessor subsystems are included in this subsystem. The Development Assistant window is shown in Figure 3.3.

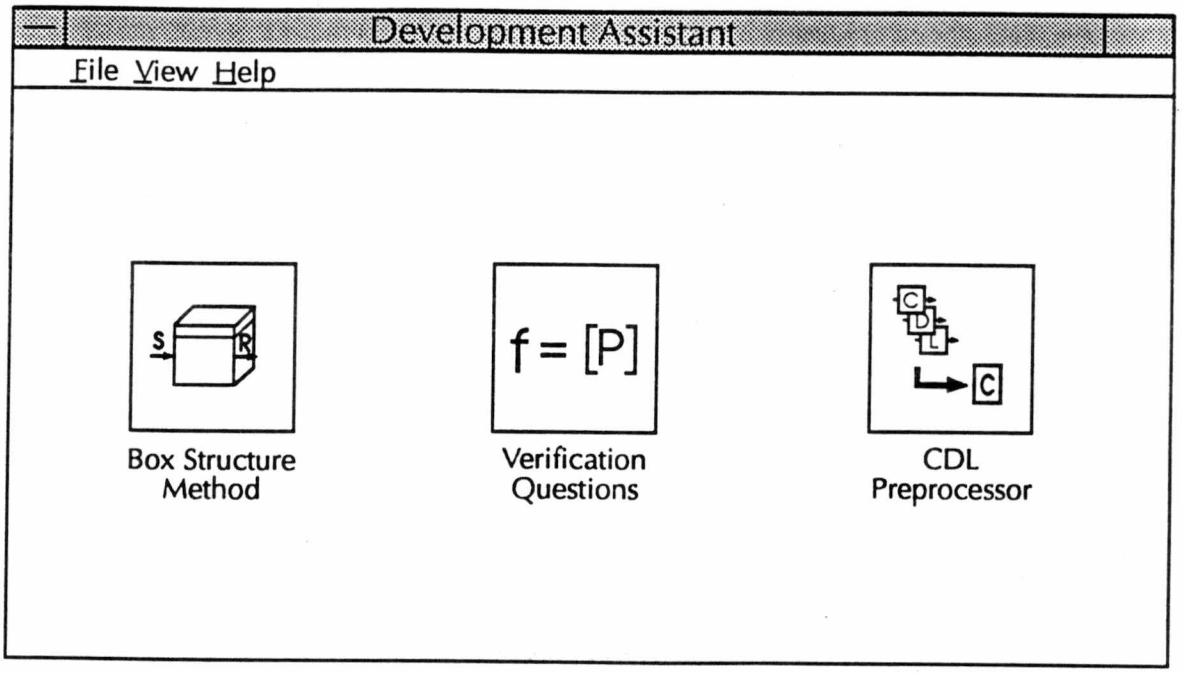


Figure 3.3 Development Assistant

Box Structure Method

The Cleanroom Software Development Method uses the box structure analysis and design process. The CRE development tool will provide box structure aids to the user. The Box Structure Method (BSM) subsystem is in the early stages of implementation.

The data to be used in the BSM subsystem include the black box, state box, and clear box structures. A black box is an expression of specifications in the forms of stimulus histories, responses, and transition rules without references to state data or procedures. A state box is created from the black box and explicates state data requirements. A state box expresses its transition rule as a mapping of the current stimulus and state to a new state and response without referencing procedures. A clear box can be designed from either a black box or a state box to express the procedure for implementation of the associated black box.

Verification Questions

The Cleanroom Software Development Method incorporates the use of functional verification. The development team examines all areas of design in sufficient detail to be convinced of the system's correctness. The verification process includes the examination of code and design process. The box structure method develops box structures from a top-down approach: black box -> state box -> clear box. After designing the clear box, the verification process is performed by reversing the order and deriving the black box from the state box, which is derived from the clear box. Boxes at any level can be derived from the lower level boxes if the design is correct. The Verification Questions (VFQ) subsystem has not been implemented, but will provide a tool to support box structure verifications.

The VFQ subsystem will incorporate a checklist of target language features to ensure correct syntactical expressions. Data will be provided which will detail the verification process. If problems occur later in the software development or testing, this data may be helpful in determining the point of faulty design or process execution.

If the user selects the VFQ subsystem, a dialogue box is displayed informing the user that implementation is pending. After implementation, the subsystem will provide options allowing updating and verification.

CDL Preprocessor

The CDL Preprocessor (CDLP) subsystem uses CDL files as input to create C source code. After selecting the CDLP subsystem, the CDL Preprocessor window is displayed (shown in Figure 3.4).

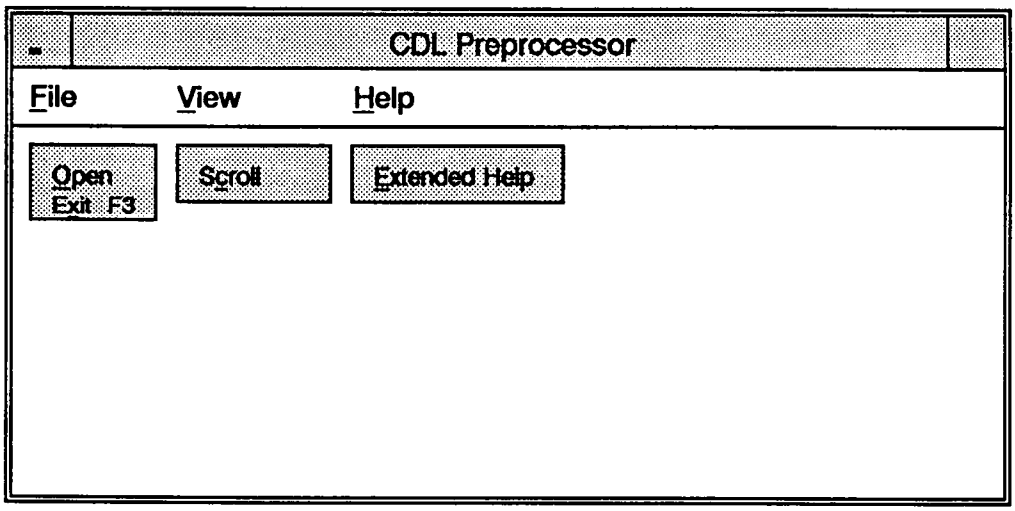


Figure 3.4 CDL Preprocessor Window With Pull-Down Menus

The box structure method incorporates the use of the CDL as a means to create black boxes, state boxes, and clear boxes. CDL files are manually created outside of CRE. The CDLP subsystem removes the black box and state box design and converts the clear box into C source code. The source code is stored into a C file with the same primary name as the CDL file. The resulting C files may be compiled by a standard C compiler to create an executable file.

The CDL Preprocessor window provides three pull-down menus: **File**, **View**, and **Help**. The **File** menu is used to select a CDL file. The **View** and **Help** menus have not been implemented.

The **Open** option presents the window shown in Figure 3.5. The user must select a directory from the left portion of the window, which presents the CDL files available in that directory on the right portion of the window. When the user selects a CDL file, the selected file is positioned in **File Name**. The user may select the file by pointing and clicking with the mouse, or by entering the file name. When the CDL file is selected, the CDLP subsystem processes the file. A message indicating that the file is being processed is displayed. After the CDL file has been processed, the number of errors found is displayed. Errors are stored in the ERR file.

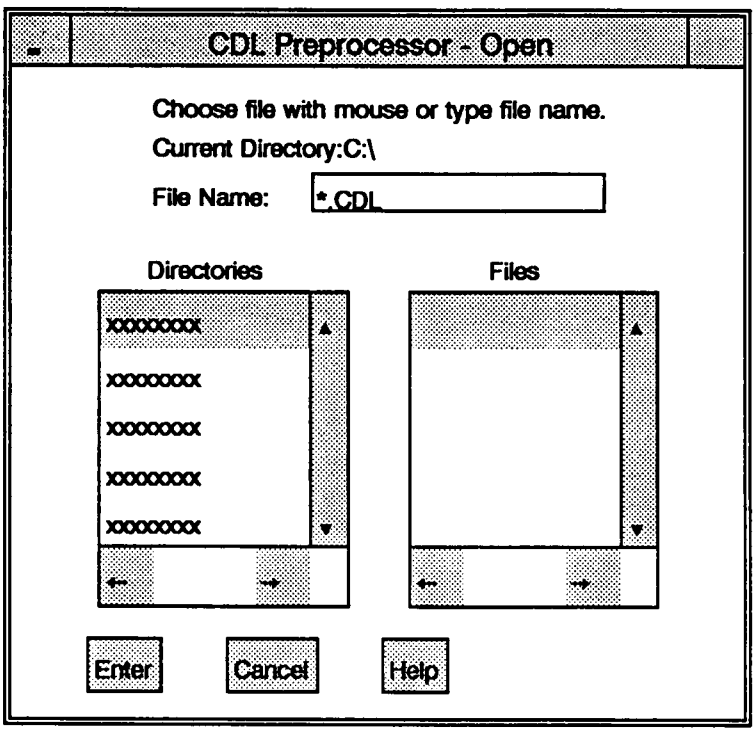


Figure 3.5 CDL Preprocessor - Open Window

If more than one CDL file is to be converted to a C file, each file must be opened through the **Open** menu. The CDLP subsystem main window is redisplayed after each CDL file is processed.

CDLP Subsystem

Input File	Output Files
*.CDL	*.C
	*.ERR

Certification Assistant

The Certification Assistant subsystem provides tools for testing and certification of software developed in the Cleanroom project. This subsystem is the most mathematically intensive portion of CRE. The Certification Assistant subsystem allows the user to benefit from the certification model with minimum effort. The Usage Model Editor, Test Case Generator, Testing Markov Chain, and Certification Model subsystems are accessed through this interface. The Certification Assistant window is shown in Figure 3.6.

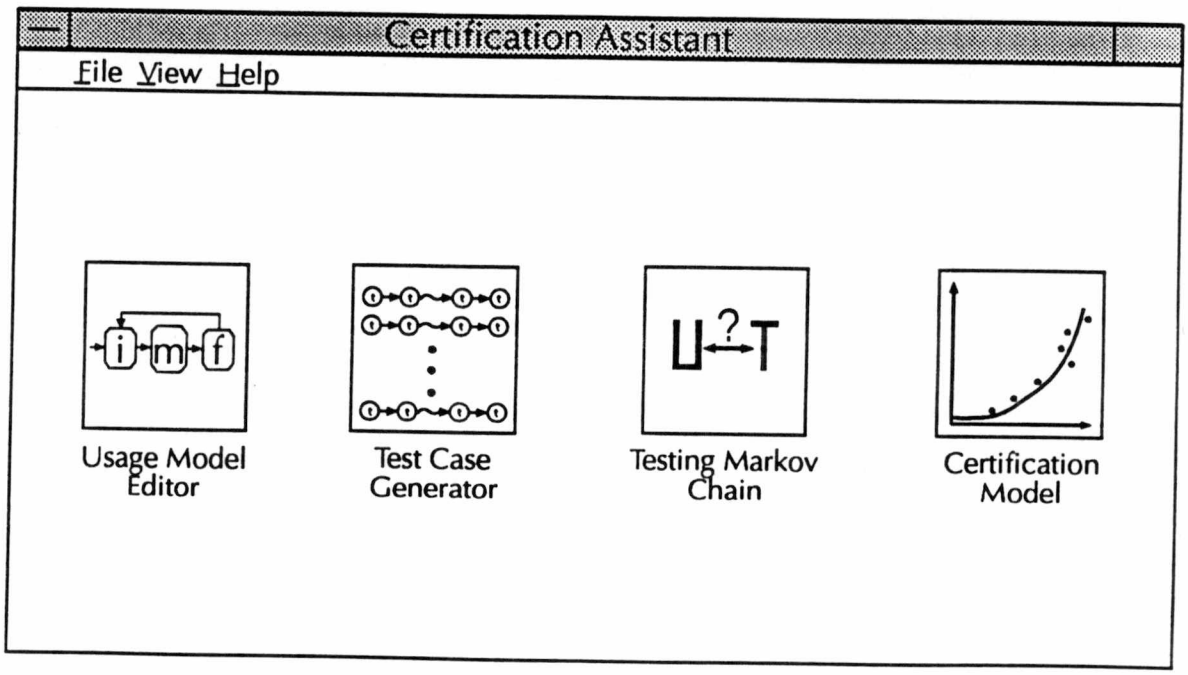


Figure 3.6 Certification Assistant

Usage Model Editor

The main function of the Usage Model Editor (UME) subsystem is to generate the FTP file that is used in both the Test Case Generator and the Testing Markov Chain subsystems. The user has two choices: create the FTF or MMN file manually before entry into CRE, or the MMN file may be used to generate the FTF file and the SYM file. The SYM file is used as input to the Test Case Generator subsystem and must be provided either through the MMN file or manually.

After selecting the UME subsystem, the UME subsystem window is displayed (shown in Figure 3.7) offering three pull-down menus: **File**, **Action**, and **Help**. The **File** menu must be used to select an FTF file before the **Action** menu may be used. The **Help** pull-down menu has not been implemented.

The **Translate MMN** and the **Open** options display similar windows allowing the selection of the directory and either an MMN or FTF file. The user must select a directory from the left portion of the window presenting the appropriate MMN or FTF files available in that directory on the right portion of the window. When the user selects an MMN or FTF file, the UME subsystem attempts to process the file. If no errors were detected, the UME subsystem main window is again displayed. If an error occurs, an error message is displayed.

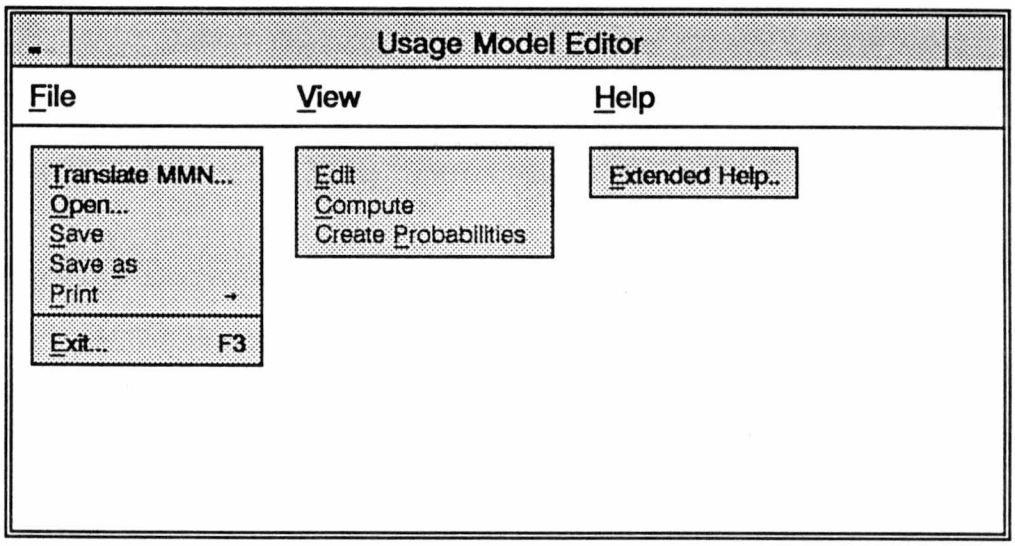


Figure 3.7 Usage Model Editor Window With Pull-Down Menus

Processing an MMN file from the **Translate MMN** option creates or replaces the FTF file and an SYM file. If an FTF file already exists with the same primary name as the input MMN file, the option of canceling the operation or overwriting the current FTF file is presented. If errors occur while generating the SYM file, a list of undefined symbols found in the MMN file is stored in the UND file.

If the user has created an FTF file separately or through the use of the **Translate MMN** option, the **Open** option is used to select the FTF file. The **Open** menu opens and validates the FTF file. If errors are found in the FTF file, an SER file is created or updated with the errors.

Option	Input File	Output Files
Translate MMN	*.MMN	*.FTF *.SYM *.UND
Open	*.FTF	*.SER

Once an FTF file has been opened, the user is able to perform **Edit, Compute,** or **Create Probabilities** actions through the **Action** menu. The **Compute** option converts the FTF file into an FRE file. If an FRE file already exists with the same primary name as the input FTF file, the option of canceling the operation or overwriting the current FRE file is presented. The **Create Probabilities** option uses the FTF file to compute the probabilities and create the FTP file. If an FTP file already exists with the same primary name as the input FTF file, a dialogue box displays the option of canceling the operation or overwriting the current FTP file. The **Edit** option has not been implemented.

Option	Input File	Output File
Compute	*.FTF	*.FRE
Create Probabilities	*.FTF	*.FTP

Once an FTF file has been opened, the user may again select the **File** menu where the **Save, Save as, Print,** and **Exit** options are available. The **Save** and **Save as** options have not been implemented. The **Print** option displays the following list of options: **Frequencies, Probabilities, Symbols**, and **Analytical Results**. The user may select any of the **Print** options to print the selected information. If the required file for each print option is not on the current directory (if they have not been created using the **Action** options), those print options are not highlighted and may not be chosen.

The following ASCII files are used to print the information:

Printed Information	ASCII File
Frequencies	*.FTF
Probabilities	*.FTP
Symbols	*.SYM
Analytical Results	*.FRE

Test Case Generator

The main function of the Test Case Generator (TCG) subsystem is to generate the SEQ from the FTP and SYM files using random generation. During the creation or appending of the SEQ, the FRE file is updated with the actual number of occurrences of each state. FRE and SEQ files with the same primary name as the input FTP file must exist in the same directory. The SEQ file is used by the testers of the software during certification and as input to the Testing Markov Chain subsystem. The FRE file is used as an input file to the Testing Markov Chain subsystem.

The TCG subsystem is the second window displayed after the selection of Certification Assistant. The TCG subsystem requires that the FTP file, which is generated in the UME subsystem, be available before it may be used.

After selecting the TCG subsystem, the TCG subsystem window is displayed (shown in Figure 3.8) offering three pull-down menus: **File**, **View**, and **Help**. The **File** menu is used to select an FTP file. The **View** and **Help** menus have not been implemented.

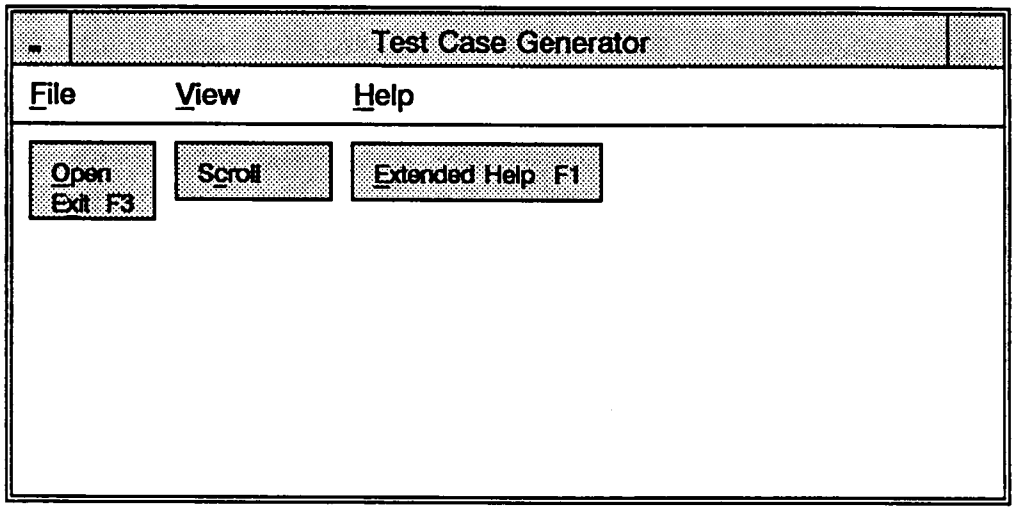


Figure 3.8 Test Case Generator Window With Pull-Down Menus

The **Open** option presents a window allowing selection of a directory and the FTP files located in the selected directory. When the user selects a directory, the FTP files available in that directory are displayed for selection. The user may select the file by pointing and clicking with the mouse or by entering the file name. When the FTP file is selected, the user is prompted with a dialogue box to enter the number of test cases to be generated.

If an SEQ file already exists, the user is prompted with the dialogue box: "Overwrite (Yes, No, Cancel)". The user may select **Yes** to create a new SEQ file. If the user selects **NO**, the indicated number of test cases will be generated and appended to the end of the

existing SEQ file. The **Cancel** option terminates the generation of the test cases.

TCG Subsystem

Input File	Output Files	Accessed Files
*.FTP	*.SEQ	*.SYM
	*.FRE	*.FRE

Testing Markov Chain

The Testing Markov Chain subsystem analyzes data from testing. The Testing Markov Chain subsystem uses data from the UME and TCG subsystems to generate Reliability (R), MTBF, and Discrimination (D) measures (criteria for termination of testing). Analysis of testing incorporates the FTP file from the UME subsystem, FRE file (created in the UME subsystem and updated in the TCG subsystem), SEQ file (created in TCG subsystem), and the TST file (manually generated by the user). All files must have the same primary name and be in the same directory. The Testing Markov Chain subsystem tracks the testing process by analyzing the test cases generated and the observed failures to create the TMC file (current testing Markov chain). In addition, the Testing Markov Chain subsystem creates the EST file that is a text estimation report about the testing matrix.

After selecting the Testing Markov Chain subsystem, the Testing Markov Chain subsystem window is displayed (shown in Figure 3.9) offering three pull-down menus: **File**, **Actions**, and **Help**. The **File** menu must be used to select an FTP file before the **Action** menu may be used. The **Help** pull-down menu has two options: **Extended Help** and **About**. The **Extended Help** option has not been implemented. The **About** option provides a dialogue box displaying text about where CRE is being developed.

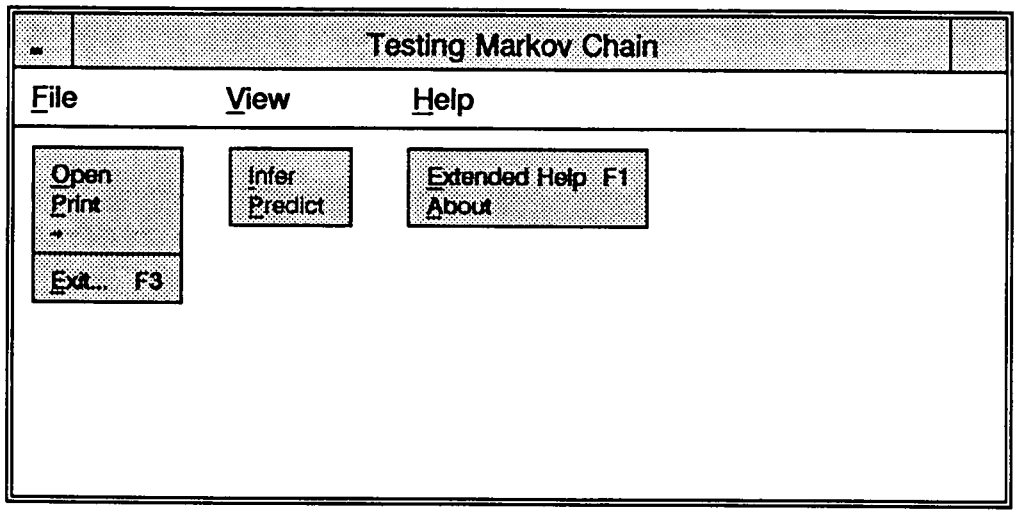


Figure 3.9 Testing Markov Chain Window With Pull-Down Menus

The **Open** option is used to select the FTP file. A window is presented allowing selection of the directory and the FTP files located in the selected directory. The user must select an FTP file in the manner utilized in the TCG subsystem. An error is reported if the associated FRE file is not present on the same directory as the FTP file.

The **Print** option presents the two selections: **Estimation report** and **Prediction report**. The **Estimation report** option is active only if the associated EST file is in the current directory. The intent is that the user selects the FTP file and performs an action before returning to the printing capabilities. However, the estimation report may be printed anytime the EST file is available. Currently, the **Prediction report** option has not been implemented. A PRE file will contain the prediction data.

The **Exit** option is used to return to the Certification Assistant window.

After the FTP file has been selected, the **ACTION** menu provides the **Infer** and **Predict** options. The **Infer** option requires that the FTP, SEQ, and FRE files be available on the directory with the same primary name. In addition, the TST file, which contains the testing data must be available. If the TST is not available, zero failures are assumed and processing continues without error or warning. An inference dialogue box (shown in Figure 3.10) is presented after selection of the **Infer** option. The **Predict** option has not been implemented.

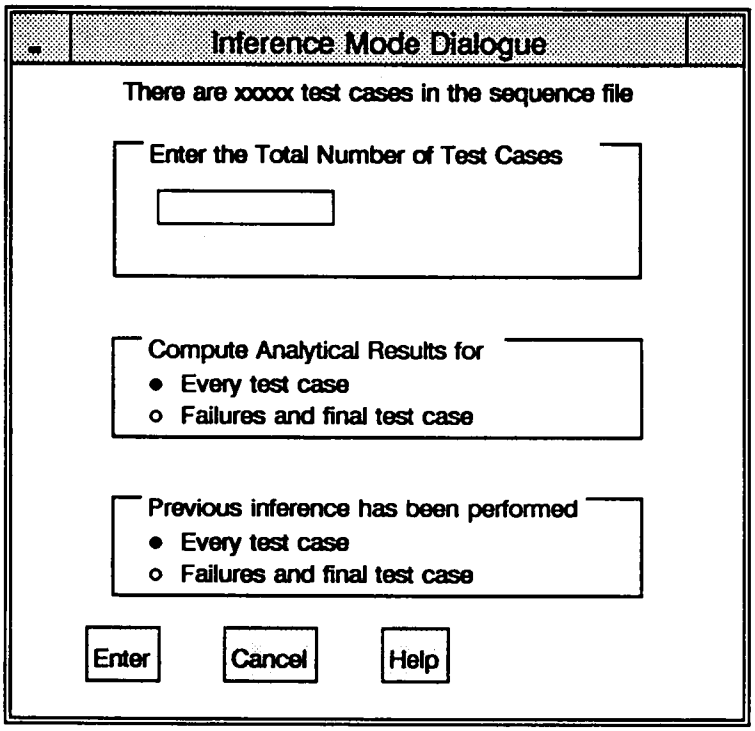


Figure 3.10 Inference Mode Dialogue

The user must enter the number of inferences to be made. The number entered must be at least one and no more than the number of test cases in the SEQ file displayed. The selected method of computing the analytical results defaults to every test case; however, the user may select that the results be computed for only failures and the final test case. The default selection of storing the TMC file is to append or continue the inferences. The user may change the default and restart the inference process to recreate the TMC file with the new inferences.

Testing Markov Chain Subsystem

Input File	Output Files	Accessed Files
*.FTP	*.EST	*.FRE
	*.TMC	*.SEQ
		*.TST

Certification Model

The Certification Model (CM) subsystem generates the CMR file and displays the certification model graphical representation. The TTF file, which contains relative time to failure, or the DB file, which contains the actual time to failure, must be used as input to this subsystem. The TTF file is manually created outside of CRE. The DB file is created through the Database subsystem of the Manager Assistant.

After selecting the Certification Model subsystem, the Certification Reliability Model window is displayed (shown in Figure 3.11) which presents three pull-down menus: **File**, **View**, and **Help**. The **File** menu must be used to select the TTF or DB file. The **View** menu allows the user to select the certification graph or the certification report to be displayed after processing the input file. The **Help** pull-down menu has two options: **Extended Help** and **About**. The **Extended Help** option has not been implemented. The **About** option provides a dialogue box displaying text about where CRE is being developed.

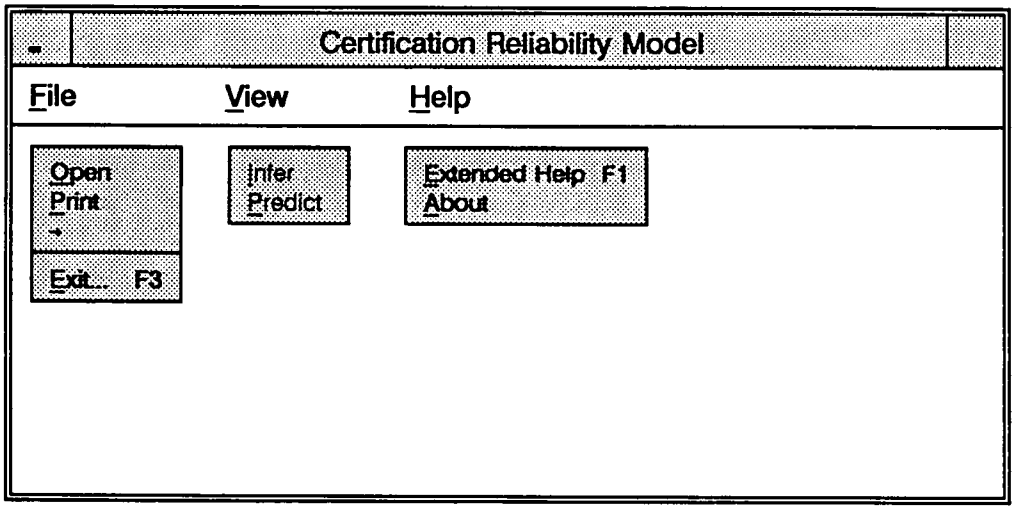


Figure 3.11 Certification Reliability Model Window With Pull-Down Menus

The **Open** option is used to select the FTP file. A window is presented allowing selection of the directory and the TTF files located in the selected directory. The user must select a directory from the left portion of the window which presents the TTF files available in that directory on the right portion of the window. The user may select the file by pointing and clicking with the mouse or by entering the file name. If the user wishes to use a DB file as input, the DB file name must be entered in the File Name position after selection of the appropriate directory. After a file selection has been made, the certification graph is displayed, and the certification model report is created and stored in the CMR file.

The **Print** option may be selected to automatically print the certification report from the CMR file. This option cannot be selected until the user selects an input file. Other subsystems allow the use of the print option if the file to be printed is available; however, even if the CMR file is present, the Certification Model does not allow printing of the certification report until after processing.

Selection of the **View** menu presents the options **Graph** and **Report**. Currently, only the **Graph** option is available. The **Report** option is displayed but may not be selected.

Certification Model Subsystem

Input File	Output File
*.TTF	*.CMR
or	
*.DB	

Management Assistant

The Management Assistant interface of CRE is intended to be the primary method of user input and output of data. Subsystems Planning, Policies, Reports, and Database are included in this interface. The Management Assistant window is shown in Figure 3.12.

Planning

Software managers have many planning needs including schedules, milestones, optimizing the use of common software, verification sessions, etc. The Planning (PLN) subsystem has not been implemented. A study of the best planning tools and data to be used will be performed before design of this subsystem begins. If the user selects the PLN subsystem, a dialogue box is displayed indicating that implementation is pending.

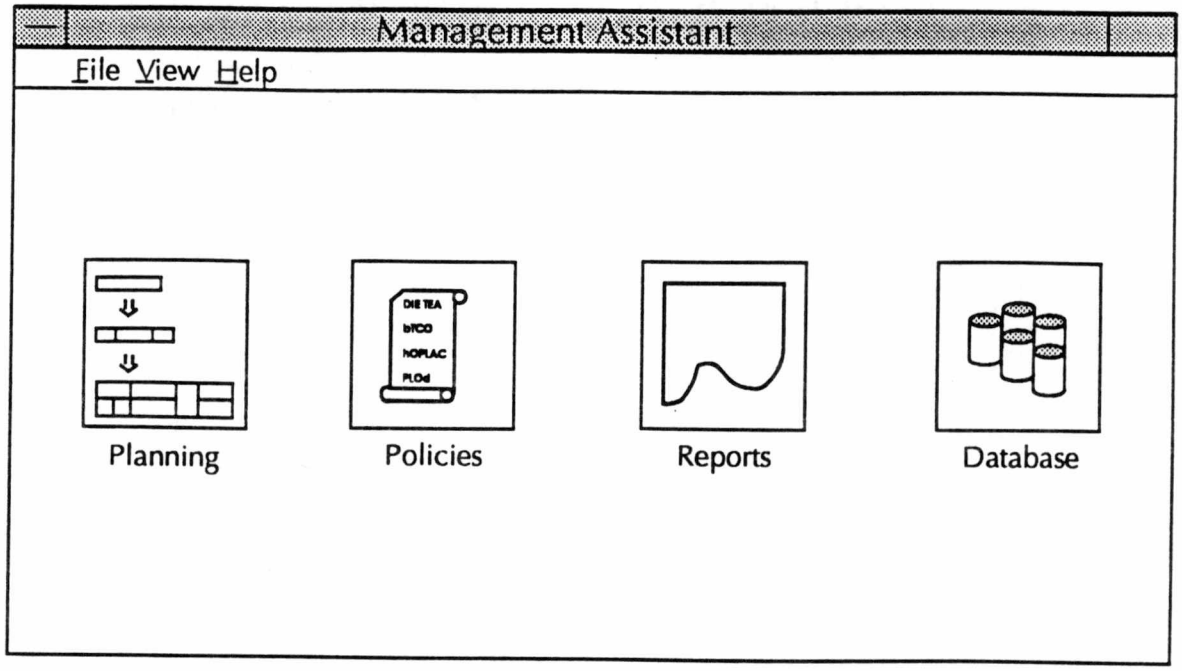


Figure 3.12 Management Assistant

Policies

Policy refers to the method used to interpret failure reports for use by reliability models. Every failure observed should be entered into the failure database. Different developers of software projects may require various methods of interpreting failures. All information from development and certification may be interpreted by mathematical imperative, corporate policy, and managerial prerogative. Policies affect the interpretation of failures as reported and result in differing statistical certifications. CRE provides two distinct methods of interpreting failures; however, users may have further needs requiring additional user definitions. CRE offers the Policies subsystem as a method of defining policies so that the certification process can interpret the failures and report appropriately.

Users should record all failures and all engineering changes. Several failures may be recorded before the testers determine that testing must stop and an engineering change must be made. As an example, a typographical error in a heading is recorded as a failure, but testing may continue even beyond a system failure. CRE allows the user to apply a policy and choose either all failures or only engineering changes to be given to the certification model. This function is provided by the Database subsystem when creating the DB file that is an input to the Certification Model subsystem. The Policies subsystem is an extension of that capability that will allow more diverse policy rules to be created by the user.

The Policies subsystem has not been implemented. If the user selects the Policies subsystem, a dialogue box is displayed indicating that implementation is pending.

Reports

CRE includes many data items that should be reported or made available. Managers of software projects will find it helpful to be able to generate screen or printed reports from one central subsystem. The Reports subsystem will provide the user a wide variety of reports including a certification report, reliability information, and errors removed in verification, compilation, and testing. Information reported will include system generated and user supplied data.

The Reports subsystem has not been implemented. If the user selects the Reports subsystem, a dialogue box is displayed indicating that implementation is pending.

Database

Query Manager

The Database (DBS) subsystem allows the user to enter project, testing failures, and engineering changes data into Database Manager logical tables. The OS/2 Query Manager menu facility is used as the interactive tool.

Upon selecting the DBS subsystem, processing control is passed to Query Manager and a dialogue box indicates that Query Manager is starting. If the user is not already logged onto Query Manager, the user is prompted with a logon window which requires entry of identification and a password.

After the Query Manager logon window, CRE initiates the CDB_V1 database (Cleanroom Database version 1) which is the database name saved in the CDB.QPF procedure. A message is displayed in a dialogue box indicating that the CDB_V1 database is being loaded. Processing control is passed to Query Manager, and the Cleanroom software remains in wait until Query Manager is exited or another window of CRE is made current by the user. If the logon is canceled by the user initiation of the database is still attempted, and the SQL1093N system error is displayed indicating that the user is not logged on.

The Query Manager windows have the same functionality as the Cleanroom control windows. The windows may be moved, maximized, minimized, and resized. The minimized icon for Query Manager is supplied by the OS/2 Database Manager software.

If the logon is successful, the Query Manager displays the Certification Database Main Menu. The Certification Database Main Menu presents the **Exit** on the menu bar. Two options allowing the user to add and select projects are in the client area of the window.

When the user selects the **Add a new project** option, a window is presented that allows the user to enter the project name and comments. This window presents an **Action** and **Exit** menu. The **Action** menu contains the options: Add Project, Blank Panel, Show Field (which cannot be used), and Print.

When the user selects the **Select existing project** option, a dialogue box is displayed allowing the user to enter the project name to be selected.

After a project has been selected or added, the Certification Database Actions window (shown in Figure 3.13) is presented. This menu allows the user to add or change data or execute a policy.

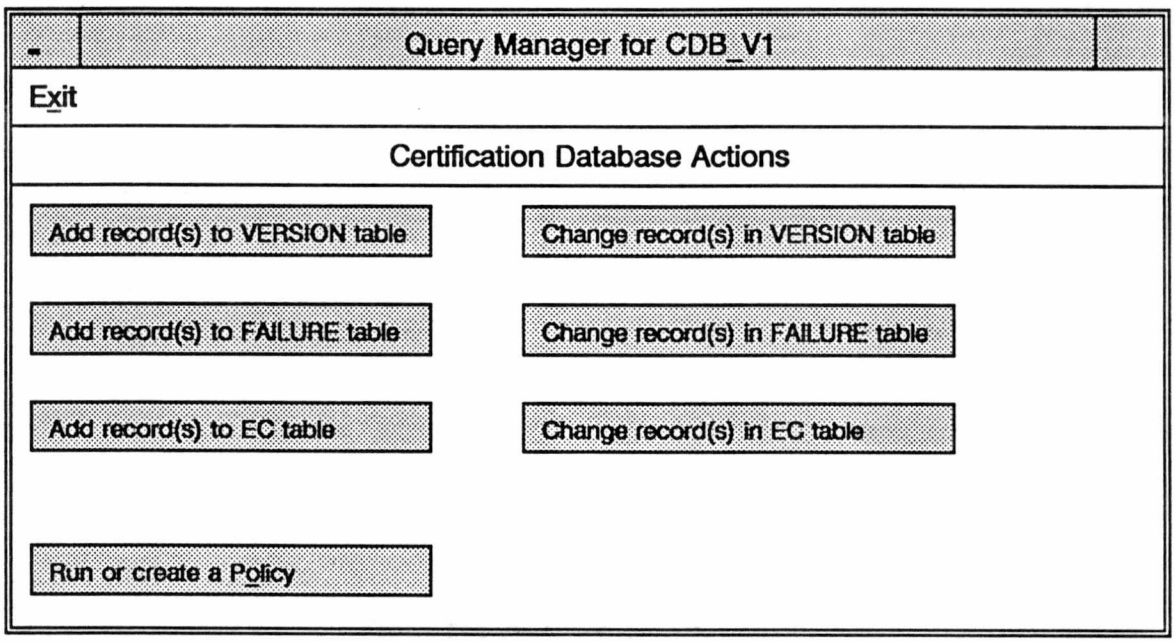


Figure 3.13 Certification Database Actions Window

Logical Tables

The **Add record(s) to the VERSION table** option presents the Add Version(s) window (shown in Figure 3.14). The user may select an option by positioning the mouse pointer on the option and clicking twice.

The user input fields that are displayed in bold text are required. The Comments field is not required. The selected Project Name does not default to the currently selected project. The user may enter any existing Project Name in the field. If the entered Project Name does not match an existing Project Name, an error message is displayed. The Project Name field is case sensitive and must exactly match an existing Project Name.

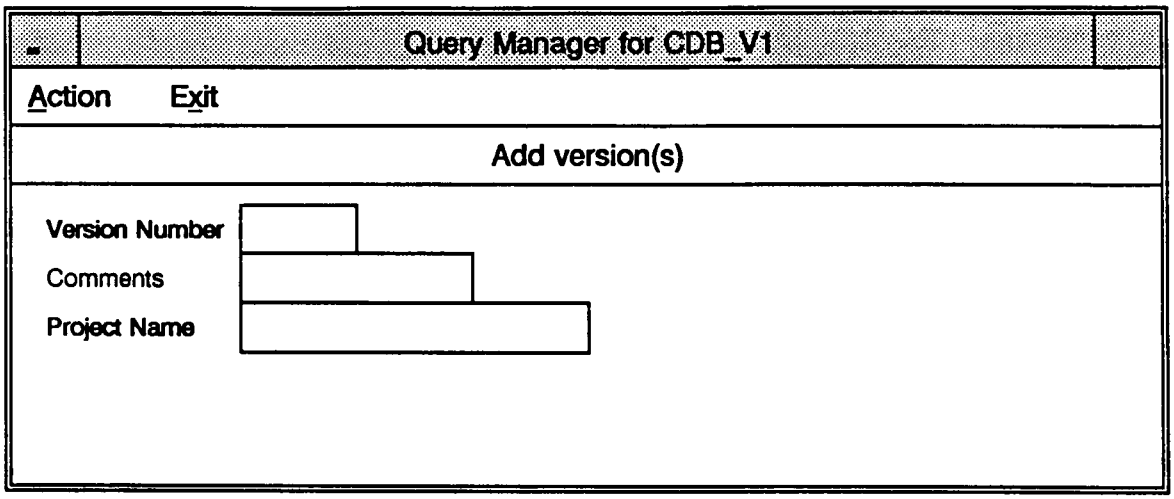


Figure 3.14 Add Version(s) Window

The **Add record(s) to the FAILURE table** and **Add record(s) to the EC table** options present the Add Failures(s) window (shown in Figure 3.15) and the Add Engineering Change(s) window (shown in Figure 3.16).

The user input fields that are displayed in bold text are required; other fields are optional. The selected Project Name does not default to the currently selected project. The user may enter any existing Project Name in the field.

The windows (or panels) allowing change to the logical tables are the same as their add counterpart, except the title bars for the change option display edit/browse instead of add. All update windows provide two pull-down menus and a help facility. The help facility lists generic help information provided by Query Manager. **Action** and **Exit** pull-down menus are the same for all three tables. The **Exit** pull-down menu allows the user to exit the current panel or resume processing. The **Action** menus display different options according to whether the function is an add or a change.

The **Action** menu in the add window provides the options: **Add and next, Add and Keep, Blank panel, Show field,** and **Print.** The **Action** menu in the change window provides the options: **Change and next, Delete and next, Next, Search, Show field,** and **Print.**

The **Show Field** can be used only when the cursor is on a field that is too long to show on one line. Only the Comments field may be used with this option. If this option is used, a separate window is displayed showing the entire comment.

If the user selects the **Search** option, a separate set of options is presented: **Perform search, Extended search, Show field, Blank panel,** and **End search.** If the user selects these search options, additional windows are displayed for user input of search criteria.

These database functions logically update a version table, failure table, and an engineering table. However, the actual information is stored in the CDB_V1 database, which is only accessible through OS/2 Database Services.

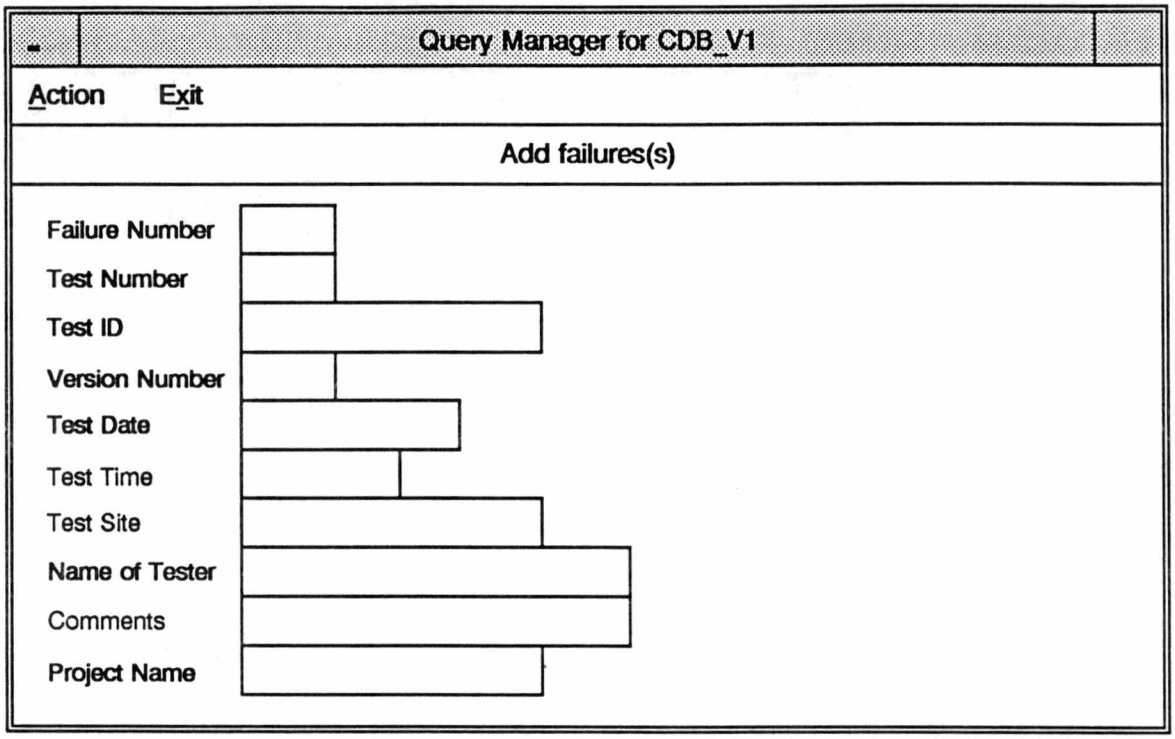


Figure 3.15 Add Failures(s) Window

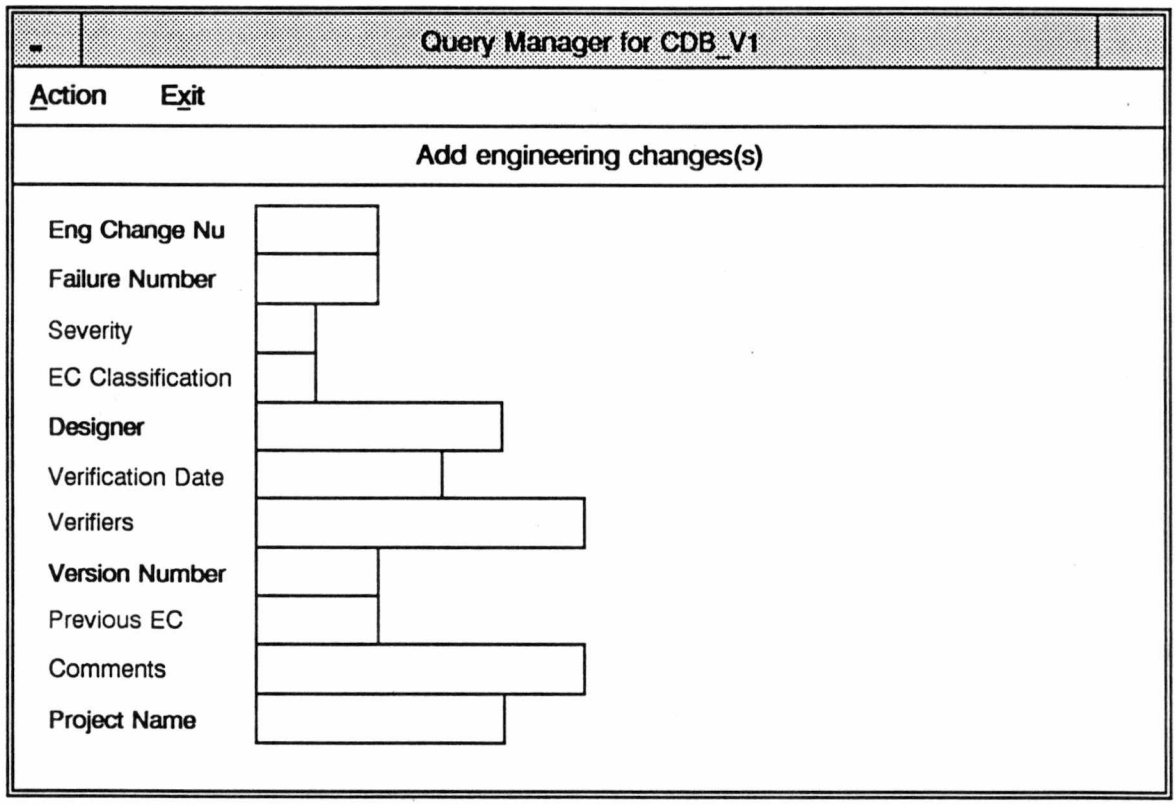


Figure 3.16 Add Engineering Change(s) Window

Policy Generation

The policy portion of the DBS subsystem is the only function that creates data used by other subsystems in CRE. After selecting the **Run or create a Policy** option, the Policy Menu window is displayed (shown is Figure 3.17). The first two options present the predefined reporting policies. The third option allows the user to create a user defined option.

The two predefined reporting policies use either all failures or just the engineering changes to report failures. The time-of-failure data is retrieved from the failure logical table for both policy selections. Either version numbers from the failure logical table or engineering changes from the engineering changes logical table are reported. The **Create a new Policy** option has not been implemented.

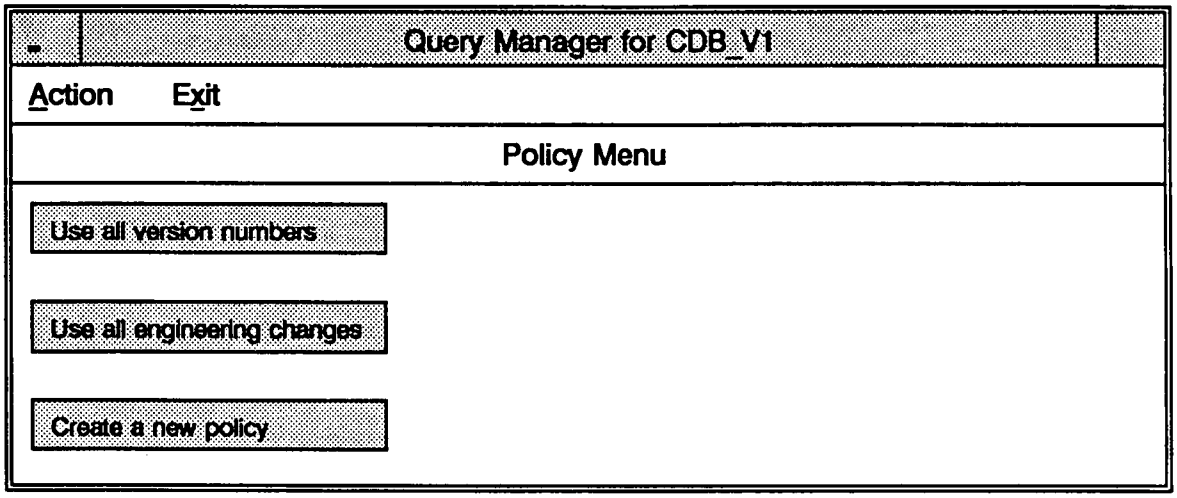


Figure 3.17 Policy Menu Window

When a policy has been chosen, a report is displayed providing either a list of version numbers and the associated failure times or a list of engineering changes and the associated failure times.

The report window lists three pull-down menus: **Action**, **Display**, and **Exit**. The Query Manager help facility is always available in the upper-right corner. The **Display** menu shows options for form and query, but may not be selected. The **Exit** menu allows the user to exit the report or to resume.

The **Action** menu displays three options: **Save data**, **Print**, and **Graph**. The **Save data** option allows the user to save the report in a Query Manager file. However, there is no access provided by this system to access reports that have been saved. The **Graph** option provides a graph of the report if the graphical interface to Query Manager has been installed correctly. The **Print** option allows the user to print the report to either a file or a printer. Saving the report to a file allows the user to create an input file to the

Certification Model subsystem. The file must be given the extension name of .DB in order to use it as input. The report, and subsequently the DB file, contain failure data in actual failure time. The alternate input to the Certification Model subsystem is the TTF file, which contains failure data in times-to-failure (relative time).

CHAPTER 4

RESEARCH OF DATABASE SOLUTION

Requirements

Environment

CRE is being developed using IBM's OS/2 (Version 1.3) operating system. IBM has recently released Version 2.0 of its OS/2 operating system. The laboratory team plans to migrate the system to OS/2, Version 2.0. Any database software to be used in CRE must have the ability to execute in either OS/2 environment.

The compatibility requirement that database software must be able to interface with the C language narrows the search. Some database software packages may be executed using a single C code command, but limits access to the data to the features offered in the database software. These database software packages have the advantage that the system developers do not have to write code to create user interface screens, but the flexibility of the system to gain access to data from other modules of code and to validate data input is limited. There are database software packages that allow C code as an Application Programming Interface (API). These database software packages may or may not have interactive interfaces. In some instances, additional software may be purchased that include an interactive interface.

IBM C/2 Version 1.1 is being used to develop CRE's executable C code. The C compiler used to develop the system's code plays a significant part in the requirements for database software. Normally, a database software package is purchased precompiled under a specific C compiler, with the buyer having a choice of several marketed C compilers. Often the database software package source code may also be purchased (which allows the developer to compile the database software using the same C compiler that is being used in-house). Some risk is involved with this method. Different C compilers support different features of the C language. The database software source code may use features of C that are not supported in the in-house C compiler. Currently, few database software packages advertise compatibility with IBM's C/2 or the new IBM C Set/2 compilers. Any database product chosen to be used in CRE must be able to be compiled with the IBM C/2 compiler. Compatibility with the IBM C Developer Set/2 compiler is highly recommended.

Marketing Considerations

CRE is being developed for internal use in the IBM software development laboratories. Anyone using the system software needs only a licensed copy of OS/2. The current database interface to CRE uses the Database Manager, a part of the OS/2 licensed software. If a database product other than the Database Manager is used with OS/2 Versions 1.3 or 2.0, the licensing agreement with that product must be considered. Most

database products have an unlimited runtime version of their software available, allowing developers to distribute copies of their systems (which include the executable database product). The runtime versions of database products do not include the development features of the database.

Performance

The primary development objectives for CRE are reliability, maintainability, and processing speed. Reliability and maintainability are primarily controlled by the quality of the code. The performance level of a system is influenced by the quality of the code, as well as the speed of access controlled by the database product. Most systems using a database structure suffer loss of speed during input and output functions. Because CRE may require some database tables to contain a large number of records, the speed with which C function calls to the database are executed should be considered when selecting a database product. The performance of the database functions will determine the speed of database oriented CRE modules.

The time required to initially connect to a database software package should also be considered. The time required to connect and logon to the OS/2 Database Manager is so long it detracts from the performance of CRE. In addition, the compulsory security measures are not required for CRE.

Solutions Researched

Many database oriented software packages were examined. Three key features directed the software search: 1) compatibility with C, 2) compatibility with IBM OS/2, and 3) ability to access and update data through an API. Most database software packages were easily disqualified. Few database packages are OS/2 compatible. The computing industry has not developed much software for the OS/2 environment.

OS/2 Database Manager

The current database interface uses the OS/2 Version 1.3 Query Manager facility provided with the OS/2 Database Manager. CRE is to be migrated to execute under IBM's OS/2 Version 2.0; therefore, Version 2.0 Database Manager was also considered.

The use of Query Manager alone has not provided the flexibility needed for CRE. A more flexible solution would require the use of Database Manager's Database Services facility. Query Manager, a menu application that provides menu access to Database Services only, may be eliminated as a candidate.

Database Services provides the various facilities needed to create and maintain a database through a Structured Query Language (SQL) programming interface. The SQL statements can be embedded in the C application programs. Even though IBM's OS/2

Version 2.0 Database Services has more capabilities than Version 1.3, the security measures installed for data access are still a negative factor.

The user must gain access to Database Manager by entering a valid userid and password. Access to any Database Manager data must be performed while the user is logged on and executing the Database Manager interface.

Database Manager's structure and operations are much like IBM's DB2 mainframe database. A database consists of a group of data sets or tables. The data security controls and access are more sophisticated than general PC databases. Access to the database object must be authorized. Tables in a database are created with a specific creator's authorization ID or another name for table grouping. A user binding a C program to the database must have authorization to bind each of the tables in the program.

There are three user types: administrator, local administrator, and user. Each type issues various levels of authorization. Database access is controlled by User Profile Management. Privileges can be granted to users to allow access to certain objects. Privileges may be granted to create tables. The creator of tables has the authorization to grant access privileges to those tables. Authorizations to access plans are separate from authorizations to objects, but may be implicitly provided if needed.

Database Manager automatically creates several objects when a database is created. The SQL00001, SQL00002.... directories for the first, second, and subsequent databases contain object information about the databases. Catalog tables are created which contain descriptions of all data. A recovery log is maintained for each database. Each table created in Database Manager must have one configuration file. The configuration file is created whenever the database is created. The configuration file contains information about resources allocated to the database. It may be tailored for performance turning. In addition to a configuration file for each database, there is one configuration file for the Database Manager.

The security measures and the complexity of files to be generated and maintained make it difficult to develop a system for users unfamiliar with Database Manager. Most features offered in Database Manager are not needed in CRE. The complexities of Database Manager are a hindrance to CRE database access.

Database Manager meets the requirements of CRE; however, due to its complexity and limited access by other software products, it is not recommended as the CRE database solution.

In-House Designed C Database Product

Most systems developed to provide users a variety of information require a database. Many PC developed systems use simple ASCII files containing database information. These systems do not require purchased database software packages and ASCII file data access requires less computing time than the execution of database input and output

functions. In-house code is the accepted method of data access for ASCII files only. If the product is to utilize a more structured method of data storage, in-house code becomes questionable.

Recent technology advances have created a market for more sophisticated databases, which allow access to certain fields, browse, indexes, quick searches, predefined data types, and graphical interfaces. A few developers write custom database software packages; however, with the many database software packages available, it is not always feasible to develop an in-house database system interface. Unless the development process itself is needed as an educational experience, an in-house database development is not recommended. Marketed database software packages offer more flexibility than a customized package, with a wide variety of tools available to manipulate databases. Marketed database software packages also allow users and developers to migrate the databases to other platforms.

Q+E Database Library

Q+E Database Library (QELIB) by Pioneer Software has been praised by several computing magazines. Q+E Database Editor (another product by Pioneer Software) is still a widely used and popular database tool that allows a graphical interface to several different data structures. QELIB supports Oracle, Microsoft, and Sybase SQL Servers, IBM DB2, Novell NetWare SQL servers, Paradox, dBase, ASCII text, and other file formats. It is compatible with OS/2 and Microsoft Windows (separate packages must be purchased for each.)

QELIB allows any programming language that allows calls to Dynamic Link Libraries (DLL) to access data directly through SQL function calls. The C compiler used by CRE (C/2 Version 1.1) and the IBM C Set/2 (which IBM has recently released) have the ability to make DLL calls.

QELIB offers the flexibility and speed of C code, while retaining the database control of a native dBase language. Many of the functions available in a native dBase language are available in C through use of QELIB. All dBase indexes are supported for performance. The DBASE IV version of indexes is available for ease in using multiple indexes.

QELIB meets the requirements for CRE. It executes under all versions of IBM OS/2. It allows database update, create, and deletion through IBM C/2 code. It has a runtime version. It provides the ability to be used even if the C compiler or database platform is changed. It provides C code performance. The ability to interface with ASCII text is a plus, since CRE utilizes many ASCII files along with the database tables.

The ability to store tables in the dBase format allows users and the CRE development team to utilize any dBase software product to produce reports or graphics.

QELIB provides the ability to access data with only minimal assistance. There is no connectivity time lag because access to the database tables is constructed during the compilation of the C code. The C code directs the SQL commands to link to the

appropriate database driver.

The single developer QELIB costs $399, with a runtime version available for $999. This cost is reasonable from both a development and distribution viewpoint.

CHAPTER 5

PROPOSED DATABASE SOLUTION

Database Software Solution

It is recommended that the developers of CRE acquire the development and runtime versions of QELIB by Pioneer Software. This software product provides the ability to access and update dBase structured tables through CRE's C code. QELIB can be used even if CRE is ported to another C compiler or newer versions of OS/2. Acquisition of the runtime software will allow CRE to be distributed to any OS/2 user without the requirement of additional software.

Database Methodology

Tables for CRE can be created using any dBase product or QELIB. QELIB can only create tables and indexes through SQL code. The dBase platform is suggested because of the large number of users familiar with the dBase products, as well as the large number of software tools available for their use. If a dBase product is used instead of QELIB, Ashton-Tates' DBASE IV software product is the recommended product by which to create the DBF tables and indexes. DBASE IV indexes are more effective with QELIB than other formats.

QELIB can be installed using the DBF database driver (the ASCII driver is optional). No dBase software product needs to be integrated into CRE. (DBASE IV can be used to create tables initially, but is not required.) CRE IBM C/2 code can make DLL calls to QELIB to access, create, and update the DBF tables.

SQL commands can be embedded in the C/2 code to connect to the DBF database driver. Database access is performed using DLL calls in the form of SQL statements within the C code. Select statements allow selection of records based on various criteria. The retrieved data record can be manipulated using C code. Updates to tables are performed with SQL update statements.

CRE should support several different software projects and increments concurrently. Each project can be developed and certified in any number of increments (much like separate projects). This can be accomplished by storing all database tables and ASCII files for one increment in a separate directory designated for that increment.

The directory should be named with the project name as the primary name, and the directory extension should be the number of the increment. CRE is to provide a default project and increment. If only one project is available, that project is to be automatically selected along with the latest increment. If more than one project is available, the most recent unfinished project and increment are to be used as the defaults.

The user may change the project and the increment that are to be accessed by entering the Project Identifier subsystem. Until the directory selection is changed, all data access is from that directory.

Placing all data for one increment in one directory allows all tables and ASCII files to have a common identifying primary name. This structure allows each subsystem to automatically select the appropriate data without the assistance of the user selecting files (adding to the user friendliness). Multiple ASCII files for the same increment are allowed. The user may select the file from a list of names distinguished for that increment only. This method is a change from the existing practice of requiring the user to select a file after determining the appropriate directory.

Changes to the existing CRE data structures and processing control are minimized. The existing data structures do not change (except the TST, TTF, and DB ASCII files). These files are to be replaced by a combination of new Failure and Engineering Changes tables.

There are eleven new tables and two new ASCII files. The new data structures are divided into four categories: 1) Project Data, Failure Data, Project Development Data, and Reliability Model Data. The Failure Data contains a Failure Table and Engineering Change Table that can be used to create input to the Certification Model subsystem, thereby eliminating the need for the TST and DB files. The Failure Table can directly replace the TST file that is to be used as input to the Testing Markov Chain subsystem.

The DBS subsystem should be replaced. The DBS subsystem is to be used to access all of the new database tables (with the exception of project and increment data). Each of the eleven database tables is to be updated through C code that provides an update window for that database table. Each field is to be validated before an update to the table is made. It is recommended that CASEPM (Computer Aided Software Engineering For Presentation Manager) by CASEWorks be used as a development tool. CASEPM is a GUI (Graphical User Interface) for developing software that automatically generates IBM C/2 code for a Presentation Manager graphical window. The C code generated can be integrated into CRE's code.

CHAPTER 6

PROPOSED DATA STRUCTURES

There are thirteen proposed new data structures. These structures include eleven dBase tables and two ASCII files. All dBase tables must have the file extension .DBF. The Certification Reliability Model and Policy ASCII files are system generated by the Policies subsystem. The Project Data tables are selected and updated from the Project Identifier subsystem. The Totals table is updated by the CDLP and the UME subsystems. All other tables are updated through the DBS subsystem by the user. Many of the tables contain metric information for purposes of management reports. This chapter provides a brief description of and use of each proposed data structure.

Indexes in DBASE IV tables may contain more than one index and are automatically opened in QELIB when the table is opened. Indexes should have the extension name .MDX and have the same primary name as the table. Indexes provide a means of fast, direct access to a specific record. Fields that are indexed may be matched to a value when selecting records from a table, and record selection may be made in ascending or descending order of the indexed field. Only fields that are to be used to select records should be indexed. Indexes must be updated each time the table is updated. A large number of indexes on one table creates a lengthy update time for the table. Each data structure is described with a recommendation of the fields to be indexed.

The data structures are defined using a list of field names, data types, and descriptions. Field names may be up to ten characters and may contain letters, numbers, or the underscore character. Field names cannot contain blanks, and the first character must be a letter.

Data types used are defined as: Char, Date, Numeric, and Memo. The Char (or Character) may contain letters, number, or any of the punctuation keys. A length parameter provides the maximum length of the character value. The Numeric values may contain only numbers. The values may include a decimal point and a leading minus sign. Two parameters provide the maximum number of digits in the number and the number of digits to the right of the decimal point. The second parameter is optional. The Date data type contains date values. The Date field allows several date formats; however, the MM/DD/YYYY format is recommended. The Memo data type contains a multi-line text data. The value may contain up to 65,535 characters.

Project Data

Project Table

The PROJECT.DBF table is edited from the Project Identifier subsystem. There is one record for each project. The table contains the start date, finish date, description, and general comments for every project available in CRE.

Table Name: PROJECT.DBF

Field Name	Data Type	Description
PRJ_NAME	Char 8	Project Name
PRJ_DESC	Char 30	Project Description
PRJ_STD	Date (mm/dd/yyyy)	Project Start Date
PRJ_FND	Date (mm/dd/yyyy)	Project Finish Date
COMMENTS	Memo	Project Comments

Project Name is to be alphanumeric. It must be no less than one character and no more than eight characters.

The date fields must be in MM/DD/YYYY format.

PROJECT.DBF should include indexes on the PRJ_NAME, PRJ_STD, and PRJ_FND fields.

Increment Table

The INCR.DBF table is edited from the Project Identifier subsystem. There should be a directory created for each record in this table. All data for a particular increment should be stored in that directory. There is one record for each increment associated with all projects. The table contains the start date, finish date, description, and general comments for every increment project available in CRE. This table is updated throughout the life cycle of the increment. The table format is provided below:

Table Name: INCR.DBF

Field Name	Data Type	Description
PRJ_NAME	Char 8	Project Name
INCR_NO	Numeric 5	Increment Number
INCR_DESC	Char 30	Increment Description
INCR_STD	Date (mm/dd/yyyy)	Increment Start Date
INCR_FND	Date (mm/dd/yyyy)	Increment Finish Date
COMMENTS	Memo	Increment Comments

The Project Name must match a record in the PROJECT.DBF. When an INCR.DBF is added, the INCR_NO is automatically defaulted to the next consecutive number.

INCR.DBF should include indexes on the PRJ_NAME and INCR_NO fields.

Failure Data

Version Table

The VERSION.DBF table is used to record each version of software that has been submitted for certification. If failures are detected in version one, one or more engineering changes are made, and version two is submitted for certification. This process is

repeated until certification is completed. There is a record for every version of software submitted. The table format is provided below:

Table Name: **VERSION.DBF**

Field Name	Data Type	Description
VER_NO	Numeric 3	Version Number
SUB_DT	Date (mm/dd/yyyy)	Date Version was Submitted for Certification
COMMENTS	Memo	Version Comments

The Version Numbers are automatically defaulted to the next consecutive number.

VERSION.DBF should have an index on the VER_NO field.

Failures Table

The FAIL.DBF table replaces the logical failure table in Query Manager, as well as the TST file in the current system. Additionally, the FAIL.DBF table is used with the EC.DBF table to create the ASCII Certification Reliability Model file to be used as input to the Certification Model subsystem. The Certification Reliability Model file replaces the TTF and DB files used by the current Certification Model subsystem.

This table contains all of the failures observed during the testing process. There is one record for each failure. All failures are numbered consecutively to indicate the failure number. The table format is provided below:

Table Name: **FAIL.DBF**

Field Name	Data Type	Description
FAIL_NO	Numeric 5	Failure Number
TEST_CASE	Numeric 5	Test Case Number
VER_NO	Numeric 3	Version Number of Software
TRAN_NO	Numeric 5	Transition Number
F_RESULT	Char 4	Failure Result (CONT or STOP)
T_TYPE	Char 11	Test Type (compilation or testing)
E_CAUSE	Char 10	Error Cause (Commission or Omission)
E_TYPE	Char 14	Data Error Type (Computation, Control, Cosmetic, Data, Initialization, Interface)
ERR_MESS	Char 30	Error Message
BLK_BY	Numeric 5	Failure Number of the failure that "blocked" the appearance of this failure earlier
TESTER	Char 3	ID of Person Performing Test
TEST_DT	Date (mm/dd/yyyy)	Date of Failed Test
TEST_TM	Char 5 (hh:mm)	Time of Failed Test
SITE	Char 10	Site of Failed Test
PREV_INC	Char 8	Failure From Code in Previous Increment
EC_NO	Numeric 5	Error Due to Engineering Change
USED_CD	Char 20	Failure From Pre-written Code
COMMENTS	Memo	Failure Comments

The Test Case field is populated with the test case number corresponding to the test case number in the SEQ file.

Version Number is populated with the version number of the software being tested. The Version Number must match a record in the VERSION.DBF.

The Transition Number corresponds to the transition number in the SEQ file that lists a series of actions to be performed.

The Failure Result indicates whether the failure caused termination (STOP) or allowed continuation (CONT) of testing.

The Test Type field identifies whether the failure occurred during compilation or testing.

The Error Cause field identifies the failure as either being caused by data commission or omission.

The Data Error field categorizes the error as one of the following: Computation, Control, Cosmetic, Data, Initialization, or Interface.

The Blocked By field identifies a Failure Number of an error that occurred previously which delayed the appearance of this failure. It must match the FAIL_NO of an existing failure.

The Tester field identifies the tester conducting the failed test.

If multiple sites are used for testing, the Site field is used to identify where the failed test was conducted.

If the failure is from code certified in a previous increment, the Previous Increment field contains the Increment Number that matches a record in the INCR.DBF table.

If the failure is due to an engineering change in this increment, the Engineering Change Number is populated with the associated Engineering Number matching a record in the EC.DBF.

If the failure is due to faulty code not written by the development team, the Used Code field is populated with a description of that code.

FAIL.DBF should include indexes on the FAIL_NO, TEST_CASE, VER_NO, and the EC_NO fields.

Engineering Changes Table

The EC.DBF table contains data about each engineering change made. Engineering changes are made to a software version to correct errors. More than one engineering change may be made to the same version of software. The EC.DBF table contains a

record for each engineering change. All engineering changes are numbered consecutively. The table format is provided below:

Table Name: EC.DBF

Field Name	Data Type	Description
EC_NO	Numeric 5	Engineering Change Number
VER_NO	Numeric 3	Version Number
DESIGNER	Char 30	Name of Person Designing Change
SEVERITY	Char 5	Code Indicating the Severity of Change
CLASSIFY	Char 5	Code Indicating Change Classification
COMMENTS	Memo	Comments About Change

The Version Number must match a record used in the VERSION.DBF. The maximum VER_NO in the VERSION.DBF is used as a default.

The Designer field is populated with one or more of the identities of the engineering change designers.

The Severity Code is a user defined code that indicates the severity of the change or error requiring the change. Severity codes are defined by the user.

The Classification Code is a user defined code that indicates the type of change. Classification codes are defined by the user.

EC.DBF should include indexes on the EC_NO and FAIL_NO.

Project Development Data

Verification Table

The VERIFY.DBF table contains information about each verification session. This table contains one record for each verification session. Most verification sessions are held for version one of the software, before the first compilation of the software. However, there should be a verification record for every engineering change that is made to subsequent versions of the software. The table format is provided below:

Table Name: VERIFY.DBF

Field Name	Data Type	Description
VF_NO	Numeric 5	Verification Session Number
VER_NO	Numeric 3	Version Number Being Verified
VF_DT	Date (mm/dd/yyyy)	Date of Verification Session
VF_ST_TM	Char 5 (hh:mm)	Starting Time of Session
VF_FN_TM	Char 5 (hh:mm)	Ending Time of Session
COMMENTS	Memo	Comments about verification session

The Verification Numbers are defaulted to the next consecutive number.

A Version Number must be associated with each verification session. The maximum existing Version Number is used as the default. The VERSION.DBF record must be entered before the VERIFY.DBF record.

VERIFY.DBF should include indexes on the VF_NO and VER_NO fields.

Errors Found Table

The FOUND.DBF table contains a record for each error found during verification sessions. Each record in this table must have a corresponding record in the VERIFY.DBF table. FOUND.DBF records are to be created after the VERIFY.DBF record. There is a many-to-one relationship with the VERIFY.DBF table. The Verification Number is used as the key. The table format is provided below:

Table Name: **FOUND.DBF**

Field Name	Data Type	Description
VF_NO	Numeric 5	Verification Number
ERR_DESC	Char 30	Description of Error Found
E_CAUSE	Char 11	Cause of Error (Commission or Omission)
E_TYPE	Char 10	Data Error Type (Computation, Control, Cosmetic, Data, Initialization, Interface)
COMMENTS	Memo	Comments

The Error Cause field identifies the failure as either being caused by commission or omission.

The Data Error field categorizes the error as one of the following: Computation, Control, Cosmetic, Data, Initialization, or Interface.

FOUND.DBF should have an index on the VF_NO field.

Verification Attendance Table

The ATTEND.DBF table contains data about team members' attendance at each verification session. Each record has a Verification Number and the identification of a participating team member. Every record in the VERIFY.DBF table should have one or more associated records in the ATTEND.DBF table. The table format is provided below:

Table Name: **ATTEND.DBF**

Field Name	Data Type	Description
VF_NO	Numeric 5	Verification Number
MEMBER	Char 3	Team Member Attending
PER_TIME	Numeric 3	Percent of Time Member Participated

The Verification Number must match a corresponding record in VERIFY.DBF.

The Member field contains a three character identification of the participating team member.

The Percent of Time indicates the percentage of the total verification session during which the member participated. (If the member participated for the entire time of the verification, the entry would be 100.)

ATTEND.DBF should include indexes on the VF_NO and MEMBER fields.

Hours Worked Table

The HOURS.DBF table is used to record time charged to all aspects of work.

Tasks are divided into the following classifications:

IP	(increment planning)
BSM	(design of box structures)
VF	(verification)
EC	(design of engineering changes)
UM	(creation of the usage model)
CT	(certification testing)
DOC	(documentation)
MISC	(miscellaneous time).

All hours spent on any Cleanroom task should be recorded in this table. (Formal verification sessions are recorded in the VERIFY.DBF. This table records any verification activity, including verification sessions.) This table contains data recorded weekly by each member of the Cleanroom team. There is one record for each member each week. The table format is provided below:

Table Name: **HOURS.DBF**

Field Name	Data Type	Description
MEMBER	Char 3	Team Member Attending
WE_DATE	Date (mm/dd/yyyy)	Week Ending Date
IP_HRS	Numeric 5.2	Hours Spent in Increment Planning
BSM_HRS	Numeric 5.2	Hours Spent Design Boxes
VF_HRS	Numeric 5.2	Hours Spent in Verification
EC_HRS	Numeric 5.2	Hours Spent Designing ECs
UM_HRS	Numeric 5.2	Hours Spent Developing Usage Model
CT_HRS	Numeric 5.2	Hours Spent Certifying
DOC_HRS	Numeric 5.2	Hours Spent Documenting
MISC_HRS	Numeric 5.2	Hours Spent in Miscellaneous Tasks

HOURS.DBF should include indexes on the MEMBER and WE_DATE fields.

Count Table

The COUNT.DBF table is entered by the user. The table provides a method of storing data frequencies. The number of design boxes, CDL, and C lines are stored in several different ways. There is only one record in the table.

All black boxes, state boxes, and clear boxes for an increment should be counted and categorized as New, Old, Changed, or Deleted. All lines of CDL and C files should be counted and placed in the same categorization as the boxes. No lines of CDL or C should be counted in more than one category. The user should populate this table after the increment has been certified. The table format is provided below:

Table Name: **COUNT.DBF**

Field Name	Data Type	Description
USED_CD	Numeric 8	Reused Code (not written by team)
ND_LINES	Numeric 8	New Design Language (BSM) Lines
NS_LINES	Numeric 8	New Source Code (Clear Box) Lines
OD_LINES	Numeric 8	Old Design Language (BSM) Lines
OS_LINES	Numeric 8	Old Source Code (CLear Box) Lines
CD_LINES	Numeric 8	Changed Design Language (BSM) Lines
CS_LINES	Numeric 8	Changed Source Code (Clear Box) Lines
DD_LINES	Numeric 8	Deleted Design Language (BSM) Lines
DS_LINES	Numeric 8	Deleted Source Code (Clear Box) Lines
BB_NEW	Numeric 8	Number of New Black Boxes
SB_NEW	Numeric 8	Number of New State Boxes
CB_NEW	Numeric 8	Number of New Clear Boxes
BB_OLD	Numeric 8	Number of Old Black Boxes
SB_OLD	Numeric 8	Number of Old State Boxes
CB_OLD	Numeric 8	Number of Old Clear Boxes
BB_CHG	Numeric 8	Number of Changed Black Boxes
SB_CHG	Numeric 8	Number of Changed State Boxes
CB_CHG	Numeric 8	Number of Changed Clear Boxes
BB_DEL	Numeric 8	Number of Deleted Black Boxes
SB_DEL	Numeric 8	Number of Deleted State Boxes
CB_DEL	Numeric 8	Number of Deleted Clear Boxes

Boxes may be classified as new if no lines in the box existed from a previous increment. Boxes may be classified as old if the box has not been changed from a previous increment.

Boxes are considered changed if modifications have been made. The changed lines can be counted as the total after any number of deletions, modifications, or additions to an existing box. For example: if a clear box of 60 lines had an additional line added and three lines deleted, the result would be 58 lines changed, because even though only one line is added and three lines are deleted, the 58 remaining lines constitute a changed module of code and should be counted as changed (Cleanroom Software Engineering Documentation Study, p. 6-4). If three lines of code are changed in a clear box, the functionality of that box may be entirely changed. The clear box is the smallest measure

of functionality. A changed clear box should be reverified. Any reliability associated with the clear box in a previous increment should be void.

Deleted boxes are boxes that existed in the previous increment, but are no longer being used.

The number of design lines includes the total number of lines in the black boxes, state boxes, and clear boxes for the different classifications (new, old, changed, and deleted).

The number of source lines include the total number of lines in the clear boxes only for the different clear box classifications (new, old, changed, and deleted).

The Reused code field indicates the number of lines of code used, but not written by the team.

The COUNT.DBF table contains only one record and needs no index.

Totals Table

The TOTALS.DBF table is system generated. The CDLP and UME subsystems are to populate this table when creating the C file and FRE file. This table has only one record. The information in this table should complement the user defined counts in the COUNT.DBF. The table format is provided below:

Table Name: **TOTALS.DBF**

Field Name	Data Type	Description
TOT_DL	Numeric 8	Total Design Language Lines
TOT_SL	Numeric 8	Total Source Code Lines
TOT_BB	Numeric 8	Total Number of Black Boxes
TOT_SB	Numeric 8	Total Number of State Boxes
TOT_CB	Numeric 8	Total Number of New Clear Boxes
TOT_US	Numeric 8	Total Number of States in Usage Model
NO_ARCS	Numeric 8	Number of Arcs in the Model
TOT_NZ	Numeric 8	Total Number of Non-zero Matrix Entries
PER_NZ	Numeric 5.2	Percent of Non-zero Matrix Entries

The CDLP subsystem is to populate the total CDL lines (TOT_DL), C lines (TOT_SL), black boxes, state boxes, and clear boxes when processing the CDL file.

The UME subsystem is to populate the total number of states in the usage model, number of arcs, total non-zero matrix entries, and the percent of non-zero matrix entries when creating the FRE file.

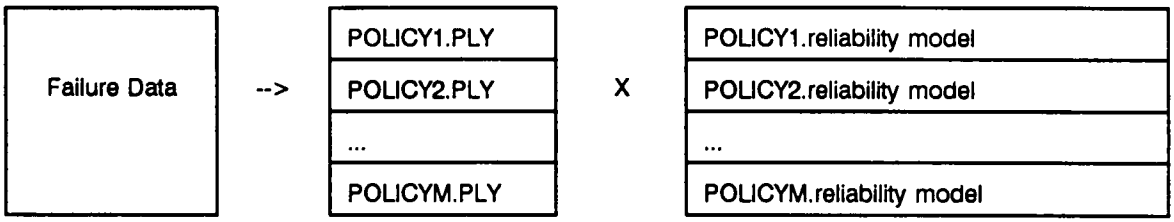
The TOTALS.DBF table contains only one record and needs no index.

Reliability Model Data

Certification Reliability Model File

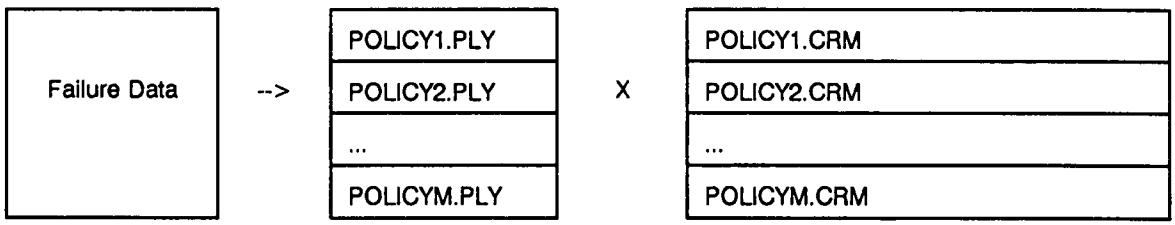
The Policies subsystem (which has not been implemented) should be used to interpret failure data and prepare files to be used as input to a selected reliability model. The Policies subsystem should allow the user to select a policy and reliability model. There is no need for the user to select the failure data, the failure data (FAIL.DBF and EC.DBF tables) in the currently selected project and increment directory is automatically used. After the user enters the policy and reliability model selections, the Policy subsystem should generate a file by interpreting the FAIL.DBF and EC.DBF tables according to the selected policy. The generated file may be used as input to the selected reliability model. A reliability model should accept only one kind of file.

The input file to a reliability model should have a primary name matching the policy and an extension name that reflects the specific reliability model for which it is intended.



In order to use the proposed FAIL.DBF and EC.DBF tables, the Policies subsystem should be implemented to allow the user to select a policy for the Certification Reliability Model. Future increments may include updates to the Policies subsystem to allow selection of various other reliability models with additional policies. This section details the proposed Certification Reliability Model (CRM) file to be generated by the Policies subsystem as input to the Certification Model subsystem.

If the user selects the Certification Reliability Model in the Policies subsystem, the generated output file is to have the primary name matching the policy file name and the extension CRM. The CRM file is to be used as input to the Certification Model subsystem. The Certification Model subsystem currently accepts two different files (TTF and DB) as input; however, the proposed design allows only the CRM file as input.



The CRM file contains lines of failure data with the Independent Variable and Failure Time values on each line. (Reliability models included in future increments may have a different format.) These data lines are created in the Policies subsystem by interpreting failure data (FAIL.DBF and EC.DBF) based upon the selected policy.

*.CRM
<Independent Variable> <Failure Time> <NewLine>
<Independent Variable> <Failure Time> <NewLine>
<Independent Variable> <Failure Time> <NewLine>
...

Policy File

Policy (PLY) files are to contain policy criteria. A PLY file is to be used to interpret failure data while creating a reliability model file. The proposed Policies subsystem allows the user to create PLY files and use them to interpret failure data.

When creating a PLY file, the Policies subsystem should allow the user to define the policy criteria for the Certification Reliability Model. There are two proposed types of policy codes. The first policy code determines the Independent Variable - Version Numbers (VN) or Engineering Change Numbers (ECN). The second policy code, Failure Reporting Policy (FRP), determines the failures to be included. The following Failure Reporting Policies are proposed:

RAF	-	Report All Failures
EFO	-	Report Execution Failures Only
AEC	-	Report All Failures Except Cosmetic
OWC	-	Report Only Failures Cause By Code Written in Project
OIC	-	Report Only Failures Cause By Current Increment

Other Independent Variable Policies or Failure Reporting Policies may be incorporated in a future increment.

After the user defines a PLY file, the file contains only one line. The PLY format is:

*.PLY
<Independent Variable Policy> <FRP> <FRP> <FRP> <FRP> <FRP>

CHAPTER 7

PROPOSED CRE ADDITIONS AND MODIFICATIONS

Proposed CRE Overview

This chapter describes the proposed additions and modifications to CRE. The proposed Project Identifier, Policies, and Reports subsystems are described as additions to CRE. The proposed DBS subsystem is described as a replacement of the current DBS subsystem. All other modifications required to CRE are described.

The proposed CRE uses dBase and ASCII files to store and retrieve data. Figure 7.1 shows the proposed CRE data flow. ASCII file names are shown as "*.file extension" (the asterisk is used in place of the file name given by the user). The file extension identifies the type of file. The dbase tables must use the file extension .DBF. The dbase primary name identifies the table. Only one dBase table of each type is allowed per increment, but ASCII files may have multiple files of the same type for one increment.

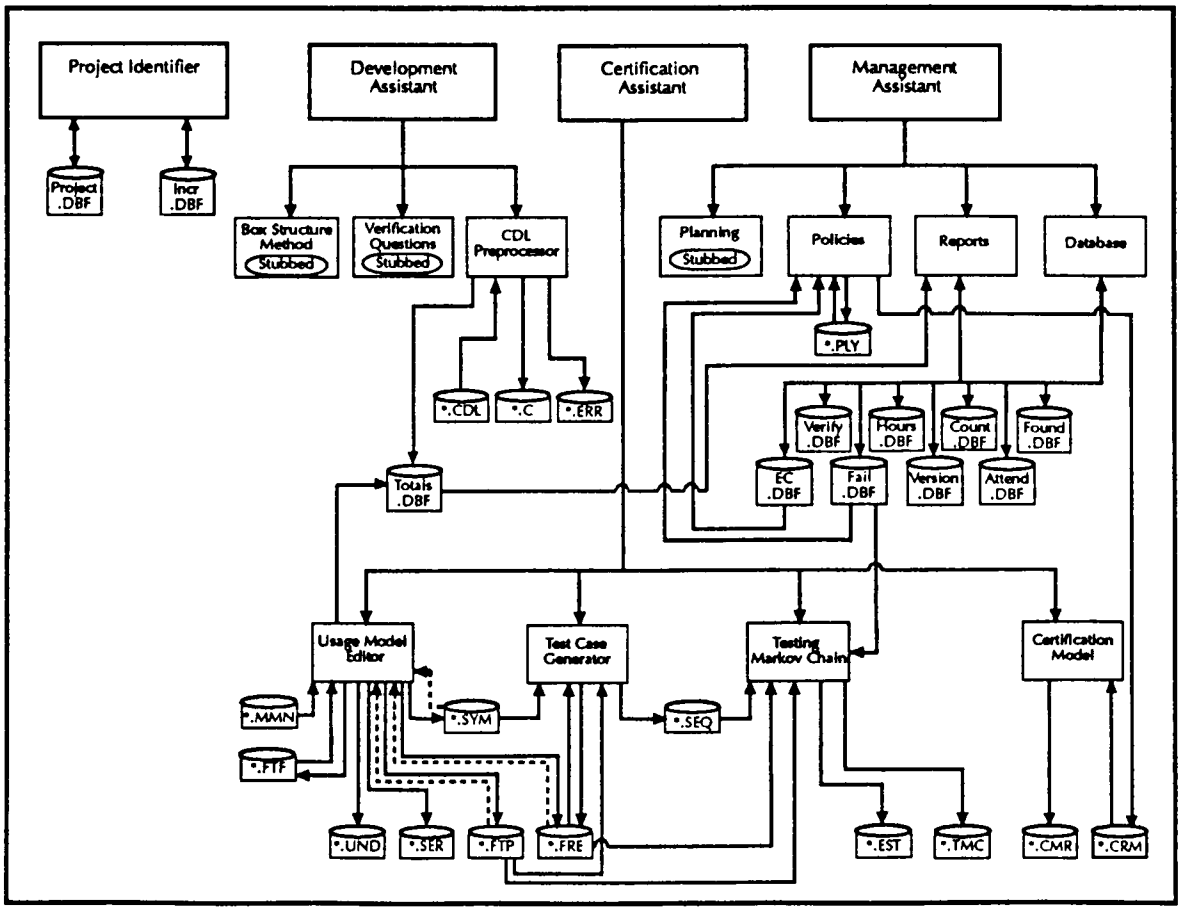


Figure 7.1 Proposed CRE Data Flow

Project Identifier

The Cleanroom Environment control window is presented after initiation of CRE. This window presents access to the other control windows: 1) Development Assistant, 2) Certification Assistance, and 3) Management Assistant. The proposed Project Identifier (PI) subsystem is to be included in the Cleanroom Environment control window. The proposed Cleanroom Environment window is shown in Figure 7.2.

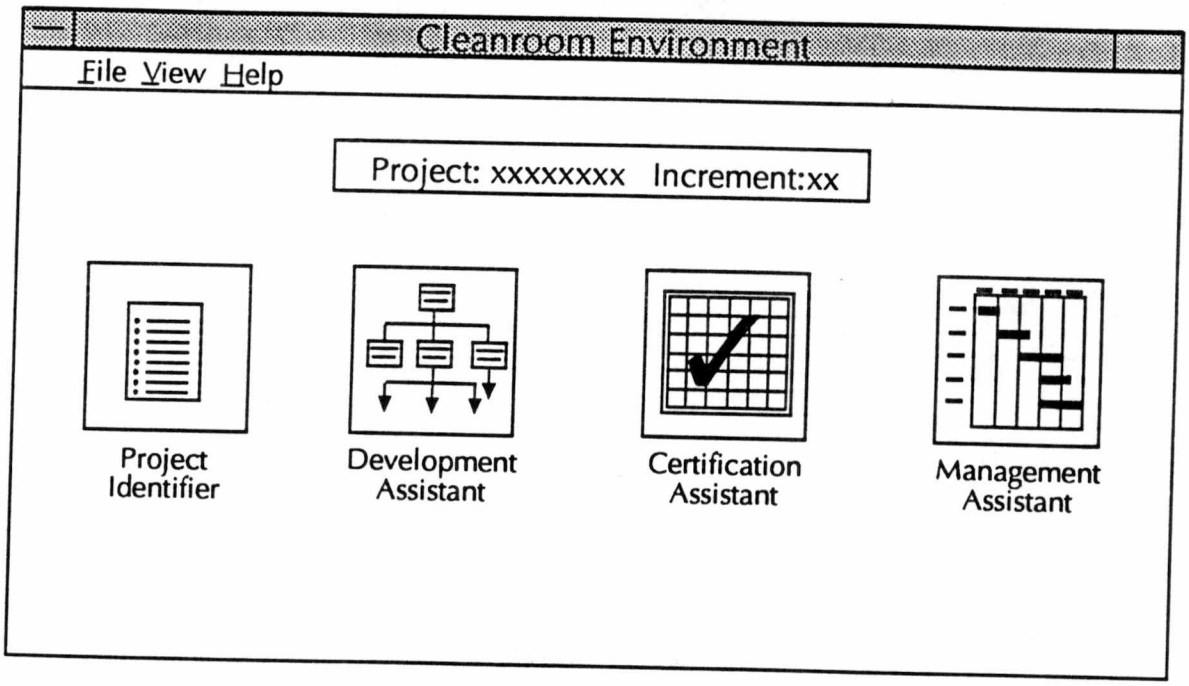


Figure 7.2 Proposed Cleanroom Environment Window

The proposed database structure relies on the ability to identify a particular directory before accessing other Cleanroom subsystems. All subsystems limit access to data from the selected project/increment directory. The defaulted project/increment may be used without entering the PI subsystem. The PI subsystem must be used to change the selection. The user may enter the PI subsystem to change the selected project/increment or edit the PROJECT.DBF and INCR.DBF tables.

All control windows display the selected project and increment. CRE.PRG code (which generates the Cleanroom Environment window) accesses the PROJECT.DBF and INCR.DBF to obtain the default project and increment. SQL is used to obtain the PROJECT.DBF. The fields selected are PRJ_NAME, PRJ_STD, and PRJ_FND (where PRJ_FND is blank). The PRJ_STD is to be indexed. The MAX function may be used on the PRJ_STD field to find the project with the latest start date, but not completed. The INCR.DBF table is accessed through SQL statements to select the records where PRJ_NAME equals PRJ_NAME of the PROJECT.DBF record. The MAX function may be used on the INCR_NO field to find the latest increment of the project. The INCR.DBF table should have an index on both the PRJ_NAME and the INCR_NO fields. If no

INCR.DBF records are associated with the PROJECT.DBF, a blank Increment is given as a selection. No system access is allowed until the INCR.DBF record has been added or another PROJECT.DBF record is selected.

The proposed CRE is directory driven. The executable files and the PROJ.DBF and INCR.DBF tables reside in the directory C:\CLEANRM. Each time an increment within a project is created, a new directory is created from the CLEANRM directory. The new directory has the same primary name as the project, and the extension is the number of the increment. If the project or increment is deleted from CRE, all data are erased, and the directory is deleted.

When a new directory is created for an increment, original versions of the remaining nine dBase tables (all tables except PROJ.DBF and INCR.DBF) are copied to that directory. Original versions of tables are stored in the directory C:\CLEANRM\ORIG. The COUNT.DBF and TOTALS.DBF tables have one blank record originally stored. These tables never contain more than one record (additions are not allowed). All other original dBase tables contain no records. In addition to the DBF files that are automatically placed in the project/increment directory, all ASCII files that are user created for that increment must be stored in the appropriate directory.

After opening the PI subsystem, the Project Identifier window is displayed. The Project Identifier window is shown with the pull-down menus in Figure 7.3.

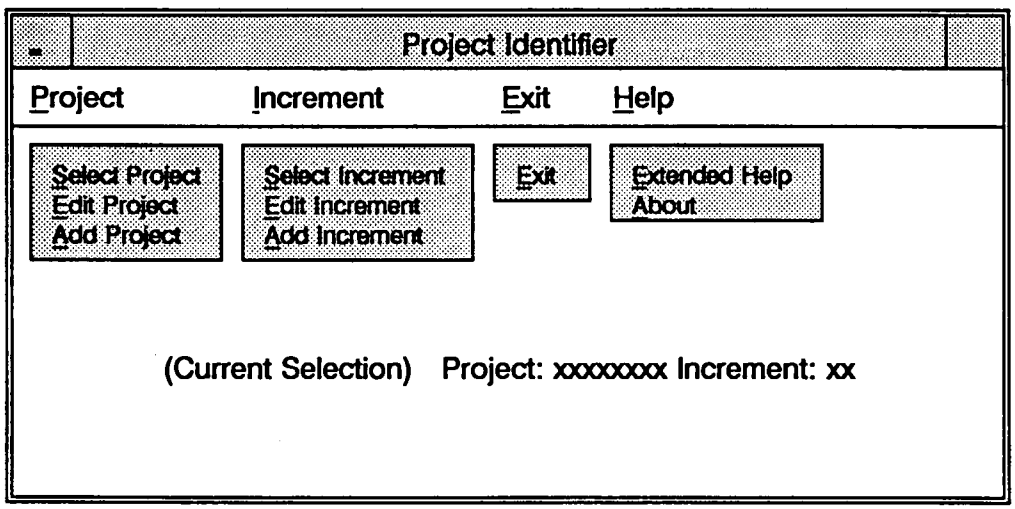


Figure 7.3 Project Identifier Window

The current selection is displayed indicating the selected project name (xxxxxxxx) and increment number (xx). If the user has not changed the selection, the current selection is the default.

Project Pull-Down Menu

The **Select Project** option allows the user to change the project selection. After selecting this option, the Project Selection window (shown in Figure 7.4) is displayed. The user may select a project from the list of available projects, or by typing the project name in **Project Name**. The user may choose to exit this window without selecting a different project. After selecting a project or exiting, the Project Identifier window is again displayed.

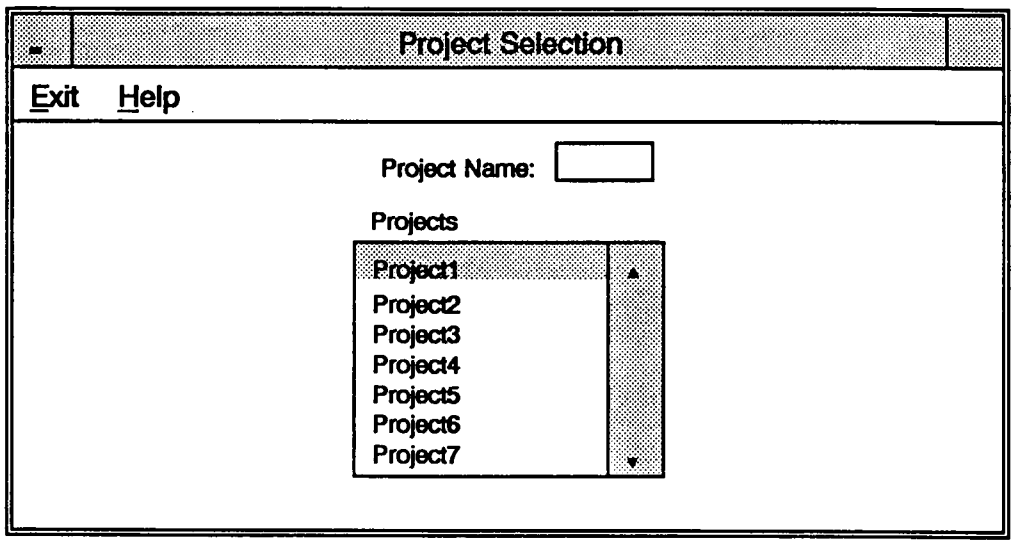


Figure 7.4 Project Selection Window

The PROJECT.DBF table and PRJ_NAME indexes are used to display the list of projects. The list of projects is displayed in the manner as is a list of directories in the current CRE.

If a different project is selected, the INCR.DBF file is selected for all records that have the PRJ_NAME field matching the PRJ_NAME field of the PROJECT.DBF tables. The MAX function is used to select the latest increment record. The new project and the associated latest increment are displayed as the current selection in the Project Identifier window.

The **Edit Project** option allows the user to edit or delete the project record in the PROJECT.DBF table. If the user selects this option, the Edit Project window (shown in Figure 7.5) is displayed. The currently selected project is used as the project record to be edited or deleted.

The **Action** pull-down menu contains four options: **Save Changes**, **Delete Project**, **Abandon Update**, and **Exit**. The **Help** option is used to display help information. If the user selects the **Delete Project**, a dialogue box is displayed inquiring if the user wishes to confirm or cancel the deletion. If the **Save Changes** option is selected, entries in the fields are validated. If no errors are found, the project record is changed to the new field

values. If no fields are changed or an error is found, a dialogue box is displayed with a message inquiring if the user wishes to exit or resume. If the **Abandon Changes** option is selected, any changes to fields are restored to their original values. The **Exit** option is used to return to the Project Identifier window. If changes were made before the **Exit** option was selected, a dialogue box is displayed indicating that changes were made, and inquires if the user wishes to exit without saving.

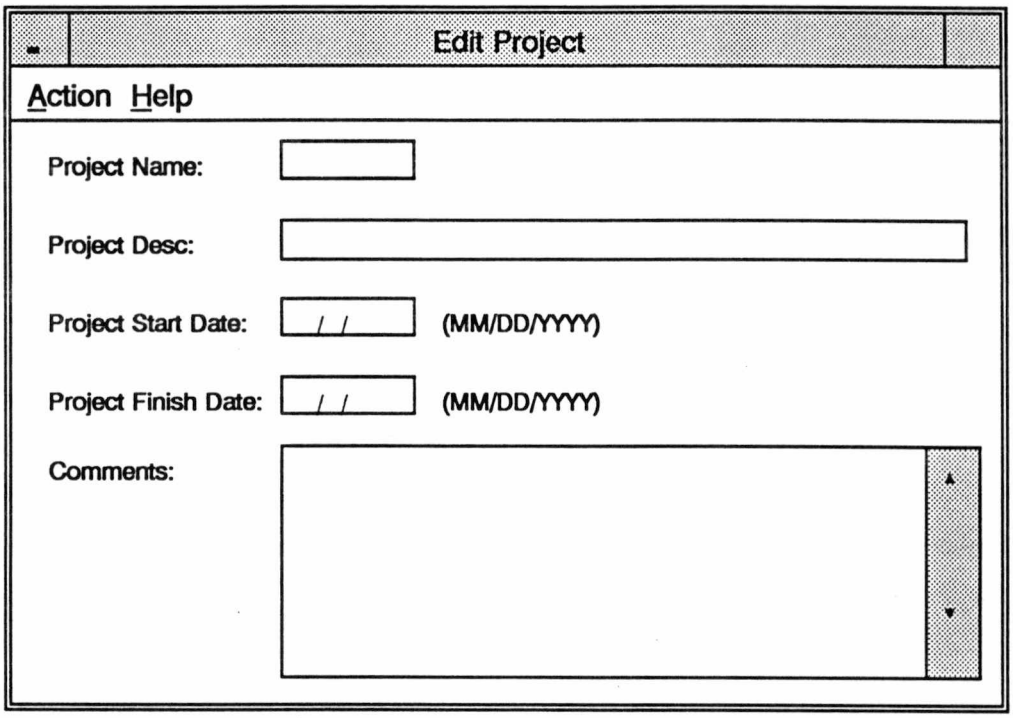


Figure 7.5 Edit Project Window

The **Add Project** option presents a window similar to the Edit Project window. The Add Project window (shown in Figure 7.6) presents project fields that are blank. This window provides the **Add**, **Exit**, and **Help** options with no pull-down windows. The user may populate the fields the same as in the Edit Project window.

The **Add** option is used to add the project. Validations are performed on the input fields after selection of the **Add** option. If no errors are found, a dialogue box is displayed indicating the error and inquiring if the user wishes to resume or cancel.

The **Exit** option is used to return to the Project Identifier window. If the project fields have been populated but not added, a dialogue box is displayed inquiring if the user wishes to exit without adding the project.

The user may update any of the displayed fields. A tab or enter may be used to proceed to the next field position. The mouse pointer may be used to select a field position.

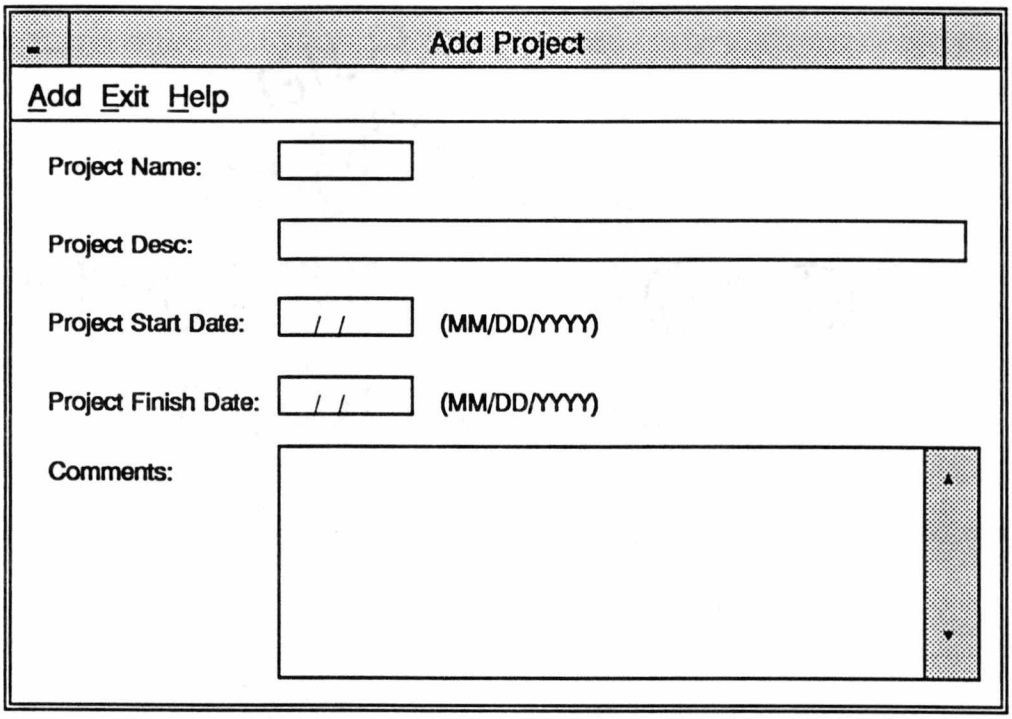


Figure 7.6 Add Project Window

Project Field Validations

The Project Name field (PRJ_NAME) is required. Duplicates are not allowed. If the Project Name is renamed, the directory name must be renamed, and all the INCR.DBF records for that project must have the PRJ_NAME field changed. If a Project Name is added, no directory is added until the first increment is added to the project. (All projects must have at least one increment.)

The Project Description field (PRJ_DESC) may contain any information. This field is not required.

The Project Start Date field (PRJ_STD) is required. It must be in MM/DD/YYYY format. The date should be checked for a reasonable value. It should not be greater than or less than five years from the current date.

The Project Finish Date field (PRJ_FND) is not required. If the date is populated, it must be in MM/DD/YYYY format. The date may not be less than or equal to the Project Start Date field. It should be checked for a reasonable value by not allowing a date greater than five years from the current date.

The Comments field is not required. It may contain any information defined by the user. This field is a dBase memo field. A memo field may contain up to 65,535 characters; however, a more reasonable number of characters may be considered for performance issues. The user may scroll the Comments Field by clicking on the arrow symbols.

Increment Pull-Down Menu

The Increment pull-down menu provides the same options as the Project pull-down menu. The **Select Increment**, **Edit Increment**, and **Add Increment** options allow the user to select or update an increment within the currently selected project. If no increment exists for the currently selected project, only the **Add Increment** option may be chosen.

The **Select Increment** option allows the user to change the increment selection within the selected project. After selecting this option, the Increment Selection window (shown in Figure 7.7) is displayed. The user may select an increment number from the list of increments available for the project, or by typing the increment number in the **Increment Number** field. The user may choose to exit this window without selecting a different increment. After selecting an increment or exiting, the Project Identifier window is displayed.

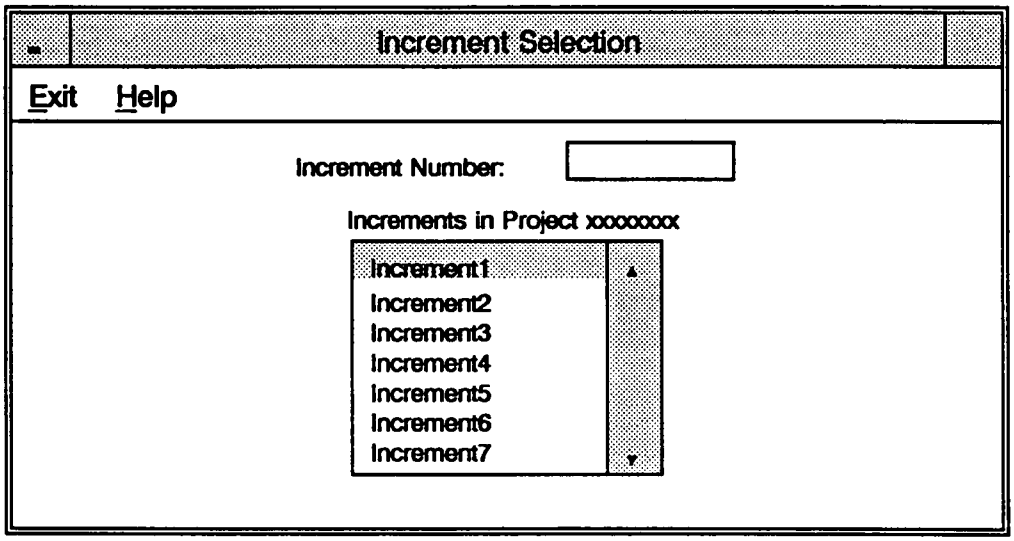


Figure 7.7 Increment Selection Window

The increments displayed are the INCR_NO fields retrieved from the INCR.DBF table for all records having their PRJ_NAME match the current project name from the PROJ_DBF table. The PRJ_NAME fields are indexed in both tables. Only the INCR_NO field from the INCR.DBF table needs to be retrieved. The INCR_NO fields are used to display the list of available increments.

If an increment is selected, the new increment number and the associated project are displayed as the current selection in the Project Identifier window.

The **Edit Increment** option allows the user to edit or delete the currently selected INCR.DBF record. If the user selects this option, the Edit Increment window (shown in Figure 7.8) is displayed. The Project and Increment Number is defaulted to the selected project and increment. The user may not update the Increment Number.

The **Action** pull-down menu contains four options: **Save Changes**, **Delete Increment**, **Abandon Update**, and **Exit**. The **Help** option is used to display help information. If the user selects the **Delete Increment** option, a dialogue box is displayed inquiring if the user wishes to confirm or cancel the deletion. If the **Save Changes** option is selected, entries in the fields are validated. If no errors are found, the INCR.DBF record is changed to the new field values. If no fields were changed or an error is found, a dialogue box is displayed with a message inquiring if the user wishes to exit or resume.

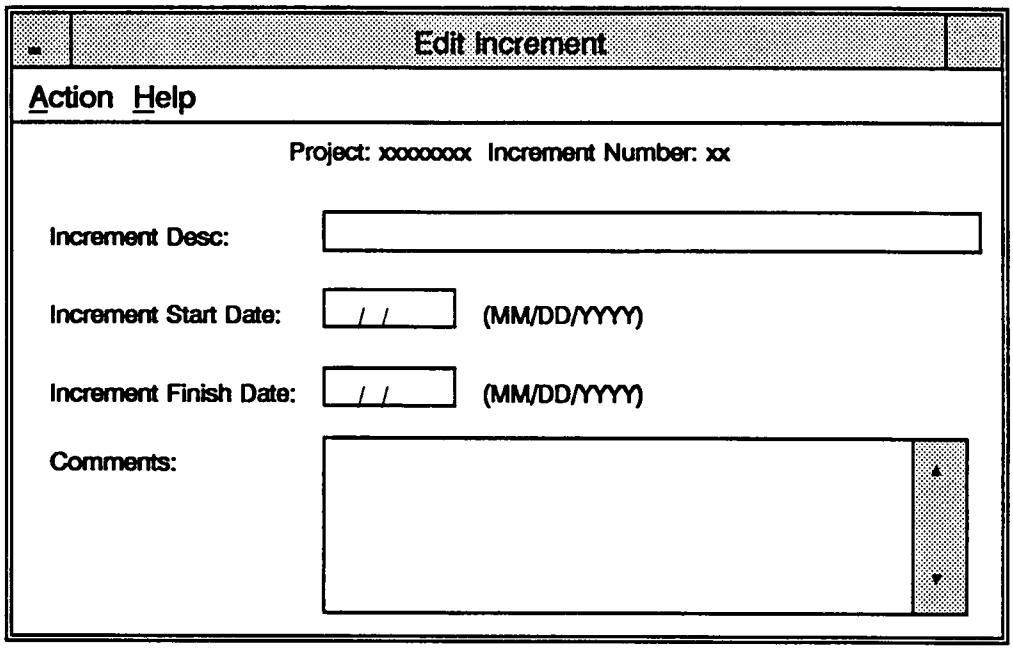


Figure 7.8 Edit Increment Window

If the **Abandon Changes** option is selected, any changes to fields are restored to their original values. The **Exit** option is used to return to the Project Identifier window. If changes were made before the **Exit** option was selected, a dialogue box indicates that changes were made and inquires if the user wishes to exit without saving.

The **Add Increment** option presents a window similar to the Edit Increment window. The Add Increment window (shown in Figure 7.9) presents increment fields that are blank. This window provides the **Add**, **Exit**, and **Help** options with no pull-down windows. The user may populate the fields in the manner used in the Edit Increment window.

The **Add** option is used to add an increment to the selected project. The Increment Number is defaulted to the maximum existing increment number, plus one. The Increment Number default may not be changed.

Validations are performed on the input fields after selection of the **Add** option. If an error is found, a dialogue box is displayed indicating the error and inquiring if the user wishes to resume or cancel. If no errors are found, a record is inserted into the INCR.DBF record with the fields populated as input. The PRJ_NAME field is automatically updated as the

PRJ_NAME in the selected PROJECT.DBF table. A new directory is created with the PRJ_NAME as the primary directory name and the INCR_NO as the directory extension.

When a new project/increment directory is created, original versions of all tables (except PROJ.DBF and INCR.DBF) are copied into the new directory. The COUNT.DBF and TOTALS.DBF tables are to contain one blank record each, while all other tables contain no records. The original tables should be stored in the C:\CLEANRM\ORIG directory. A *COPY C:\CLEANRM\ORIG*.** command may be used to populate the directory.

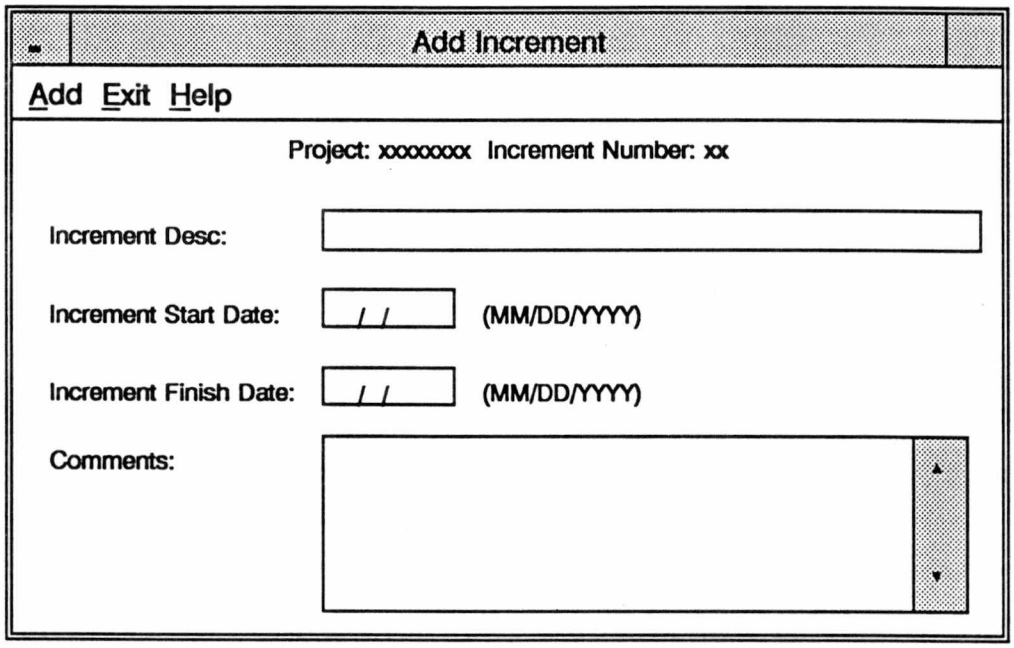


Figure 7.9 Add Increment Window

The **Exit** option is used to return to the Project Identifier window. If the increment fields have been populated but not added, a dialogue box is displayed inquiring if the user wishes to exit without adding the increment.

The Edit Increment and Add Increment windows do not allow update of the Project Name or the Increment Number. All other displayed fields may be updated. A tab or enter may be used to proceed to the next field position. The mouse pointer may be used to select a field position.

Increment Field Validations

The Increment Number field (INCR_NO) is stored in the INCR_NO field of the INCR.DBF table. The Increment Number is defaulted to the next numerically ordered number. An increment number cannot be changed.

The Increment Description field (INCR_DESC) may contain any information. This field is not required.

The Increment Start Date field (INCR_STD) is required. This field must be in MM/DD/YYYY format. The start date cannot be less than the PRJ_STD on the PROJ.DBF record. If the PRJ_FND date is populated on the PROJ.DBF record, the Increment Start Date cannot be greater than the project finish date. The date should be checked for a reasonable value. It should not be greater than or less than five years from the current date.

The Increment Finish Date field (INCR_FND) is not required. If the date is populated, it must be in MM/DD/YYYY format. The date may not be less than or equal to the Increment Start Date field. If the PRJ_FND date is populated on the PROJ.DBF record, the Increment Start Date cannot be greater than the project finish date. It should be checked for a reasonable value by not allowing a date greater than five years from the current date.

The Comments field (COMMENTS) is not required. It may contain any information defined by the user. This field is a dBase memo field. A memo field may contain up to 65,535 characters; however, a more reasonable number of characters may be considered for performance issues. The user may scroll the Comments Field by clicking on the arrow symbols.

DBS Subsystem Replacement

The Management Assistant control window presents four subsystem selections: 1) Planning, 2) Policies, 3) Reports, and 4) Database. The DBS subsystem is used to update and view failure and project development data. The tables included are: 1) VERSION.DBF, 2) FAIL.DBF, 3) EC.DBF, 4) VERIFY.DBF, 5) FOUND.DBF, 6) ATTEND.DBF, 7) HOURS.DBF, and 8) COUNT.DBF. (The TOTALS.DBF table is updated by the CDLP and UME subsystems.) When the user selects the DBS subsystem, the Database window (shown in Figure 7.10) is presented.

The Database window displays the available database information. Descriptions of the tables are used instead of the table names. The **Action** pull-down menu presents three options: 1) **Select Data**, 2) **Add Data**, and 3) **Exit**. The **Select Data** option is used to set the control to display a list of existing data records after selection of a data type. The **Add Data** option is used to set the control to allow the user to directly add a record after a data type selection is made. The **Exit** option is used to return to the Project Identifier window. The **Help** option is used to display help information. The user may make a selection of data by pressing the enter key on the highlighted data, clicking on the selection, or by pressing the underlined letter of the data. After selection of a data type, a window for the selected data type is displayed for either selection or add according to the control set in the **Action** pull-down menu. If a data type is selected and no action is specified, the **Select Data** option is used as the default.

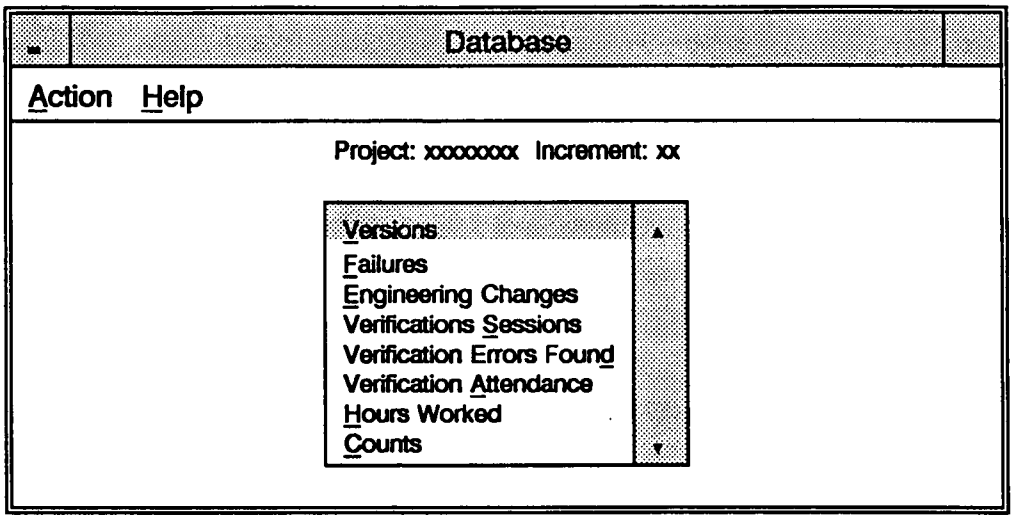


Figure 7.10 Database Window

If the **Action** is set to **Select Data**, the first ten records of the selected table is displayed. If the **Action** is set to **Add Data**, the add window of the selected data type is displayed. The selection window for each data type provides the ability to add a record without returning to the Database window. The add window for each data type is the same regardless of whether the add option was selected from the Database window or a selection window.

Versions

If the action control is set to **Select Data**, and the user selects the **Versions** data, the Versions Data window (shown in Figure 7.11) is presented. The Version Data window lists ten records in the VERSION.DBF table. The list of records may be scrolled by clicking on the arrow symbols at the top and bottom of the list. All of the fields in a VERSION.DBF record can be viewed in one line except the Comments field, which is truncated. The user may select a version by highlighting the selected version. The user may use the mouse or arrow keys to highlight different lines. The **Exit** option is used to return to the Database window. The **Help** option is used to view help information. After the user selects a version record, the **Action** pull-down menu is used to select an action for that record. The **Action** menu presents the options: **View**, **Add**, **Edit**, and **Delete**. The **Add** option ignores the selected version record.

If the **Delete** option is selected, and no FAIL.DBF, EC.DBF, or VERIFY.DBF records are found with matching VER_NO fields, a dialogue box is displayed inquiring if the user wishes to delete the record or cancel the deletion. If any records in the FAIL.DBF, EC.DBF, or VERIFY.DBF exist with a matching VER_NO field, no deletion is allowed and a dialogue box is displayed indicating the error and the FAIL_NO, EC_NO, or VF_NO values of the first matching record.

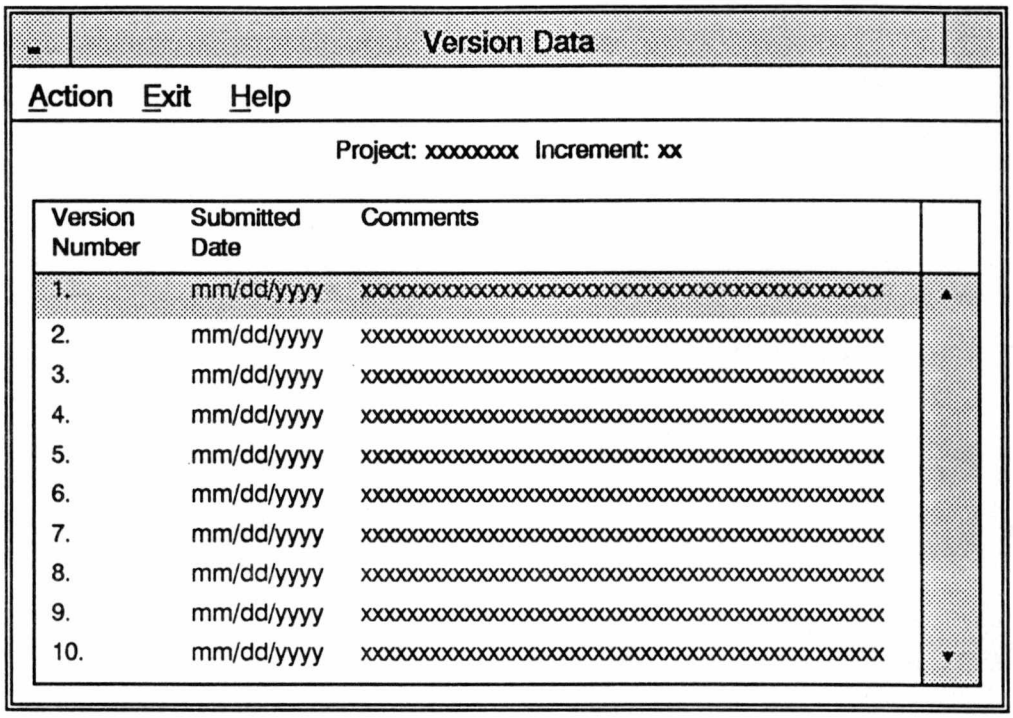


Figure 7.11 Version Data Window

The **View**, **Edit**, and **Add** options present the (View/Edit/Add) Version window (shown in Figure 7.12). (This window is also presented from the Database window if the **Add Data** option is set and the Versions Data is selected.) The title bar is to contain the appropriate title View, Edit, or Add for the option selected.

The user may not change the Version Number field. If the **Add** option is chosen (or if the **Add Data** action is set in the Database window), the Version Number is automatically selected by adding one to the maximum Version Number already existing. (If an intermediate version record has been deleted, the highest version number is still used to select the next version number.) The Submit Date and Comments fields are displayed blank for the **Add** option and may be updated.

The **Edit** and **View** options populate the version record fields before displaying. The user may update the Submit Date and Comment fields if the option is **Edit**. No updates are allowed in the **View** option. The only reason to select the **View** option is to view all of the Comments fields.

The **Action** pull-down menu is not available in the **View** selection. (An **Exit** option is used instead of **Action**.) The **Action** pull-down menu displays three options for the **Edit** and **Add** selections. The Edit Version Data window contains the actions: **Save Changes**, **Abandon Update**, and **Exit**. The Add Version Data window contains the actions: **Add Version**, **Abandon Add**, and **Exit**. The **Help** option is used to display help information.

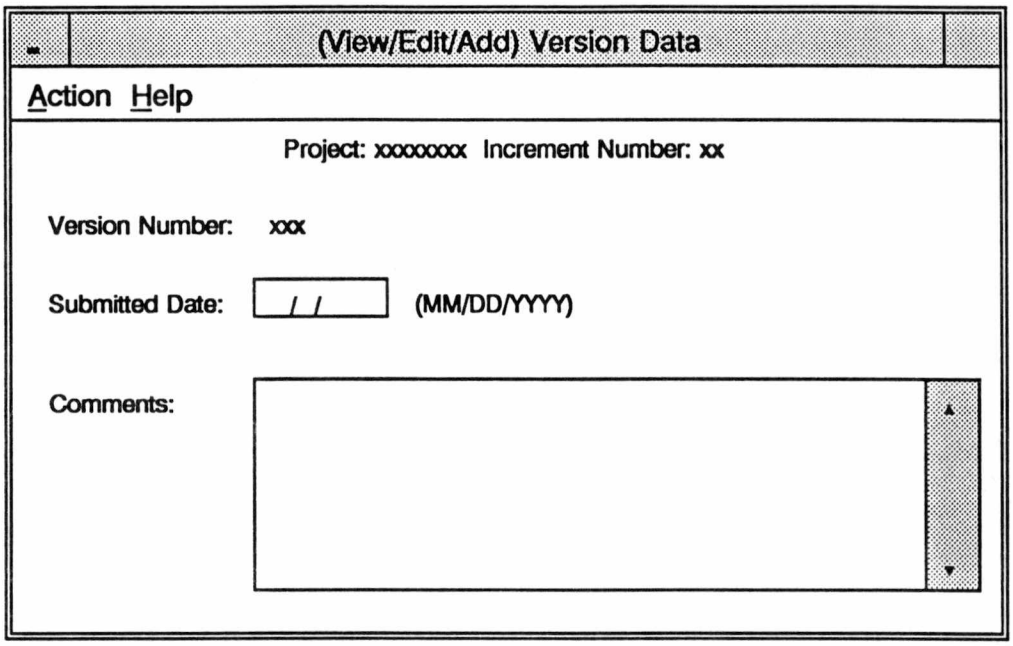


Figure 7.12 (View/Edit/Add) Version Data Window

If either the **Save Changes** or **Add Version** option is selected, entries in the fields are validated. If no errors are found, the version record is changed or added accordingly. If no fields are changed, or an error is found, a dialogue box is displayed with a message inquiring if the user wishes to exit or resume. If either the **Abandon Changes** or **Abandon Add** option is selected, any changes to fields are restored to their original values or blanks. The **Exit** option is used to return to the previous window. If changes were made before the **Exit** option was selected, a dialogue box indicates that changes were made and inquires if the user wishes to exit without saving.

Version Field Validations

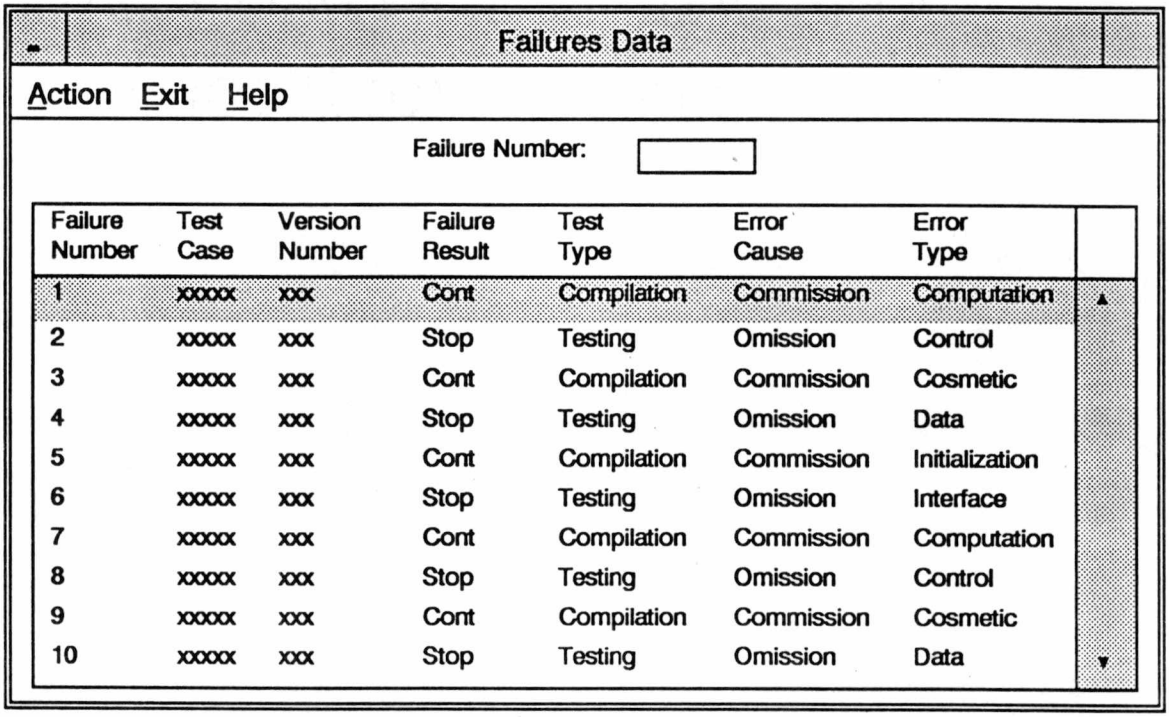
The Version Number field (VER_NO) cannot be changed.

The Version Submit Date field (SUB_DT) must be in MM/DD/YYYY format. The date cannot be less than the PRJ_STD on the PROJ.DBF record. If the PRJ_FND date is populated on the PROJ.DBF record, the Version Submit Date cannot be greater than the project finish date. The date should be checked for a reasonable value. It should not be greater than or less than five years from the current date. The Version Submit Date is a required field.

The Comments field (COMMENTS) is not required. It may contain any information defined by the user. This field is a dBase memo field. A memo field may contain up to 65,535 characters; however, a more reasonable number of characters may be considered for performance issues. The user may scroll the Comments Field by clicking on the arrow symbols.

Failures

If the action control is set to **Select Data**, and the user selects the **Failures** data, the Failures Data window (shown in Figure 7.13) is presented. The Failures Data window lists ten records in the FAIL.DBF table. The list of records may be scrolled by clicking on the arrow symbols at the top and bottom of the list. Only the first seven fields of the VERSION.DBF records are displayed for each record. The user may select a failure by highlighting the associated failure line. The user may type the Failure Number in **Failure Number**, and the matching FAIL.DBF record is positioned as the first line in the display. The user may use the mouse or arrow keys to move the display and highlight different lines. The **Exit** option is used to return to the Database window. The **Help** option is used to view help information.



Failure Number	Test Case	Version Number	Failure Result	Test Type	Error Cause	Error Type
1	xxxxx	xxx	Cont	Compilation	Commission	Computation
2	xxxxx	xxx	Stop	Testing	Omission	Control
3	xxxxx	xxx	Cont	Compilation	Commission	Cosmetic
4	xxxxx	xxx	Stop	Testing	Omission	Data
5	xxxxx	xxx	Cont	Compilation	Commission	Initialization
6	xxxxx	xxx	Stop	Testing	Omission	Interface
7	xxxxx	xxx	Cont	Compilation	Commission	Computation
8	xxxxx	xxx	Stop	Testing	Omission	Control
9	xxxxx	xxx	Cont	Compilation	Commission	Cosmetic
10	xxxxx	xxx	Stop	Testing	Omission	Data

Figure 7.13 Failures Data Window

After the user selects a failure record, the **Action** pull-down menu is used to select an action for that record. The **Action** menu presents the options: **View**, **Add**, **Edit**, and **Delete**. The **Add** option ignores the selected version record, but allows the user to add a new failure.

If the **Delete** option is selected, a dialogue box is displayed inquiring if the user wishes to delete the failure or cancel the deletion. If a deletion is performed, all records in the FAIL.DBF table with FAIL_NO greater than the deleted FAIL_NO must be decreased by one (to keep the Failure Numbers consecutive).

The **View**, **Edit**, and **Add** options present the (View/Edit/Add) Failure window shown in Figure 7.14. (This window is also presented from the Database window if the **Add Data** option is set and the Failures Data is selected.) The title bar is to contain the appropriate title View, Edit, or Add for the option selected.

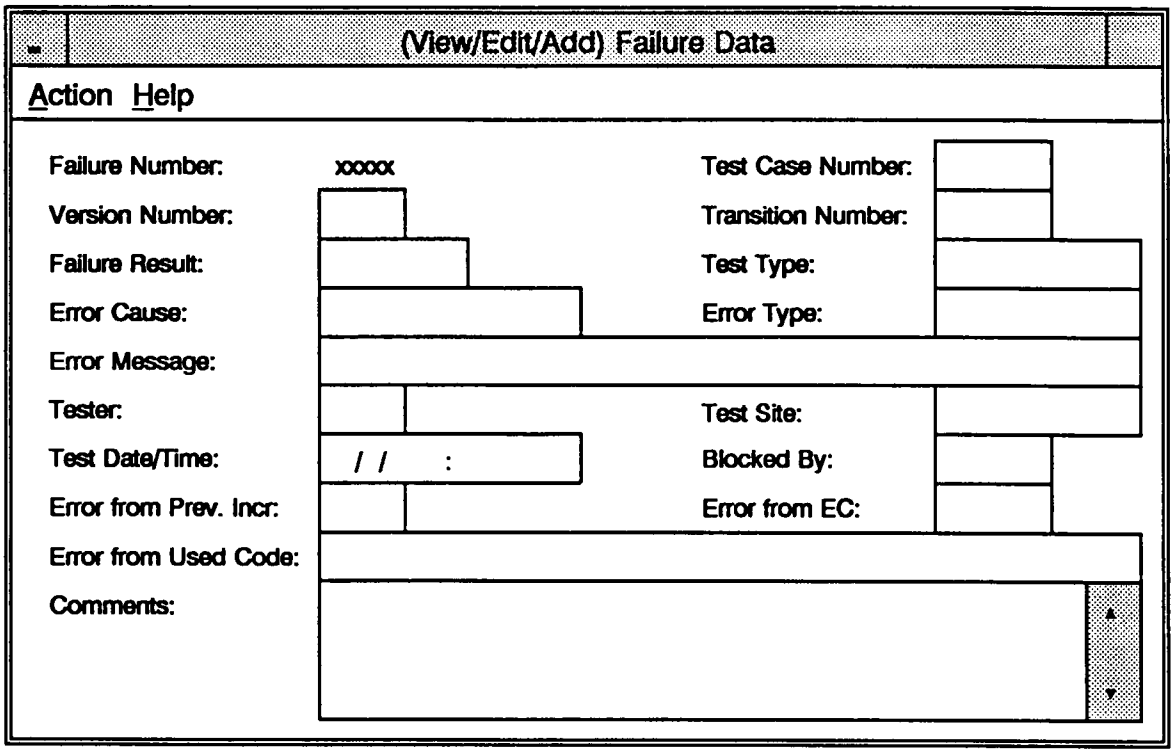


Figure 7.14 (View/Edit/Add) Failure Data Window

The user may not change the Failure Number field. If the **Add** option is chosen (or if the **Add Data** action is set in the Database window), the Failure Number is automatically selected by adding one to the maximum Failure Number already existing. All other fields are displayed with blanks for the **Add** option and may be updated.

The **Edit** and **View** options populate the failure record fields before displaying. If the option is **Edit**, the user may update any of the displayed fields except the Failure Number. No updates are allowed in the **View** option.

The **Action** pull-down menu is not available in the **View** selection. (An **Exit** option is used instead of **Action**.) The **Action** pull-down menu displays three options for the **Edit** and **Add** selections. The Edit Failure Data window contains the actions: **Save Changes**, **Abandon Update**, and **Exit**. The Add Failure Data window contains the actions: **Add Failure**, **Abandon Add**, and **Exit**. The **Help** option is used to display help information.

If the **Save Changes** or **Add Failure** option is selected, entries in the fields are validated. If no errors are found, the version record is changed or added accordingly. If no fields

are changed, or an error is found, a dialogue box is displayed with a message inquiring if the user wishes to exit or resume. If either the **Abandon Changes** or **Abandon Add** option is selected, any changes to fields are restored to their original values or blanks. The **Exit** option is used to return to the previous window (either the Failure Data window or the Database window). If changes were made before the **Exit** option was selected, a dialogue box indicates that changes were made and inquires if the user wishes to exit without saving.

Failure Field Validations

The Failure Number field (FAIL_NO) cannot be changed.

The Test Case Number field (TEST_CASE) should be populated with a matching Test Case Number from the SEQ file. This field is a required field and must be numeric. No other system validations are to be performed.

The Version Number field (VER_NO) must match an existing VER_NO on a record in the VERSION.DBF.

The Transition Number field (TRAN_NO) should be populated with a Transition Number on the SEQ file. The Transition Number on the SEQ file should be listed under the associated Test Case Number in the SEQ file. This field is a required field and must be numeric. No other system validations are to be performed.

The Failure Result field (F_RESULT) must be CONT or STOP. It is a required field. Validation of this field may use an upper character function to convert any lower-case input characters to upper case.

The Test Type field (T_TYPE) must be COMPILATION or TESTING. This field is a required field. The validation comparison should not be case sensitive.

The Error Cause field (E_CAUSE) must be COMMISSION or OMISSION. This field is a required field. Validation of this field may utilize the UPPER() function to convert any lower-case input characters to upper case.

The Error Type field (E_TYPE) must be one of the following: 1) COMPUTATION, 2) CONTROL, 3) COSMETIC, 4) DATA, 5) INITIALIZATION, or 6) INTERFACE. This field is a required field. Validation of this field may utilize the UPPER() function to convert any lower-case input characters to upper case. (Note: if the available selections for Error Type are likely to change, a validation table may be used to store the valid Error Type selections. The table could be maintained by the user. A dialogue box may be written providing the user a choice of valid Error Types. The Help information should list the available selections.)

The Error Message field (ERR_MESS) is not a required field. The user may input the error message returned with the failure. No system validation is performed.

The Tester field (TESTER) is not required. The user may enter the user ID of the responsible tester. No system validation is performed.

The Test Site field (SITE) is not a required field. The user may enter the specific site where the failure occurred. No system validation is performed.

The Test Date and Test Time field present the "/" and ":" delimiters on screen.

The Test Date field (TEST_DT) is not required. Test Date must be in MM/DD/YYYY format. The date cannot be less than the PRJ_STD on the PROJ.DBF record. If the PRJ_FND date is populated on the PROJ.DBF record, the Test Date cannot be greater than the project finish date. The date should be checked for a reasonable value. It should not be greater than or less than five years from the current date.

The Test Time field (TEST_TM) is not required. If entered, it must be in HH:MM format (Hours:Minutes). The Hours entry must not be less than one or greater than twenty-four. The Minutes entry must not be less than one or greater than fifty-nine.

The Blocked By field (BLK_BY) is not required. If populated, it must match a FAIL_NO field in the FAIL.DBF. This field is populated to indicate the failure that caused a delay in the appearance of this failure.

The Error From Prev. Incr. field (PREV_INC) is populated with an Increment Number only if the failure resulted from code in a previous increment. This field is not required. If it is populated, the Increment Number must be less than the currently selected Increment Number, but not zero.

The Error from EC field (EC_NO) is populated with an Engineering Change number only if the failure resulted from code introduced by an engineering change in a previous version. This field is not required, and it may not be populated if the PREV_INCR field is populated. If the field is populated, the entry must match the EC_NO field of an existing record in the EC.DBF.

The Error From Used Code field (USED_CD) is populated only if the failure is due to an error in code reused from a source outside the project. This is not a required field, and it may not be populated if either the PREV_INCR or EC_NO field is populated. The field contents are not validated. The user may describe in text format the faulty reused code.

The Comments field (COMMENTS) is not required. It may contain any information defined by the user. This field is a dBase memo field. A memo field may contain up to 65,535 characters; however, a more reasonable number of characters may be considered for performance issues. The user may scroll the Comments Field by clicking on the arrow symbols.

Engineering Changes

If the action control is set to **Select Data**, and the user selects the **Engineering Changes** data, the Engineering Changes Data window (shown in Figure 7.15) is presented. The Engineering Changes Data window lists ten records in the EC.DBF table. The list of records may be scrolled by clicking on the arrow symbols at the top and bottom of the list. All fields in the EC.DBF table, except the Designer field, are displayed for each record. The Comments field is truncated.

The user may select an engineering change by highlighting the associated engineering change line. The user may type the Engineering Change Number in the **EC Number** field, and the matching EC.DBF record is positioned as the first line in the display. The user may use the mouse or arrow keys to move the display and highlight different lines. The **Exit** option is used to return to the Database window. The **Help** option is used to display help information. After the user selects an engineering change record, the **Action** pull-down menu is used to select an action for that record. The **Action** menu provides the options: **View**, **Add**, **Edit**, and **Delete**. The **Add** option ignores the selected record, but allows the user to add an engineering change.

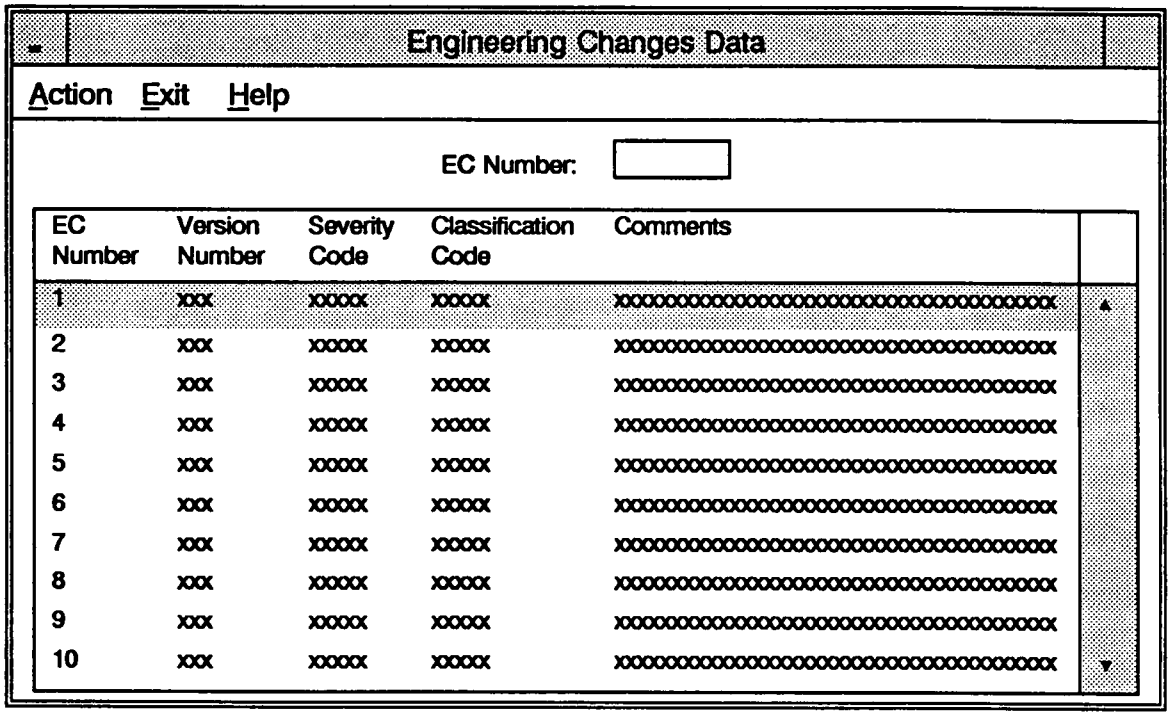


Figure 7.15 Engineering Changes Window

If the **Delete** option is selected, and there are FAIL.DBF records found with matching EC_NO fields, a dialogue box is displayed inquiring if the user wishes to delete the engineering change or cancel the delete. If one or more FAIL.DBF records with a matching EC_NO field exists, no update is allowed, and a dialogue box is displayed with

the error and Failure Number (FAIL_NO) of the first matching FAIL.DBF record. If a deletion is performed, all records in the EC.DBF table with EC_NO greater than the deleted EC_NO must be decreased by one (to keep the Engineering Change Numbers consecutive).

The **View**, **Edit**, and **Add** options present the (View/Edit/Add) Engineering Change window shown in Figure 7.16. (This window is also presented from the Database window if the **Add Data** option is set and the Engineering Changes Data is selected.) The title bar is to contain the appropriate title View, Edit, or Add for the option selected.

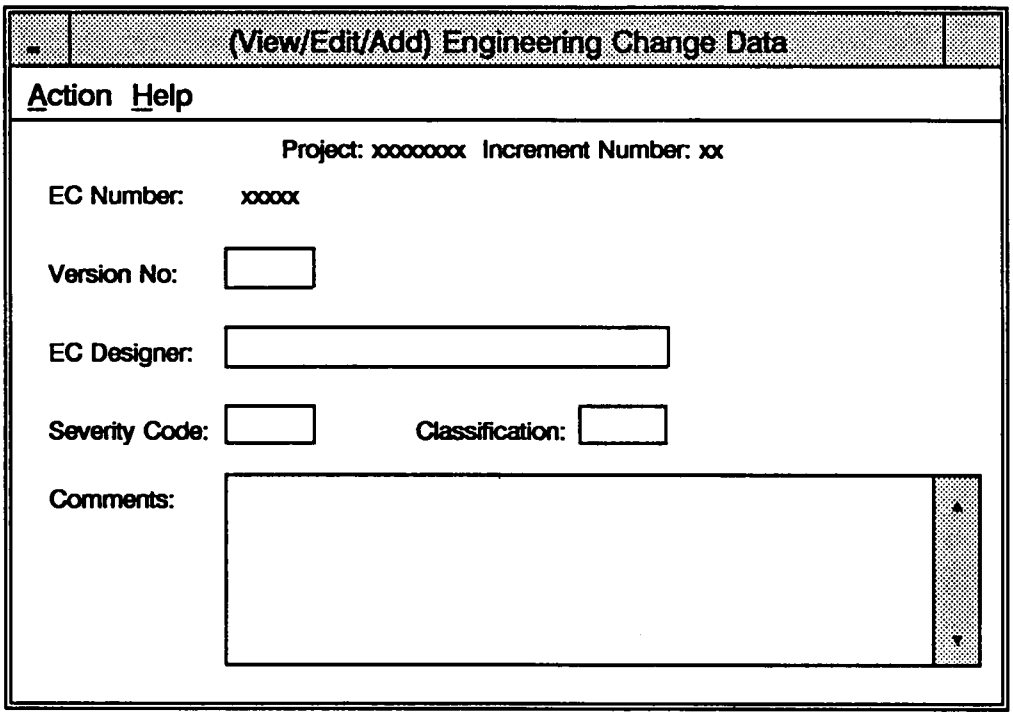


Figure 7.16 (View/Edit/Add) Engineering Change Data Window

The user may not change the EC Number field. If the **Add** option is chosen (or the **Add Data** option is set in the Database window), the EC Number is automatically selected by adding one to the maximum EC_NO in the EC.DBF table. The Version Number is defaulted to the maximum VER_NO in the VERSION.DBF. All other fields are displayed with blanks for the **Add** option and may be updated.

The **Edit** and **View** options populate the Engineering Change record fields before displaying. If the option is **Edit**, the user may update any of the displayed fields (except the EC Number). No updates are allowed in the **View** option.

The **Action** pull-down menu is not available in the **View** selection. (An **Exit** option is used instead of **Action**.) The **Action** pull-down menu displays three options for the **Edit** and **Add** selections. The Edit Engineering Change Data window contains the actions: **Save**

Changes, **Abandon Update**, and **Exit**. The Add Engineering Change Data window contains the actions: **Add Engineering Change**, **Abandon Add**, and **Exit**. The **Help** option is used to display help information.

If the **Save Changes** or **Add Engineering Change** option is selected, entries in the fields are validated. If no errors are found, the EC.DBF record is changed or added accordingly. If no fields are changed, or an error is found, a dialogue box is displayed with the message, and inquiring if the user wishes to exit or resume. If the **Abandon Changes** or **Abandon Add** option is selected, any changes to fields are restored to their original values or blanks. The **Exit** option is used to return to the previous window. If changes were made before the **Exit** option was selected, a dialogue box indicates that changes were made and inquires if the user wishes to exit without saving.

Engineering Change Field Validations

The EC Number field (EC_NO) cannot be changed.

The Version Number (VER_NO) is required. It defaults to the maximum VER_NO in the VERSION.DBF. The user may change the value, but it must exist in the VERSION.DBF table.

The Designer Identification field (DESIGNER) is required. The user must enter the name or other form of identification of the person(s) designing the engineering change.

The Severity Code (SEVERITY) is a user defined code indicating the severity level of the engineering change. This field is not required.

The Classification Code (CLASSIFY) is a user defined code indicating the classification of the engineering change. This field is not required.

The Comments field is not required. It may contain any information defined by the user. This field is a dBase memo field. A memo field may contain up to 65,535 characters; however, a more reasonable number of characters may be considered for performance issues. The user may scroll the Comments Field by clicking on the arrow symbols.

Verification Sessions Data

If the action control is set to **Select Data**, and the user selects the **Verification Sessions** data, the Verification Sessions Data window (shown in Figure 7.17) is presented. The Verification Sessions Data window lists ten records in the VERIFY.DBF table. The list of records may be scrolled by clicking on the arrow symbols at the top and bottom of the list. All fields in the VERIFY.DBF table are displayed for each record. The Comments field is truncated.

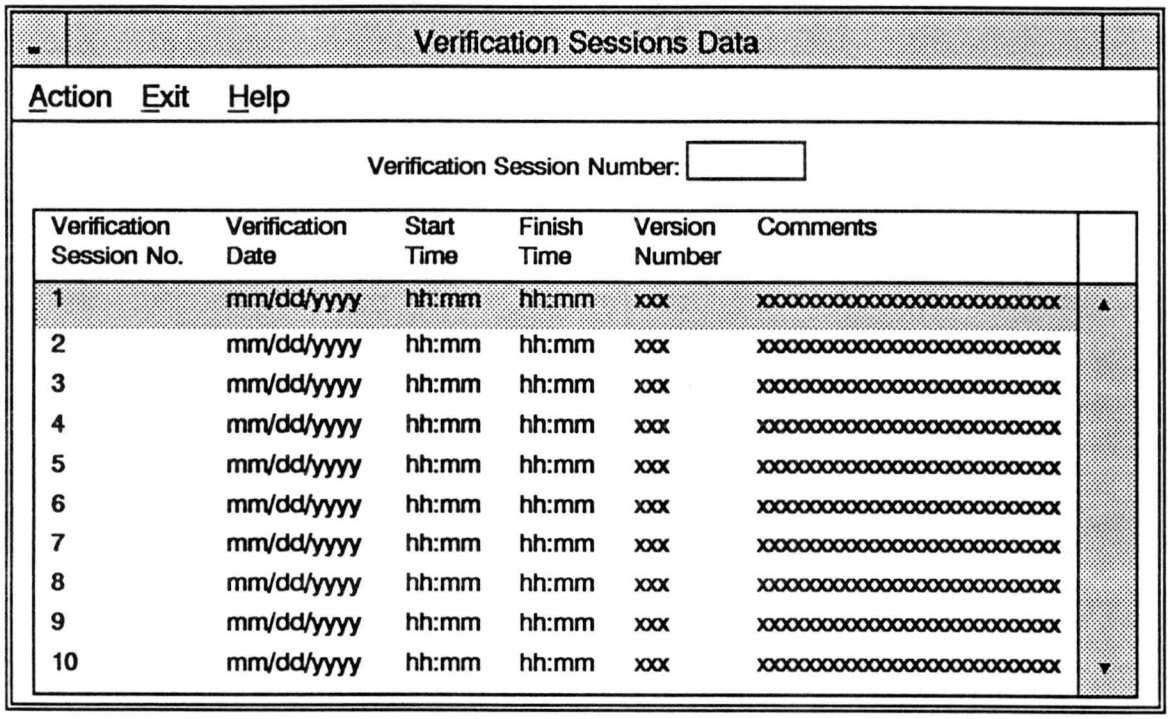


Figure 7.17 Verification Sessions Data Window

The user may select a verification session by highlighting the associated verification session line. The user may type the Verification Session Number in **Verification Session Number**, and the matching VERIFY.DBF record is positioned as the first line in the display. The user may use the mouse or arrow keys to move the display and highlight different lines. The **Exit** option is used to return to the Database window. The **Help** option is used to display help information. After the user selects a verification session record, the **Action** pull-down menu is used to select an action for that record. The **Action** menu presents the options: **View**, **Add**, **Edit**, and **Delete**. The **Add** option ignores the selected record, but allows the user to add a verification session.

If the **Delete** option is selected, and there are no FOUND.DBF or ATTEND.DBF records found with matching VF_NO fields, a dialogue box is displayed inquiring if the user wishes to delete the verification session or cancel the deletion. If either a FOUND.DBF or ATTEND.DBF record exists with a matching VF_NO field, no deletion is allowed, and a dialogue box is displayed indicating the error. If a deletion is performed, all records in the VERIFY.DBF table with VF_NO greater than the deleted VF_NO must be decreased by one (to keep the Verification Session Numbers consecutive). For each VF_NO that is changed, all associated FOUND.DBF and ATTEND.DBF records must be changed so the VF_NO field matches the intended VERIFY.DBF record.

The **View**, **Edit**, and **Add** options present the (View/Edit/Add) Verification Sessions window shown in Figure 7.18. (This window is also presented from the Database window if the **Add Data** option is set and the Verification Sessions Data is selected.) The title bar is to contain the appropriate title View, Edit, or Add for the option selected.

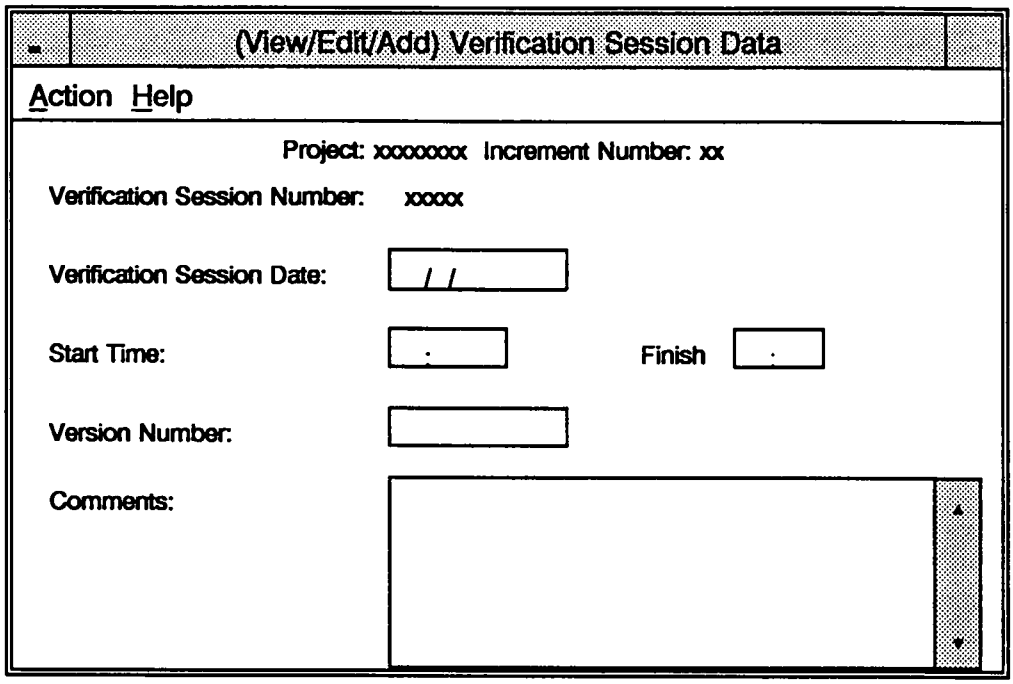


Figure 7.18 (View/Edit/Add) Verification Session Data Window

The user may not change the Verification Session Number field. If the **Add** option is chosen (or the **Add Data** option is set in the Database window), the Verification Session Number is automatically selected by adding one to the maximum VF_NO in the VERIFY.DBF table. The Version Number is defaulted to the maximum VER_NO in the VERSION.DBF. All other fields are displayed with blanks for the **Add** option and may be updated.

The **Edit** and **View** options populate the verification session record fields before displaying. If the option is **Edit**, the user may update any of the displayed fields (except the Verification Session Number). No updates are allowed in the **View** option.

The **Action** pull-down menu is not available in the **View** selection. (An **Exit** option is used instead of **Action**.) The **Action** pull-down menu displays three options for the **Edit** and **Add** selections. The Edit Verification Session Data window contains the actions: **Save Changes**, **Abandon Update**, and **Exit**. The Add Verification Session Data window contains the actions: **Add Verification Session**, **Abandon Add**, and **Exit**. The **Help** option is used to display help information. If the **Save Changes** or **Add Verification Session** option is selected, entries in the fields are validated. If no errors are found, the VERIFY.DBF record is changed or added accordingly. If no fields are changed, or an error is found, a dialogue box is displayed with a message inquiring if the user wishes to exit or resume. If the **Abandon Changes** or **Abandon Add** option is selected, any changes to fields are restored to their original values or blanks. The **Exit** option is used to return to the previous window. If changes were made before the **Exit** option was selected, a dialogue box indicates that changes were made and inquires if the user wishes to exit without saving.

Verification Sessions Field Validations

The Verification Session Number field (VF_NO) cannot be changed.

The Verification Session Date field (VF_DT) must be in MM/DD/YYYY format. This field is required. The date cannot be less than the VF_DT of the preceding VERIFY.DBF record's VF_DT field. It cannot be greater than the VF_DT of the next VERIFY.DBF record's VF_DT field. If the VF_NO equals one, the date cannot be less than the PRJ_STD field on the PROJ.DBF record, and if the PRJ_FND date is populated on the PROJ.DBF record, the VF_DT cannot be greater than the project finish date. The date should be checked for a reasonable value. It should not be greater than or less than five years from the current date.

The Start Time field (VF_ST_TM) is required. It must be in HH:MM format (Hours:Minutes). The Hours entry must not be less than one or greater than twenty-four. The Minutes entry must not be less than one or greater than fifty-nine. The colon between the hours and minutes is displayed in the input window.

The Finish Time field (VF_FN_TM) is required. It must be in HH:MM format (Hours:Minutes). The Hours entry must not be less than one, or greater than 24. The Minutes entry must not be less than one, or greater than fifty-nine. The Finish Time cannot be less than the Start Time.

The Version Number field (VER_NO) must exist in the VERSION.DBF table. It is defaulted to the maximum VER_NO in the VERSION.DBF. The user may change the value, but a valid Version Number is required.

The Comments field (COMMENTS) is not required. It may contain any information defined by the user. This field is a dBase memo field. A memo field may contain up to 65,535 characters; however, a more reasonable number of characters may be considered for performance issues. The user may scroll the Comments Field by clicking on the arrow symbols.

Verification Errors Found

If the action control is set to **Select Data**, and the user selects the **Verification Errors Found** data, the Verification Errors Found Data window (shown in Figure 7.19) is presented. The window lists ten records in the FOUND.DBF table. The list of records may be scrolled by clicking on the arrow symbols at the top and bottom of the list. All fields, except the Comments, are displayed for each record.

The user may select a verification error found by highlighting the associated line. The user may type the Verification Session Number in the **Verification Session Number** field, and the first FOUND.DBF record matching the entered Verification Session Number (VF_NO) is positioned as the first line in the display. The user may use the mouse or arrow keys to move the display and highlight different lines. The **Exit** option is used to return to the Database window. The **Help** option is used to display help information.

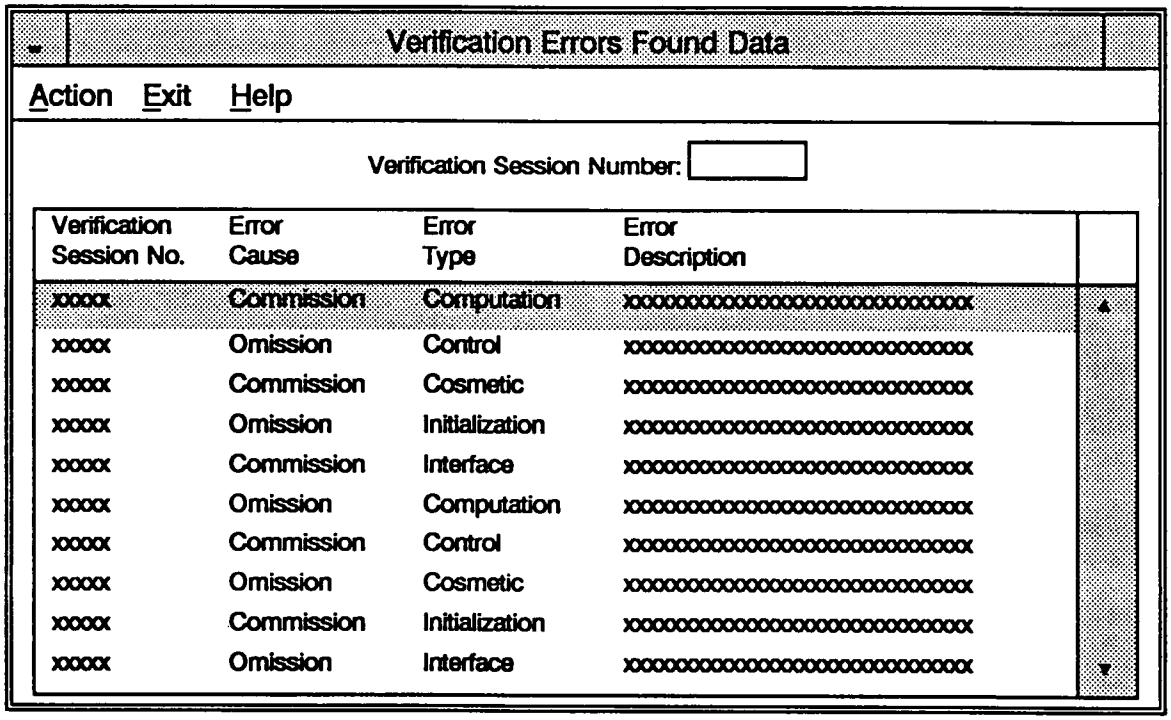


Figure 7.19 Verification Errors Found Data Window

After the user selects a verification error found record, the **Action** pull-down menu is used to select an action for that record. The **Action** menu presents the options: **View**, **Add**, **Edit**, and **Delete**. The **Add** option ignores the selected record, but allows the user to add a found error.

If the **Delete** option is selected, a dialogue box is displayed inquiring if the user wishes to delete the found error or cancel the deletion. If a deletion is performed, the FOUND.DBF is deleted. No other deletions or changes are required.

The **View**, **Edit**, and **Add** options present the (View/Edit/Add) Verification Error Found window shown in Figure 7.20. (This window is also presented from the Database window if the **Add Data** option is set and the Verification Errors Found Data is selected.) The title bar is to contain the appropriate title View, Edit, or Add for the option selected.

If the **Add** option is chosen (or the **Add Data** option is set in the Database window), all fields are displayed with blanks for the **Add** option and may be updated. The **Edit** and **View** options populate the verification errors found record fields before displaying. If the option is **Edit**, the user may update any of the displayed fields.

The **Action** pull-down menu is not available in the **View** selection. (An **Exit** option is used instead of **Action**.) The **Action** pull-down menu displays three options for the **Edit** and **Add** selections. The Edit Verification Error Found window contains the actions: **Save Changes**, **Abandon Update**, and **Exit**. The Add Verification Error Found window contains the actions: **Add Verification Error Found**, **Abandon Add**, and **Exit**. The **Help** option is used to display help information.

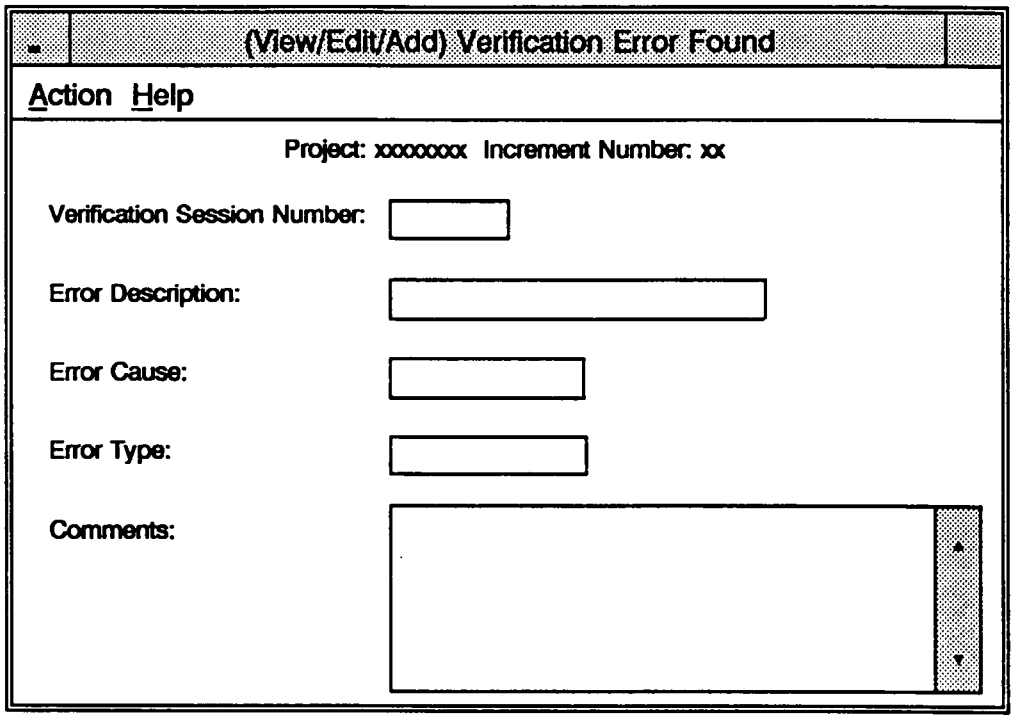


Figure 7.20 (View/Edit/Add) Verification Errors Found Window

If either the **Save Changes** or **Add Verification Error Found** option is selected, entries in the fields are validated. If no errors are found, the FOUND.DBF record is changed or added accordingly. If no fields are changed, or an error is found, a dialogue box is displayed with the message inquiring if the user wishes to exit or resume. If the **Abandon Changes** or **Abandon Add** option is selected, any changes to fields are restored to their original values or blanks. The **Exit** option is used to return to the previous window. If changes were made before the **Exit** option was selected, a dialogue box indicates that changes were made and inquires if the user wishes to exit without saving.

Verification Errors Found Field Validations

The Verification Session Number field (VF_NO) must match the VF_NO in an existing record in the VERIFY.DBF.

The Error Description field (ERR_DESC) contains a short text describing the error. Entry is required.

The Error Cause field (E_CAUSE) must be COMMISSION or OMISSION. This field is required. Validation of this field should convert any lower-case input characters to upper case.

The Error Type field (E_TYPE) must be one of the following: 1) COMPUTATION, 2) CONTROL, 3) COSMETIC, 4) DATA, 5) INITIALIZATION, or 6) INTERFACE. This field is a required field. Validation of this field should convert any lower-case input characters

to upper case. (Note: if the available selections for Error Type are likely to change, a validation table may be used to store the valid Error Type selections. The table could be maintained by the user. A dialogue box may be written providing the user a choice of valid Error Types. The Help information should list the available selections.)

The Comments field (COMMENTS) is not required. It may contain any information defined by the user. This field is a dBase memo field. A memo field may contain up to 65,535 characters; however, a more reasonable number of characters may be considered for performance issues. The user may scroll the Comments Field by clicking on the arrow symbols.

Verification Attendance

If the action control is set to **Select Data**, and the user selects the **Verification Attendance** data, the Verification Attendance Data window (shown in Figure 7.21) is presented. The window lists ten records in the ATTEND.DBF table.

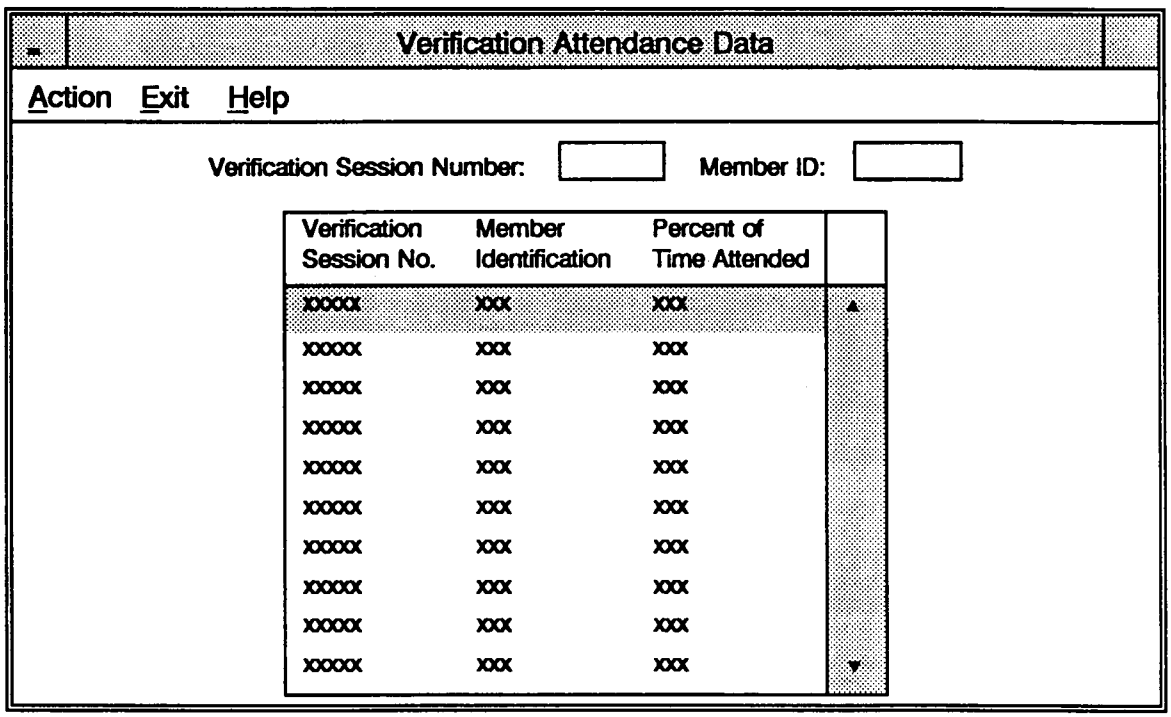


Figure 7.21 Verification Attendance Data Window

Initially, the records are listed in VF_NO/MEMBER order. The list of records may be scrolled by clicking on the arrow symbols at the top and bottom of the list. All fields are displayed for each record. The user may select a verification attendance record by highlighting the associated line. The user may type the Verification Session Number in the **Verification Session Number** field to position the first ATTEND.DBF record matching

the entered Verification Session Number (VF_NO) as the first line in the display. If the user types the Member ID in **MEMBER ID**, and presses enter, the list is ordered by MEMBER/VF_NO, and the first ATTEND.DBF record found matching the MEMBER is positioned as the first line in the list. The user may use the mouse or arrow keys to move the display and highlight different lines. The **Exit** option is used to return to the Database window. The **Help** option is used to display help information. After the user selects a verification attendance record, the **Action** pull-down menu is used to select an action for that record. The **Action** menu presents the options: **Add**, **Edit**, and **Delete**. The **Add** option ignores the selected record, but allows the user to add a verification attendance record.

If the **Delete** option is selected, a dialogue box is displayed inquiring if the user wishes to delete the attendance record or cancel the deletion. If a deletion is performed, the ATTEND.DBF is deleted. No other deletions or changes are required.

The **Edit** and **Add** options present the (Edit/Add) Verification Attendance window shown in Figure 7.22. (This window is also presented from the Database window if the **Add Data** option is set and the Verification Attendance Data is selected.) The view option is not needed since all fields are displayed in the Verification Attendance Data window. The title bar is to contain Edit, or Add, according to the option selected.

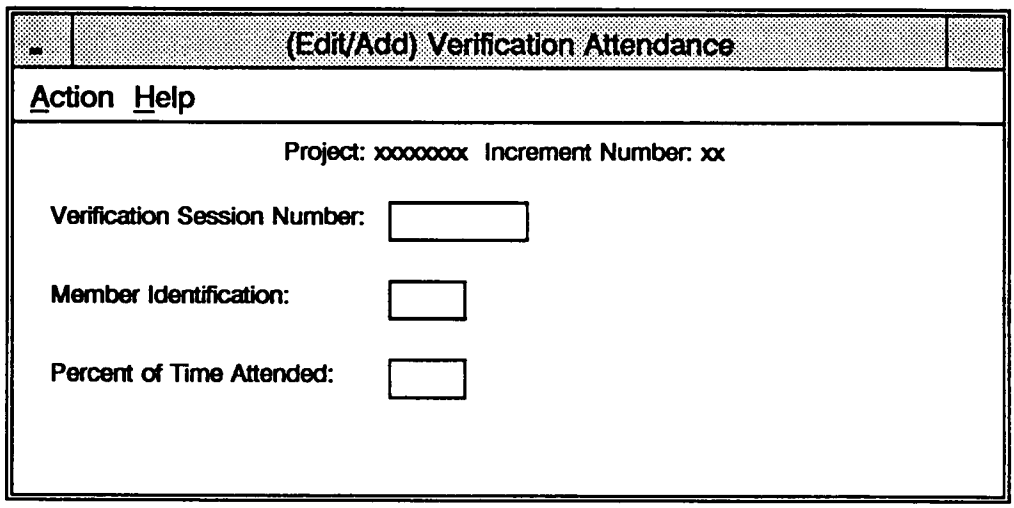


Figure 7.22 (Edit/Add) Verification Attendance Window

If the **Add** option is chosen (or the **Add Data** option is set in the Database window), all fields are displayed with blanks for the **Add** option and may be updated.

The **Edit** option populates the verification attendance record fields before displaying. The user may update any of the displayed fields.

The **Action** pull-down menu displays three options for the **Edit** and **Add** selections. The Edit Verification Attendance window contains the actions: **Save Changes**, **Abandon**

Update, and **Exit**. The Add Verification Attendance window contains the actions: **Add Verification Attendance**, **Abandon Add**, and **Exit**. The **Help** option is used to display help information.

If either the **Save Changes** or **Add Verification Attendance** option is selected, entries in the fields are validated. If no errors are found, the ATTEND.DBF record is changed or added accordingly. If no fields are changed, or an error is found, a dialogue box is displayed with the message inquiring if the user wishes to exit or resume. If the **Abandon Changes** or **Abandon Add** option is selected, any changes to fields are restored to their original values or blanks. The **Exit** option is used to return to the previous window. If changes were made before the **Exit** option was selected, a dialogue box indicates that changes were made and inquires if the user wishes to exit without saving.

Verification Attendance Field Validations

The Verification Session Number field (VF_NO) must match the VF_NO in an existing record in the VERIFY.DBF.

The Member Identification field (MEMBER) is required. The field contains the three character member identification.

The Percent of Time Attended field (PER_TIME) is not required. If no entry is input for this field, a default of 100 is used. If entry is made, the field must be numeric, greater than zero, and no greater than 100. The number must be an integer.

Hours Worked

If the action control is set to **Select Data**, and the user selects the **Hours Worked Data**, the Hours Worked Data window (shown in Figure 7.23) is presented. The window lists ten records in the HOURS.DBF table. Initially, the records are listed in WE_DATE/MEMBER order. The user may select a member's week of hours by highlighting the associated line. The list of records may be scrolled by clicking on the arrow symbols at the top and bottom of the list. All fields are displayed for each record, except Miscellaneous Hours.

If the user enters the Week Ending Date in **Week Ending Date** field and presses enter, the list is ordered by WE_DATE/MEMBER, and the first ATTEND.DBF record found matching the MEMBER is positioned as the first line in the list. If the user types the Member's Identification in **Member ID** and presses enter, the list is ordered by MEMBER/WE_DATE, and the first HOURS.DBF record matching the entered Member ID (MEMBER) is positioned in the first line. If the **Week Ending Date** is entered and, using a tab, the user enters **MEMBER ID** before pressing enter, the list is presented in MEMBER/WE_DATE order with the HOURS.DBF record matching the MEMBER and WE_DATE fields positioned as the first line.

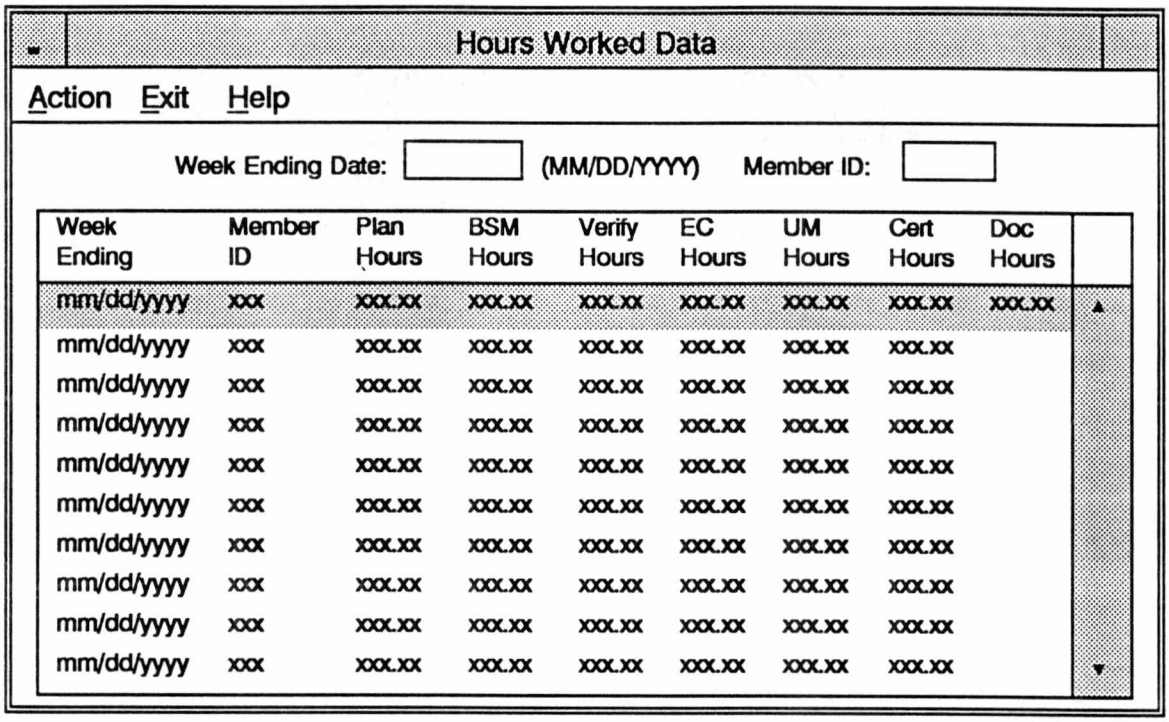


Figure 7.23 Hours Worked Data Window

The user may use the mouse or arrow keys to display additional lines and highlight different lines. The **Exit** option is used to return to the Database window. The **Help** option is used to display help information. After the user selects a member's hours worked record, the **Action** pull-down menu is used to select an action for that record. The **Action** menu presents the options: **View**, **Add**, **Edit**, and **Delete**. The **Add** option ignores the selected record, but allows the user to add a member's hours record.

If the **Delete** option is selected, a dialogue box is displayed inquiring if the user wishes to delete the hours worked record or cancel the deletion. If a deletion is performed, the HOURS.DBF is deleted. No other deletions or changes are required.

The **View, Edit**, and **Add** options present the (View/Edit/Add) Hours Worked window shown in Figure 7.24. (This window is also presented from the Database window if the **Add Data** option is set and the Hours Worked Data is selected.) The title bar is to contain the appropriate title (View, Edit or Add) for the option selected.

If the **Add** option is chosen (or the **Add Data** option is set in the Database window), all fields are displayed with blanks for the **Add** option and may be updated.

The **Edit** and **View** options populate the hours worked record fields before displaying. The user may update any of the displayed fields in the edit mode. No updates are allowed in the view mode.

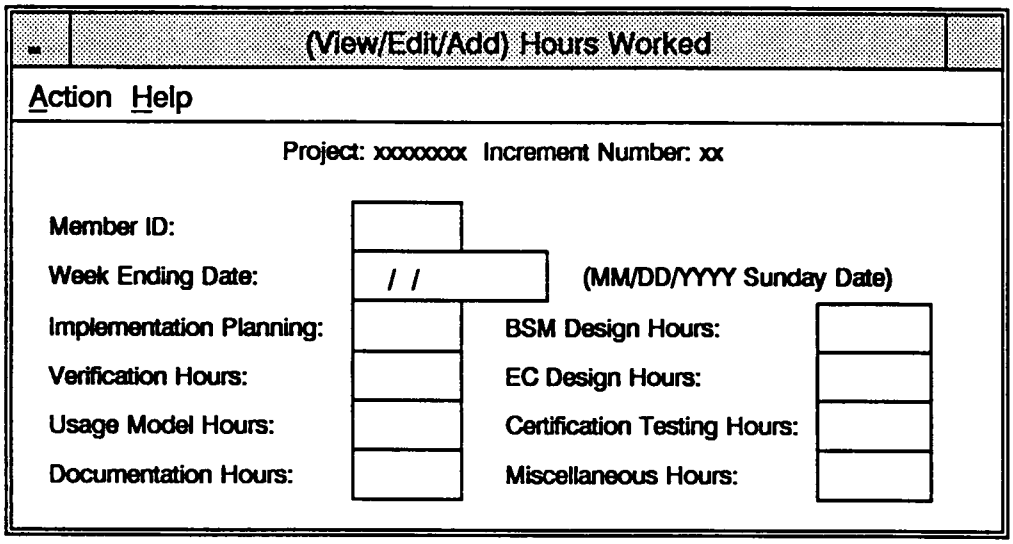


Figure 7.24 (Edit/Add) Hours Works Window

The **Action** pull-down menu displays three options for the **Edit** and **Add** selections. The Edit Hours Worked window contains the actions: **Save Changes**, **Abandon Update**, and **Exit**. The Add Hours Worked window contains the actions: **Add Hours Worked**, **Abandon Add**, and **Exit**. The **Help** option is used to display help information.

If either the **Save Changes** or **Add Hours Worked** option is selected, entries in the fields are validated. If no errors are found, the HOURS.DBF record is changed or added accordingly. If no fields are changed, or an error is found, a dialogue box is displayed with the message inquiring if the user wishes to exit or resume. If the **Abandon Changes** or **Abandon Add** option is selected, any changes to fields are restored to their original values or blanks. The **Exit** option is used to return to the previous window (Hours Worked Data or Database window). If changes were made before the **Exit** option was selected, a dialogue box indicates that changes were made and inquires if the user wishes to exit without saving.

Hours Worked Field Validations

The Member ID field (MEMBER) is required. The field is user defined. The field contains a three character ID for the member.

The Week Ending Date field (WE_DATE) is required. It must be a valid Sunday date. Work weeks end on Sunday and begin on Monday. The native dBase language contains a function that returns the day of the week for a given date. That function (CDOW) is not available with QELIB; however, QELIB provides the ability to output the first three initials of the day of the week. The WE_DATE must be a reasonable date. It cannot be less than or greater than five years from the current date.

No two records may exist in the HOURS.DBF table with the same MEMBER and WE_DATE values.

Entry is not required for the hours fields: Implementation Planning (IP_HRS), BSM Design Hours (BSM_HRS), (VF_HRS), EC Design Hours (EC_HRS), Usage Model Hours (UM_HRS), Certification Testing Hours (CT_HRS), Documentation Hours (DOC_HRS), and Miscellaneous Hours (MISC_HRS). Any number of these entries may be populated. All entries must be numeric, and their totals cannot be greater than 168 (7 multiplied by 24). No entry defaults the field to zero.

Counts

The Counts data is contained in one record in the COUNT.DBF table. The nature of this data does not require more than one record per increment. The original copy of this table in the project/increment directory contains a blank record. The Counts selection is not available in the Database window when the action control is set to **Add Data**. When the action control is set to **Select Data**, and the user selects the **Counts** data, the Count Data window (shown in Figure 7.25) is presented. This window displays the current values of all counts for view (not updatable). The Counts Data window is populated with fields from the COUNT.DBF table record. The fields are populated on initial presentation of the window.

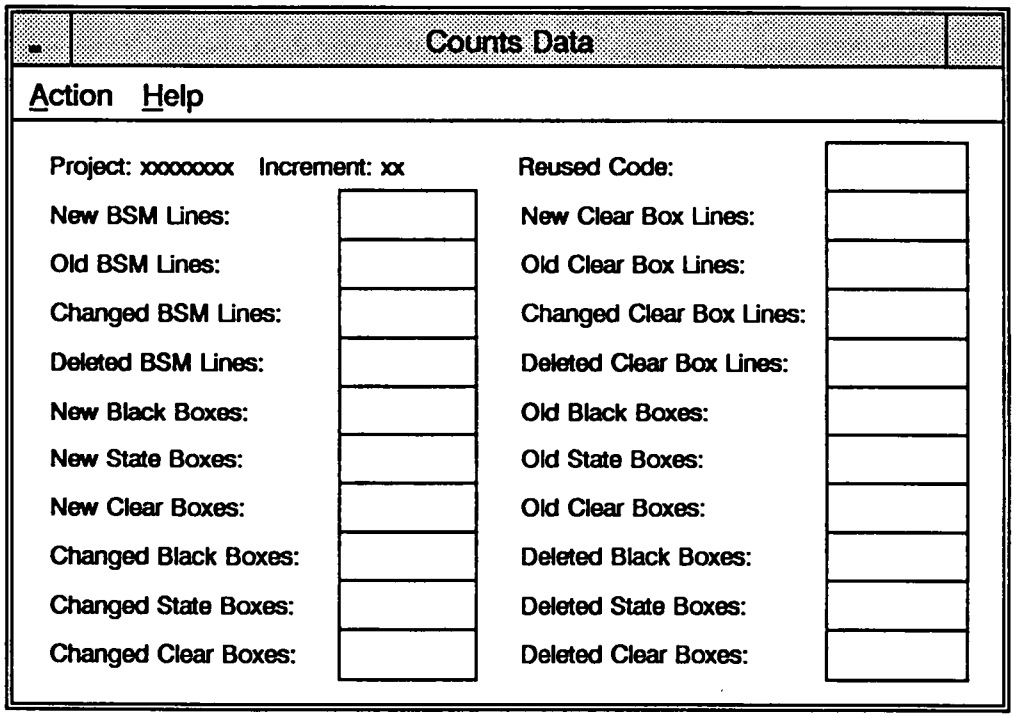


Figure 7.25 Counts Data

The **Help** option is used to display help information. The **Action** pull-down menu is used to allow update of the fields.

The **Action** pull-down menu displays four options: **Edit**, **Save Changes**, **Abandon Update**, and **Exit**. If the **Edit** option is selected, all table fields displayed are changed to allow update. If the **Save Changes** option is selected, entries in the fields are validated. If no errors are found, the COUNT.DBF record is updated. If no fields are changed, or an error is found, a dialogue box is displayed with the message inquiring if the user wishes to exit or resume. If the **Abandon Changes** option is selected, any changes to fields are restored to their original values. The **Exit** option is used to return to the Database window. If changes were made before the **Exit** option was selected (and not saved), a dialogue box indicates that changes were made and inquires if the user wishes to exit without saving.

Counts Field Validations

None of the fields is required. The only validations are that each input field must be numeric, positive integers.

The Reused Code field (USED_CD) contains the number of lines in code used that is written outside the project.

The New BSM Lines field (ND_LINES) contains the total number of lines in the new black box, state box, and clear Boxes.

The New Clear Box Lines field (NS_LINES) contains the total number of lines in the new clear boxes.

The Old BSM Lines field (OD_LINES) contains the total number of lines in the old black boxes, state boxes, and clear boxes.

The New Clear Box Lines field (OS_LINES) contains the total number of lines in the new clear boxes.

The Changed BSM Lines field (CD_LINES) contains the total number of lines in the changed black boxes, state boxes, and clear boxes.

The Changed Clear Box Lines field (CS_LINES) contains the total number of lines in the changed clear boxes.

The Deleted BSM Lines field (DD_LINES) contains the total number of lines in the deleted black boxes, state boxes, and clear boxes.

The Deleted Clear Box Lines field (DS_LINES) contains the total number of lines in the deleted clear boxes.

The New Black Boxes field (BB_NEW) contains the total number of new black boxes.

The New State Boxes field (SB_NEW) contains the total number of new state boxes.

The New Clear Boxes field (CB_NEW) contains the total number of new clear boxes.

The Old Black Boxes field (BB_OLD) contains the total number of old black boxes.

The Old State Boxes field (SB_OLD) contains the total number of old state boxes.

The Old Clear Boxes field (CB_OLD) contains the total number of old clear boxes.

The Changed Black Boxes field (BB_CHG) contains the total number of changed black boxes.

The Changed State Boxes field (SB_CHG) contains the total number of changed state boxes.

The Changed Clear Boxes field (CB_CHG) contains the total number of changed clear boxes.

The Deleted Black Boxes field (BB_DEL) contains the total number of deleted black boxes.

The Deleted State Boxes field (SB_DEL) contains the total number of deleted state boxes.

The Deleted Clear Boxes field (CB_DEL) contains the total number of deleted clear boxes.

Totals Table Update

The TOTALS.DBF table is designed to include only one record. The record contains nine fields for data that is normally reported with any CRE increment metrics. Unlike the fields in the COUNT.DBF table, these fields can be system generated from the CDLP subsystem and UME subsystems. The TOTALS.DBF table does not need to be accessed by the DBS subsystem. The table is to be used for reporting purposes only.

CDLP Updates

The CDLP subsystem is to be used to update the BSM lines and box counts. These system totals represent counts for the entire increment's design. No distinction is made between new, old, or changed design boxes or lines. Data elements to be populated include: total design language lines (line counts for all boxes), total source code lines (all line counts for clear boxes), and total counts for all boxes divided into black boxes, state boxes, and clear boxes.

Program CDLP.CDL, which is described in the CRE increment five documentation, contains the function calls to read and process the CDL lines. Black boxes are determined by the "define_BB" and "end_BB" text that mark the beginning and ending of

the boxes. The state boxes and clear boxes have similar text to mark the beginning and ending of the boxes. Function calls are used to read the .CDL text and validate the input. A line count is accumulated during processing. The CDLP.CDL program can include separate counts for black box, state box, and clear box totals.

The CDL lines count (TOT_DL) is populated with the total line count for all boxes. The C lines count (TOT_SL) is to be populated with the line count for clear boxes only. The number of black boxes (TOT_BB) is to be populated with the count of separate "define_BB" texts. The number of state boxes (TOT_SB) is to be populated with the count of separate "define_SB" texts. The number of clear boxes (TOT_CB) is to be populated with the count of separate "define_CB" texts. Table update is to be performed at the end of the CDLP.CDL program. If errors are detected in the processing of the CDL file, no update is to be performed.

The TOTALS.DBF table is originally stored in the project/increment directory with one blank record. No record creation is required. Each time the CDLP.CDL program executes successfully, the record is to be updated with the new values.

A future increment may include update of another table containing one record for each box processed. The table could contain individual line counts for each box, along with other pertinent information about the box. The number of records in the table would fluctuate with the number of boxes in the .CDL file.

Usage Model Editor Updates

The UME subsystem is to be used to update the total number of states in the usage model, total number of arcs, number of non-zero matrix entries, and the percent of non-zero matrix entries.

Program UME.CDL, which is described in the CRE increments six and eight documentation, contains a function call (UmeReadMatrix) to read the FTF matrix file. The first line in the FTF file contains the number of states in the file. This value may be used directly to populate the number of states in the usage model (TOT_US). The black box "UmeNormalize" normalizes the FTF usage model file into a matrix. This process is to calculate the number of non-zero and total number of entries in the matrix. The number of non-zero entries is used to populate the TOT_NZ table field. The percent of non-zero entries (PER_NZ) can be obtained by dividing TOT_NZ by the total number of entries in the matrix. Increment eight of CRE contained modifications to the UME subsystem which simplified the placement of arcs in the usage model. A count of the usage model's arcs is used to directly populate the (TOT_ARCS) field.

Table update is to be performed at the end of the UME.CDL program. If errors are detected in the processing of the FTF file, no update is to be performed.

The TOTALS.DBF table is originally stored in the project/increment directory with one blank record. No record creation is required. Each time the CDLP.CDL program executes successfully, the record is updated with the new values.

Policies Subsystem Development

The Policies subsystem has not been implemented. In order to eliminate input selection of more than one file to the CM subsystem, the Policies subsystem should be implemented to allow creation of CRM files for input to the CM subsystem.

The Policies subsystem allows creation of user defined PLY files and provides the ability to use the PLY files to interpret failure data to create an input file to a reliability model. The failure data (FAIL.DBF and EC.DBF tables), from the user input DBS subsystem, is interpreted according to a PLY file to create a reliability model input file.

After selecting the Policies subsystem, the Policies window is displayed (shown with pull-down menus in Figure 7.26).

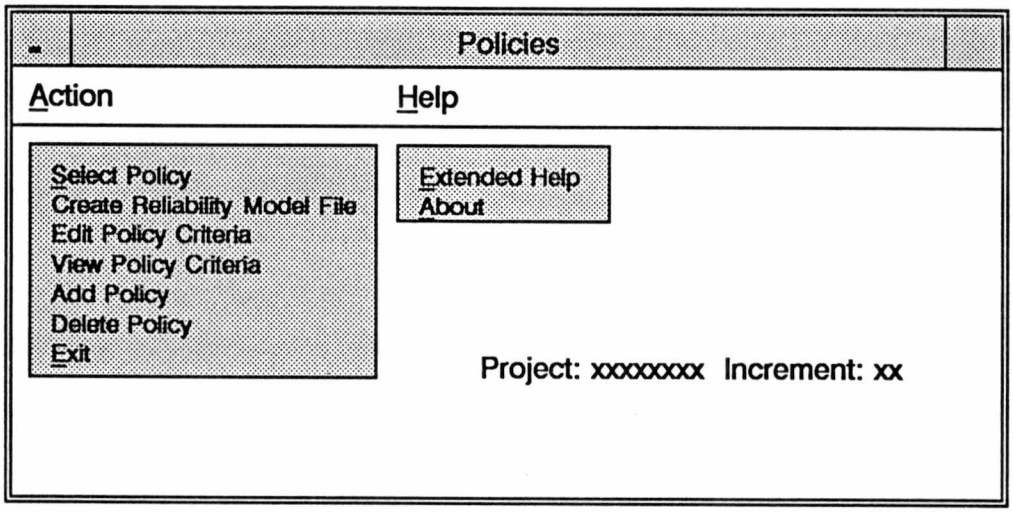


Figure 7.26 Policies Window

The **Select Policy** option allows the user to select from a list of existing PLY files. The **Create Reliability Model File** option allows the user to create a file to be used as input to a reliability model. The **Edit Policy Criteria** option allows the user to change the policy criteria in the selected policy. The **View Policy Criteria** option allows the user to view the policy criteria in the selected policy. The **Add Policy** option allows the user to create a PLY file. The **Delete Policy** option allows the user to delete the selected PLY file. If the user selects the **Delete Policy** option, a dialogue box is displayed inquiring if the user wishes to delete the PLY file or cancel the deletion. The **Exit** option is used to return to the Certification Assistant window.

If the user clicks twice on the **Select Policy** option, the list of PLY files in the selected project/increment directory is displayed (shown in Figure 7.27).

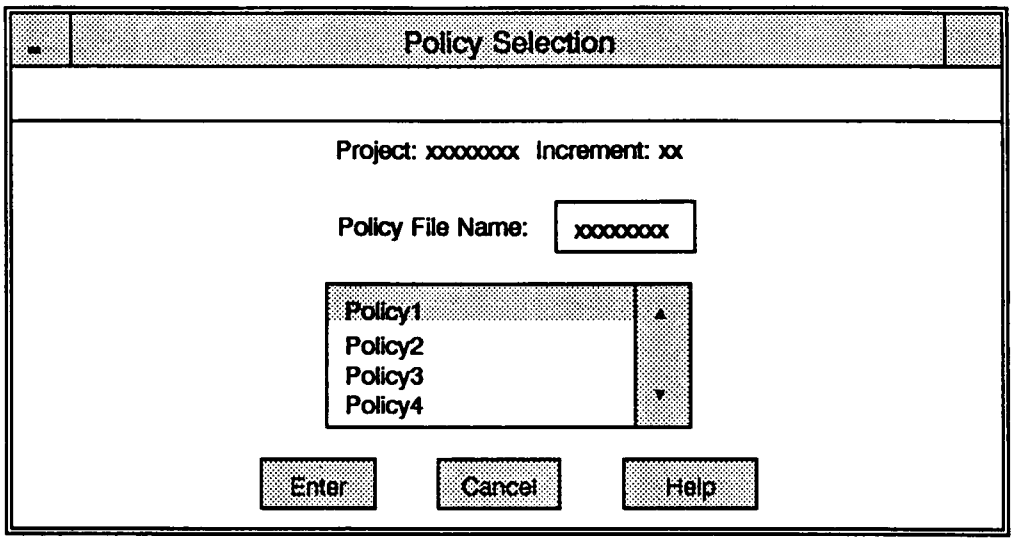


Figure 7.27 Policy Selection Window

The list of PLY files does not show the .PLY extensions in the name. The user may select a PLY file by highlighting the file name. The user may click on the up and down arrows to access additional file names. The user may select the file by typing the file name (the .PLY extension is not included) in the **Policy File Name** field and pressing enter (or clicking on the enter button). The input file name is highlighted in the selection window. If the input file name cannot be found, a dialogue box is displayed indicating that the file does not exist.

The user may use the mouse to click on the Enter, Cancel and Help buttons. Cancel is used to return to the Policies window without selecting a PLY file. Help is used to display help information. Enter is used to select the highlighted PLY file.

After a PLY has been selected, the Policies window is displayed. The selected PLY file is displayed in the window below the project and increment line. The selected PLY file is used by any subsequent action requiring a PLY file.

If the user clicks on the **Create Reliability Model File** option, the Create Reliability Model File window (shown in Figure 7.28) is displayed.

The Create Reliability Model File window displays the project and increment for which the failure data is to be retrieved and the currently selected policy. The user may select a targeted reliability model (this proposed Policies subsystem contains only the Certification Reliability Model). The user may select a reliability model by highlighting the file name and pressing the enter key or positioning the mouse pointer over the selection and clicking twice. The user may click on the up and down arrows to access additional reliability models.

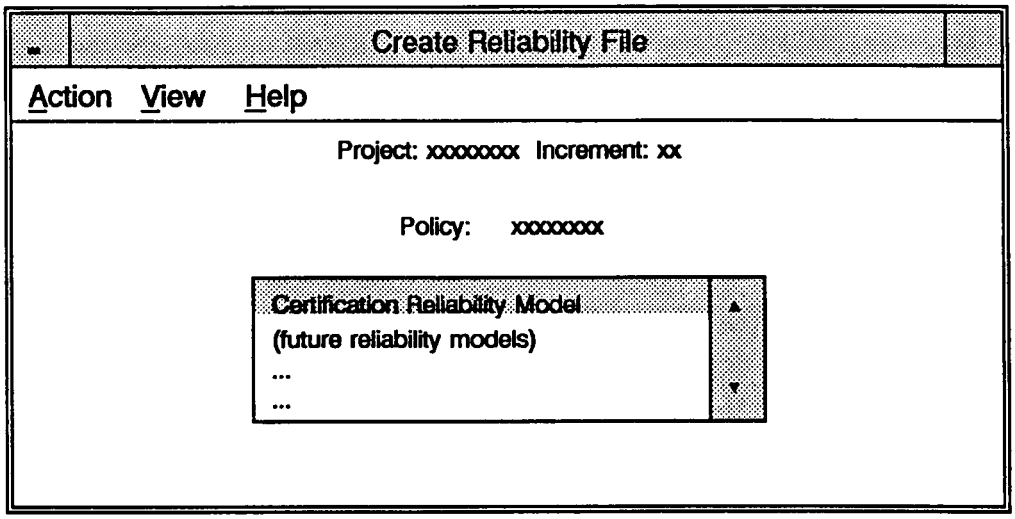


Figure 7.28 Create Reliability Model File Window

The **Action** menu displays the options: **Create File**, **Delete File**, and **Exit**. If the user selects the **Create File** option, the reliability model input file is created. The reliability model file is created with the same primary name as the PLY file and with the extension associated with the reliability model (the extension CRM is used for the Certification Reliability Model). If the reliability model file name already exists, the newly created file replaces the old file. If the user selects the **Delete File** option, the reliability model file with the primary name matching the PLY file and the extension associated with the selected reliability model is deleted. If the file does not exist, a dialogue box is displayed indicating that the file could not be found.

The **Edit Policy Criteria**, **View Policy Criteria**, and **Add Policy** options display the (Edit/View/Add) Policy window (shown in Figure 7.29). The title bar is to contain the appropriate title (Edit, View, or Add) for the option selected.

The **View Policy Criteria** option populates the field values before displaying, and no updates are allowed. The **Edit Policy Criteria** option populates the field values before displaying, and any field may be updated. The **Add Policy** option leaves the field values blank, and the user must enter values for all fields.

The **Action** pull-down menu is not available in the **View** selection. (An **Exit** option is used instead of **Action**.) The **Action** pull-down menu displays different options for the edit and add selections. The Edit Policy window contains the options: **Save**, **Abandon Update**, and **Exit**. The Add Policy window contains the **Add** and **Exit** options. The **Help** option is used to display help information.

If the **Save** or **Add** option is selected, entries in the fields are validated. If no errors are found, a PLY file is modified or created. After saving or adding a PLY file, this window is automatically exited, and the Policies window is displayed. If the option is **Save**, the PLY file replaces the changed PLY file. If the option is **Add**, the PLY file is created.

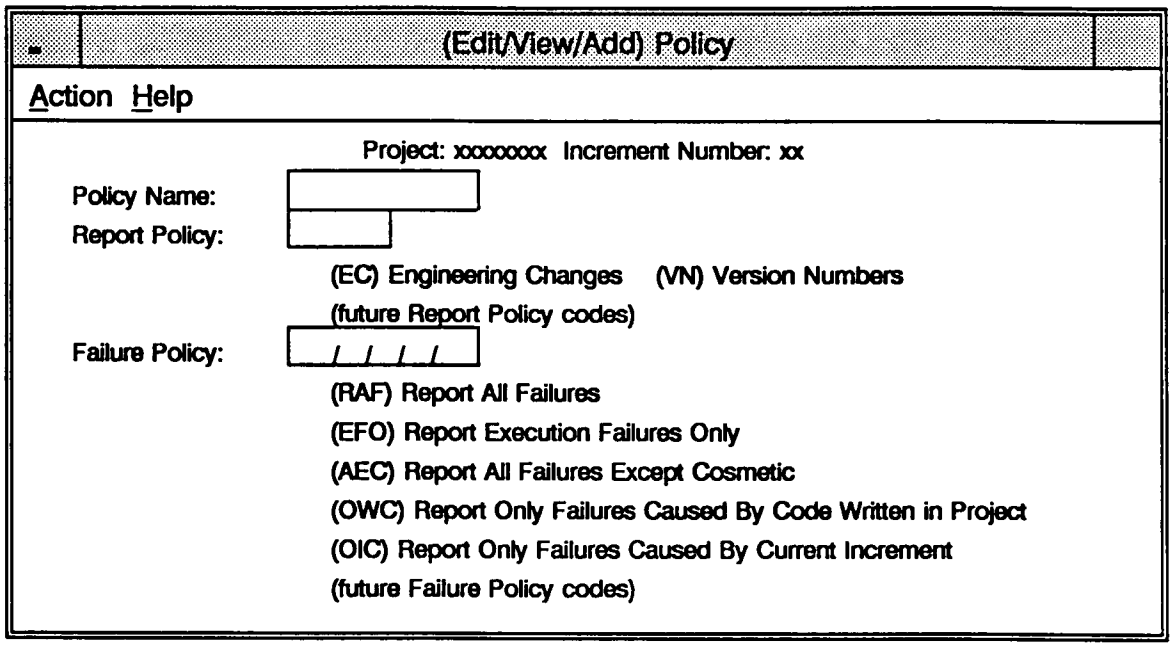


Figure 7.29 (Edit/View/Add) Policy Window

If the **Abandon Changes** option is selected, any changes to fields are restored to their original values. The **Exit** option is used to return to the Policies window. If changes were made before the **Exit** option was selected, a dialogue box is displayed indicating that changes were made and inquiring if the user wishes to exit without saving.

Policy Field Validations

The Policy Name field must be populated. The field must begin with an alphanumeric character, which cannot be more than eight characters long. Duplicate policy names may not be used in the same directory.

The Report Policy field must be either EC or VN. The field is required. In a future increment of CRE, other codes may be included.

The Failure Policy field is required. If the RAF code is used, no other codes may be included. As many as five codes may be combined to define how the failure data is to be interpreted. The combined codes include: EFO, AEC, OWC, and OIC. In a future version of CRE, the Policies subsystem may allow additional methods of interpreting failures, and other codes may be available.

Creating a Reliability Model File

If the Report Policy field in the PLY file is VN (Version Numbers), only the FAIL.DBF table is needed to create the CRM file. Each record in the FAIL.DBF is to be selected and

examined. If the failure is determined to match the Failure Reporting Policy in the PLY file, the TEST_CASE field is used to populate the Failure Time field and the VER_NO is used to populate the Independent Variable field. Each FAIL.DBF record included creates one CRM file line.

If the Report Policy field in the PLY file is EC (Engineering Change), the EC.DBF and FAIL.DBF tables are used to create the CRM file. All records in the EC.DBF table are selected. The FAIL_NO field on each EC.DBF record is used to select matching FAIL.DBF records. The matching FAIL.DBF records are examined to determine if they are to be included under the Failure Reporting Policy in the PLY file. If a FAIL.DBF record is to be included, the EC_NO field from the EC.DBF record is used to populate the Independent Variable field, and the TEST_CASE field on the FAIL.DBF is used to populate the Failure Time field. Each FAIL.DBF record included creates one CRM file line.

If the Failure Reporting Policy field in the PLY file is RAF (Report all Failures), all FAIL.DBF records examined are included in the CRM file. One field on the FAIL.DBF can be used to determine whether the FAIL.DBF record is to be included or not for each of the other Failure Reporting Policies. If the Failure Reporting Policy field is EFO (Report Execution Failures Only), the T_TYPE field must equal TESTING. If the Failure Reporting Policy field is AEC (Report All Failures Except Cosmetic), the E_TYPE field must not equal COSMETIC. If the Failure Reporting Policy field is OWC (Report Only Failures Caused By Code Written in Project), the USED_CD field must be blank. If the Failure Reporting Policy field is OIC (Report Only Failures Caused By Current Increment), the PREV_CD field must be blank.

A combination of Failure Reporting Policies may be used. The three character code is divided in the field input by "/" (which is displayed in the window and is not updatable). The PLY file does not include the slash in its policy criteria. If a combination is used, each of the associated FAIL.DBF fields is examined, and the record must match all combinations before it can be included in the CRM file.

CM Subsystem Modification

The current CM subsystem requests a TTF or DB file as input to produce the CMR file and graph. The proposed CM subsystem replaces the TTF and DB files with a CRM file as input. CRM files are created in the Policies subsystem. The interface to the CM subsystem should be changed to allow the user to select from a list of CRM files created in the Policies subsystem instead of selecting a TTF or DB file. The CRM file is the only input to the CM subsystem.

After selecting the CM subsystem, the Certification Reliability Model window is displayed (shown with pull-down menus in Figure 7.30).

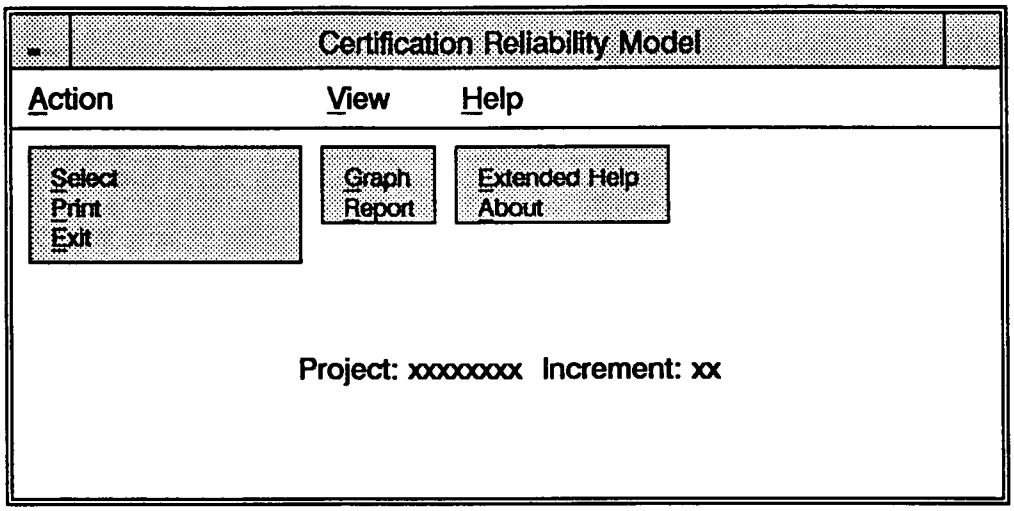


Figure 7.30 Certification Reliability Model Window

The Certification Reliability Model window is similar to the initial subsystem window in the current system (the exception being the **Action** pull-down menu replaces the **File** pull-down menu). The **View** and **Help** pull-down windows are the same.

The **Print** option is not available until the user has selected a CRM file. The **Exit** option is used to return to the Certification Assistant window. The **Select Policy** option allows the user to select from a list of CRM files created in the Policies subsystem. The selected CRM file is used to generate a certification report and graph.

If the user clicks on the **Select** option, the list of CRM files in the selected project/increment directory is displayed (shown in Figure 7.31). The current CRE lists all directories. When a directory is chosen, all TTF files in the directory are displayed. The **Select** replaces the **Open** option of the existing system.

The list of CRM files does not show the .CRM extensions in the name. The user may select a CRM file by highlighting the file name. The user may click on the up and down arrows to access additional CRM file names. The user may select the file by typing the file name (the .CRM extension is not included) in the **Certification Reliability Model File Name** field and pressing enter. The input file name is highlighted in the selection window. If the input file name cannot be found, a dialogue box is displayed indicating that the file does not exist.

After a CRM file is selected or the **Cancel** option is selected, the Certification Reliability Model window is displayed. The **Action**, **View**, and **Help** options are not available while the list of CRM files is displayed. The user may select the **Enter**, **Cancel**, or **Help** options at the bottom of the window (these options are available in the current CM subsystem's **Open** option window).

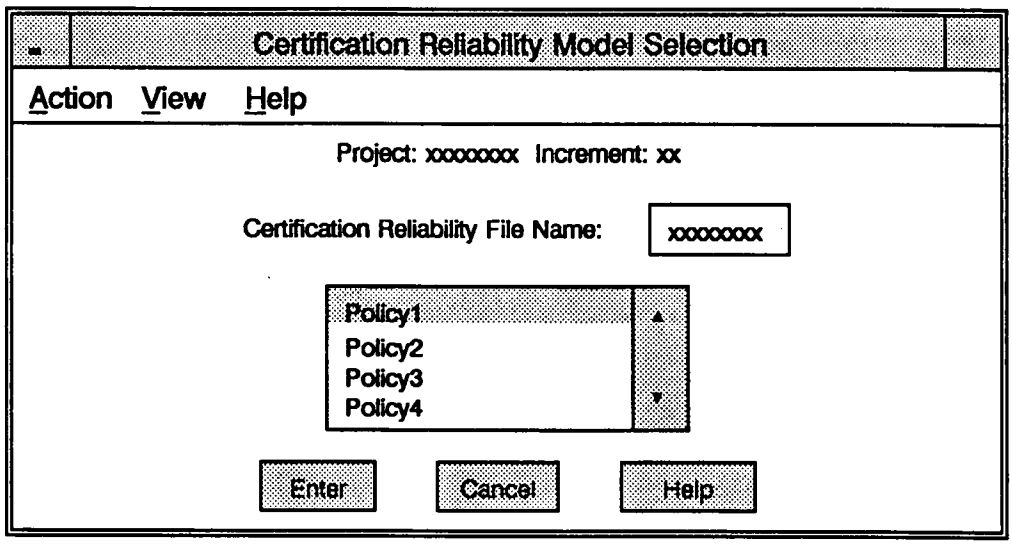


Figure 7.31 Certification Reliability Model Selection Window

After selecting a file, the CM subsystem processes the selected CRM file and the user may view the graph. The current CM subsystem distinguishes between the TTF and DB files and determines the MTTF entries according to the file being used. The MTTF for an independent variable in a TTF file is calculated by averaging the time-to-failure data (relative time) for that independent variable. The MTTF for an independent variable in a DB file is calculated by subtracting the last failure time (actual time) for the independent variable from the last failure time for the next independent variable. The CRM file contains only the failure time. The calculations required for the TTF file (relative time) are not required.

Testing Markov Chain Table Access

The Testing Markov Chain subsystem accesses the current TST file. The TST file is a user defined list of each record failure. The proposed database design includes a failure table (FAIL.DBF) that may be used for all failure data purposes. The database design does not require that the Testing Markov Chain subsystem use the FAIL.DBF; however, in order to reduce user input and data redundancy, it is recommended.

The TST file contains one text line for each failure. The lines contain consecutively numbered failure numbers, test case numbers, transition numbers, and an indication of whether the failure allowed testing to continue. These data elements are available in the FAIL.DBF.

Modifications to use the FAIL.DBF table instead of the TST ASCII file are minimal. The module function reading the TST file should be modified to perform SQL statements to read the FAIL.DBF table. Currently, if the TST file does not exist, zero failures are assumed. The same assumption can be made if the FAIL.DBF contains zero records.

If the FAIL.DBF does not exist, an error occurs. The SQL select statements can place the field values in the same C code variable names that are currently being used.

The lines containing consecutively numbered failure numbers are selected from the FAIL_NO fields. The test case numbers are selected from the TEST_CASE fields. The transition numbers are selected from TRAN_NO fields. The indication of whether a failure allowed testing to continue is obtained from the F_RESULTS fields (CONT or STOP).

File Selection Modifications

The current CRE requires that the user choose a directory before selection of input files may be made. After selection of an input file, the list of files that may be selected is limited to only files with the appropriate file extension name. The proposed directory driven data flow eliminates the requirement that the user select a directory. Modifications to the file selection windows are to reduce the user selections to only the appropriate files for the current project/increment directory.

The CDLP subsystem **File** pull-down menu presents the **Open** option that is to be modified. The code is to automatically choose the project/increment directory and display only the right portion of the selection window, which lists only files in the directory that have the .CDL file extension. The left portion of the selection window (the directory selection) is to be removed from the window.

The UME subsystem **File** pull-down menu presents the **Open** and **Translate MMN...** options that are to be modified. The code is to automatically choose the project/increment directory and display only the right portion of the selection window, which lists only files in the directory that have the .FTF or .MMN file extensions. The left portion of the selection window (the directory selection) is to be removed from the window.

The TCG subsystem **File** pull-down menu presents the **Open** option that is to be modified. The code is to automatically choose the project/increment directory and display only the right portion of the selection window, which lists only files in the directory that have the .FTP file extension. The left portion of the selection window (the directory selection) is to be removed from the window.

The Testing Markov Chain subsystem **File** pull-down menu presents the **Open** option that is to be modified. The code is to automatically choose the project/increment directory and display only the right portion of the selection window, which lists only files in the directory that have the .FTP file extension. The left portion of the selection window (the directory selection) is to be removed from the window.

The Certification Model subsystem is to be modified to accept the CRM file as input. The selection window is replaced in that modification.

Reports Subsystem Development

If the Reports subsystem is selected, the current CRE displays a dialogue box indicating that the Reports subsystem is to be implemented in a future release. This section presents proposed project development reports and a Report window interface. When the Reports subsystem is selected, the Report window (shown in Figure 7.32) is to be displayed.

The Reports window lists all reports available. The ten proposed reports are listed in Figure 7.32. The user selects a report by highlighting the associated report line. The user may print a selected report by pressing enter or clicking on the **Print** option.

The **Exit** option is used to return to the Management Assistance window. The **Help** option is used to display help information.

The proposed project development reports are shown in Appendix A. The report data sources are documented in the Appendix B.

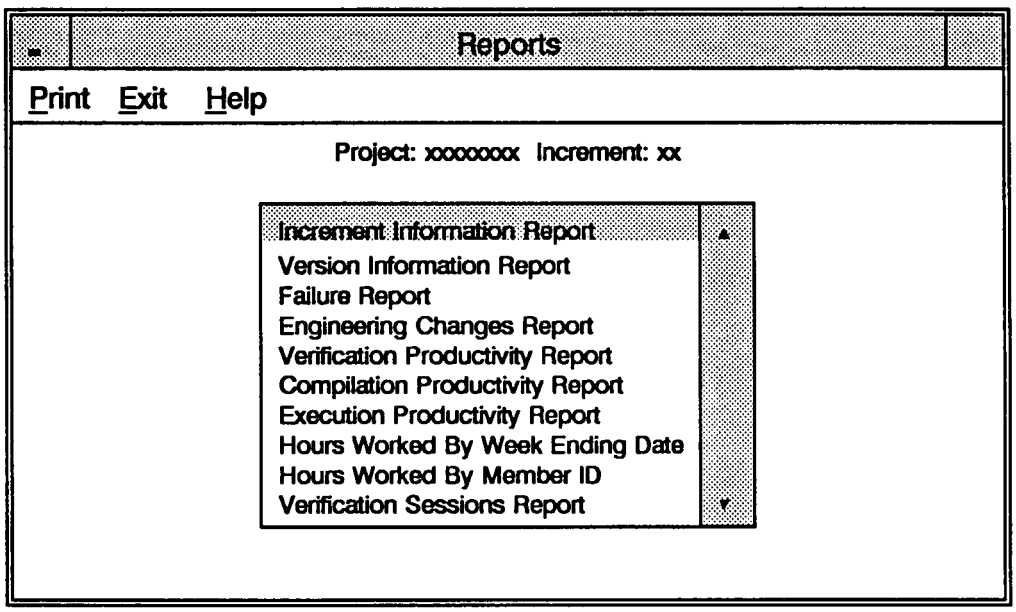


Figure 7.32 Reports Window

CHAPTER 8

DATABASE ACCESS METHODOLOGY

The QELIB User's Guide and sample programs provide complete information needed for using QELIB commands. This chapter highlights the functionality provided by QELIB in using C code to access dBase tables, clarifies some functionality that is unclear in the QELIB User's Guide, and makes suggestions for use in CRE. This chapter is not a substitute for examining the QELIB User's Guide and sample programs.

Installation

QELIB should be installed with the dBase database driver. Other database drivers may be installed, but only the dBase database driver is needed in CRE. The C sample programs should be included in the installation process. Installing the C samples creates a C:\QELIB\C directory with subdirectories: LIB and INCLUDE. The QELIB.LIB file must be included in the link step of any C program containing QELIB calls. The QELIB.H file contains definitions of all QELIB data types and other useful definitions. The QELIB.H file contains different definitions according to the C compiler used. The QELIB.H file may be altered to contain only data types recognized by the C/2 compiler.

The DLL files required for the dBase driver are automatically installed in the C:\OS2\DLL directory. The LIBPATH in the CONFIG.SYS must include this directory to gain access to the DLL files. (When using the Pioneer Software distribution license for QELIB, the QELIB files may be copied with the CRE software. A separate installation procedure is not required.)

QELIB creates a QEX.INI file that is stored in the C:\OS2 directory for the dBase database driver. This file contains default settings for access to the DBF files. The Compatibility setting is either ANSI or DBASE IV. The default setting is DBASE IV. The Compatibility setting should be changed to ANSI after installation. The Locking setting defaults to locking on a record. This setting is used when the system is executed on a network. Other Locking settings include locking on a DBF file or no locking. Default settings can be changed by executing C:\QELIB\QELSETUP.EXE and selecting the Configure option for the dBase database driver.

QELIB User's Guide

The QELIB User's Guide describes all functions used to access or manipulate data. There is a section documenting specific functionalities for each of the databases supported. The dBase database system documentation provides details for specific access to DBF tables. The environment in which CRE executes (C code accessing DBF tables) is predominantly used in examples.

Data Access

Access to database records consist of five primary steps: 1) connection to the dBase driver, 2) execution of SQL commands, 3) retrieving database field values, 4) closing the SQL statement, and 5) disconnection from dBase driver. These steps can be embedded into any portion of a C program which includes the QELIB definitions and is linked using the QELIB.LIB file.

Database Connection

A connection to the dBase driver must be established before any database activity can be performed. Connection is made with the statement:

```
hdbc = qeConnect("DRV=QEDBF");
```

The hdbc variable must be declared as an integer before being used. It is the handle to the database connection that is used as a parameter in sending SQL statements to the database. The qeConnect command is a QELIB function which uses a character string parameter determining the database driver to connect. The DRV=QEDBF parameter designates the dBase database driver.

SQL Statements

QELIB supports SQL as defined by ANSI (American National Standards Institute). SQL statements are divided into four types of statements: 1) read, 2) insert, 3) update, and 3) delete. An SQL statement is constructed as a character string and sent as a parameter in the qeExecSQL QELIB function. A select SQL statement to read the project name and description from the PROJECT.DBF table is of this format:

```
hstmt = qeExecSQL(hdbc, "SELECT PRJ_NAME, PRJ_DESC FROM PROJECT.DBF");
```

The DBF extension name is defaulted if not included. The hstmt variable must be declared as an integer before execution of the SQL statement. It is the handle to the SQL statement that is used as a parameter in all fetching functions to the database. Execution of the SQL statement does not retrieve data from the table; it provides the template to be used by fetch functions.

Data Value Binding

There are two methods for retrieving record field values into memory: qeVal and qeBindCol functions. The qeVal functions allow the code to specify the data type of the variable being populated by a database table field value. The qeBindCol functions require that all database fields be retrieved as character strings. The qeBindCol function automatically populates the C variables with the values from the database record without converting data types. The QELIB User's Guide states that when using the qeBindCol

function, a value will be retrieved as the data type of the variable receiving the value; however, that is not true of DBF tables (this is not stated in the QELIB User's Guide). The DBF files store decimal numbers in BCD (Binary-Coded Decimal) format. When a BCD value is returned, the returned value is not a numeric data type that can be defined in C (int, long, short, float, etc.). The DBF numbers should be retrieved by the qeBindCol function into a character string before being converted to a C numeric data type.

The qeBindCol function has the advantage of reading records faster (approximately twice as fast) with less overhead. In addition to the ability to retrieve data values into a variable of various data types, the qeVal functions can be used with languages that do not allow pointers to a long integer (which is required with qeBindCol), and long field names can be retrieved in smaller portions. The qeBindCol functions require that a field be retrieved in its entirety or it is truncated.

It is recommended that the qeBindCol method be used in order to maximize performance. The C language allows pointers to long integer variables (which is required for the qeBindCol function). The qeVal functionality was designed primarily as a means of retrieving data in the macro and script languages that do not provide the pointer capabilities.

The qeBindCol function has four required parameters: 1) SQL statement handle, field number (in order of sequence given in SQL statement), 3) variable name to be populated, and 4) pointer to variable containing the length (in bytes) of the receiving variable. A qeBindCol function must be executed for each field in the SQL select statement before a fetch can be performed. The qeBindCol function for a SQL select statement may be of the format:

qeBindCol (hstmt, 1, projectname, &name_len);
qeBindCol (hstmt, 2, projectdesc, &desc_len);

The hstmt variable indicates that the bind is associated with the latest executed SQL statement. The 1 and 2 parameters indicate that the PRJ_NAME and PRJ_DESC field values are to be accessed. The projectname and projectdesc variables are to be populated with the PRJ_NAME and PRJ_DESC values. (The QELIB User's Guide refers to these variables as pointers because the parameter points to the space to be filled with data. However, they are defined as character data type and not a pointer.) The name_len and desc_len variable should contain the number of bytes allocated to the projectname and projectdesc variables. The length variables must be declared as long integers and given a value before the qeBindCol function is performed.

If the qeBindCol function is being used to receive a number value (which is in BCD format), the length variable can be determined by its precision (the number of digits that can be stored in the number). Decimal numbers are defined by their precision and scale. The formula for calculating the length variable for a DBF number is:

length = (precision+2)/2;

The length calculation is required because BCD format stores two digits per byte. The

sign of the number is stored in an additional last byte. An example of this method is given in the QELIB sample programs.

QELIB provides functions that can be used before the qeBindCol function is executed that provide length information. The column width of a DBF field is returned by the qeColWidth function. The precision and scale of a decimal DBF field is returned by the qeColPrecision and qeColScale functions.

The dBase memo field may contain up to 65,535 characters. The qeBindCol function limits retrieval to 32,000 characters. It is recommended that the number of characters allowed for input to the memo fields be limited.

Data Value Fetching

After the SQL select statement has been executed and a qeBindCol function has been performed for each field in the SQL statement, the field values may be retrieved by a fetch function. There are three fetch functions: 1) qeFetchNext, 2) qeFetchPrev, and 3) qeFetchRandom. These functions require only the handle to the SQL statement as a parameter. The code to retrieve the PRJ_NAME and PRJ_DESC values from all records in the PROJECT.DBF table can be written in a While Do loop:

```
while (qeFetchNext (hstmt) == 0) {
        i = i+1;
        namearray (i)     =     projectname;
        descarray (i)     =     projectdesc;
        /* any other code */
}
```

The projectname and projectdesc variables are automatically populated from the PRJ_NAME and PRJ_DESC fields. The name_len and desc_len variables are automatically populated with the length of the returned values. If the record field value is truncated to fit into the size of the variable, the length variable is set to -1. If the record field value was blank or Null, the length variable is set to -2. (QELIB interprets blank dBase fields as null.) If the length field is to be used for a second SQL statement, it should be reestablished with the maximum expected size of the value before performing the next qeBindCol function. The length value may have been changed from the previous qeBindCol function while performing fetch routines. It is recommend that a module of code assigning the length variables be performed before performing a qeBindCol function.

The qeFetchNext function retrieves the next record in the table. If it is the first execution of the qeFetchNext function following the qeExecSQL statement, the first record in the table is retrieved. The returned code equals zero if a record is successfully retrieved. When using the qeBindCol method of data retrieval, the only other returned codes are -5 (end of file) and 4 (database system error). If the returned code is 4, the qeDBErr function can be called to retrieve the database system's error number, or the qeErrMsg function can be called to retrieve the error message. Values returned by the qeErrMsg

function may be as large as 512 characters. It is required that the variable set by the qeErrMsg function be large enough to hold 512 characters.

```
res_code      =  qeFetchNext (hstmt);
if res_code == 0 {
  number      =  qeDBErr ();
  msg         =  qeErrMsg ();
};
```

If the qeFetchPrev or qeFetchRandom functions is used, the qeFetchGetOptions function must be called prior to the fetch function.

```
qeFetchGetOptions (hstmt,1);
qeFetchPrev (hstmt);
--
--
num_recs  =  qeFetchNumRecs (hstmt);
qeFetchRandom (hstmt, num_recs);
```

The qeFetchOption requires the SQL statement handle variable and the integer 1 as parameters. This function sets a flag that allows the qeFetchPrev and qeFetchRandom functions to be performed. The example demonstrates how to retrieve the previous record and the last record. The qeFetchNumRecs sets the variable num_recs to the number of records accessed in the SQL select statement. The qeFetchRandom function allows the user to access a specific sequentially numbered record from the SQL statement.

Closing SQL Statement

The SQL statement must be closed after all functions using the hstmt variable have been performed. The statement used to close the SQL is of the format:

```
qeEndSQL (hstmt);
```

After closing an SQL statement, the next SQL statement may be executed using the same hstmt variable.

Disconnecting the Database Driver

The connection to the dBase driver remains in effect until it is disconnected. All database functions must be performed while connection is made to the dBase driver. The disconnect is of the format:

```
qeDisConnect (hdbc);
```

SQL Select Statements

The SQL select statement is a powerful feature in QELIB. It allows any number of records to be retrieved from one or more DBF tables according to a set of criteria. Retrieval of records may include the aggregate values: sum, average, count, maximum, and minimum. The select statement may be used to access records in specific sorted sequences.

The general format of a select statement is described in the QELIB User's Guide (pg. 5-38) as:

```
SELECT [DISTINCT] { * | <COL_EXPR>[, <COL_EXPR>...] }
FROM <DBF Table>[, <DBF Table> ...]
[ WHERE <conditions> ]
[ GROUP BY ,col_expr>[, <col_expr>...] ]
[ HAVING <conditions> ]
[ UNION [ALL]  (SELECT...) ]
[ ORDER BY <sort_expr>[, <sort_expr>...] ]
[ FOR UPDATE OF <col_expr>[, <col_expr>...] ]
```

Only the Select and From clauses are required. An example is:

```
SELECT PRJ_NAME, PRJ_DESC
FROM PROJECT.DBF
WHERE PRJ_NAME = projectname
ORDER BY PRJ_NAME;
```

The list of field names to be retrieved may be omitted and replaced with * to indicate that all of the table's fields are to be retrieved. The Order By clause should be used only if the order of retrieval is to be different from the index on the DBF table. The For Update clause is used to lock specific records or field names that are intended for update. The For Update clause is not required in order to update after the select statement; however, it should be used in a networking environment.

The "dBase-Compatible Database Files" section of the QELIB User's Guide provides explanations of each of the select statement options.

When using the qeExecSQL function, the SQL statement must be a character string parameter. This requires that a select statement be constructed with string copies into one character string variable and passed as a parameter. An example of C code using the qeExecSQL function to perform a select statement is:

```
strcpy (sql, "SELECT PRJ_NAME, PRJ_DESC, ROWID FROM ");
strcat (sql, Dir);
strcat (sql, "PROJECT.DBF WHERE PROJ_NAME = '");
strcat (sql, projectname);
strcat (sql, "'");

/* Send the SQL Statement to QELIB */
```

hstmt = qeExecSQL (hdbc, sql);

The **sql** variable contains the select statement and is used as a parameter in the qeExecSQL function. The **Dir** variable should contain the directory path to the location of the PROJECT.DBF table. If the directory is not included in the select statement, the DBF table must be in the current working directory.

It is recommended that the ROWID be retrieved when executing a select statement. The ROWID returns the unique row (record) number for the record retrieved. This number can be saved and used later with an update SQL statement to increase the performance of the update. The ROWID can be used in Where and Select clauses to increase the speed of performance and to ensure that the correct record is updated. ROWID is provided by the dBase database drive and functions as a field name returning an eight byte character data type.

The QELIB User's Guide contains a section describing the options available with the SQL selection statement. The Compatibility, Charset, Records, and Locking options may be changed for a specific SQL select by specifying them in the statement. The documentation does not indicate option defaults. The Compatibility and Locking defaults are set by the user during or after installation. The Records option determines if deleted records are to be retrieved - the default is undeleted. The Charset option determines whether the database files use the ANSI or IBM PC character set - the default is ANSI.

SQL Update Statements

The SQL update statement is used to change records in a DBF table. The update statement may contain a Where clause just as the select statement. It is recommended that the update statement use only ROWID as a where condition unless a global change is required. An example of an update statement is:

UPDATE PROJECT.DBF SET PRJ_STD = projectstartdate
WHERE ROWID = holdrowid;

This update statement changes the PRJ_STD field to the value in the projectstartdate variable for the specific record that matches the holdrowid variable. When retrieving the PROJECT.DBF records, the ROWID field should be moved to an array containing ROWIDs. The ROWID should be used as the record identifier.

Execution of the update statement immediately changes the PROJECT.DBF. QELIB provides the ability to suspend all update actions until a Commit is performed; however, the use of that functionality is not required.

SQL Insert Statements

The SQL insert statement is used to add records into a DBF table. The inserted record is automatically positioned correctly for each DBASE IV index associated with the table. (Note: if DBASE IV indexes are not used, all indexes for the table must be listed.) An example of an insert statement is:

```
INSERT INTO PROJECT.DBF VALUES (projectname, projectdesc,
 projectstartdate, projectfinishdate, projectcomments);
```

The insert statement updates the table immediately after execution.

SQL Delete Statements

The SQL delete statement is used to delete records in a DBF table. The delete statement may contain a Where clause just as the select statement. It is recommended that the delete statement use only ROWID as a where condition unless a global delete is required. An example of a delete statement is:

```
DELETE FROM PROJECT.DBF
        WHERE ROWID = holdrowid;
```

This update statement marks the record matching the holdrowid value as deleted immediately upon its execution. When retrieving the PROJECT.DBF record, the ROWID field should be moved to an array containing all ROWIDS. The ROWID should be used as the record identifier.

The delete statement automatically updates DBase IV indexes. (Note: if DBase IV indexes are not used, all indexes for the table must be listed.)

The delete statement does not actually remove the record from the table. The statement marks a record as deleted. A record that is marked as deleted cannot be retrieved with a select statement unless the Record option is changed to deleted. In order to remove records marked as deleted from the table, a pack statement must be used. An example is:

```
PACK PROJECT.DBF;
```

It is not recommended that the pack statement be incorporated into CRE code. The nature of the CRE system does not require large numbers of records to be deleted. The pack statement is a utility statement used to save space. The pack statement takes longer to perform than other data access statements. Logically deleted records in the table will not deteriorate the performance of the system significantly.

Sample Programs

The CRE environment uses the IBM OS/2 Programming Tools and Information Version 1.3 (Toolkit) software. The example C programs use Presentation Manager functions and contain references to files included in the Toolkit software. The C:\QELIB\C\EX02 directory includes the files: EX02.C, EX02.DEF, EX02.RC, and EX02.H. The EX02.C is a sample application that accesses the dBase files that are copied into the C:\QELIB directory. The EX02 program is the only sample application that demonstrates the use of the qeBindCol function. The Windows version of QELIB includes a MAKE file that compiles and links the sample programs using the large memory module. The OS/2 version of QELIB does not include a MAKE file. It is recommend that the large memory module be used when compiling and linking an OS/2 version sample program.

CHAPTER 9

FUTURE SUGGESTIONS

IBM C Set/2

IBM C/2 Version 1.1 is being used to develop CRE's executable C code. With the release of Version 2.0 of OS/2, IBM C Set/2 Version 1.0 for C code was made available as part of the IBM OS/2 Version 2.0 Tools for Application Development. The IBM C/2 product implements the SAA C CPI standard in the IBM Disk Operating System (DOS) and the 16-bit OS/2 environment. The IBM C Set/2 product provides a compiler that runs only in the 32-bit OS/2 environment with the compiler generating code specifically for the 32-bit OS/2. It is recommended that plans be made to migrate CRE to OS/2 Version 2.0 and the IBM C Set/2 Version 1.0 compiler.

SPSS for OS/2

SPSS for OS/2 is available from SPSS, Incorporated. SPSS executes with Presentation Manager and uses the multi-threading capabilities of OS/2. The software provides many levels of statistical reporting capabilities. It can translate data from various types of data storage - including ASCII files and dBase tables. The SPSS software may be of assistance in analyzing the statistical nature of CRE data.

There are four different modules of the SPSS software that may be purchased. The Base plus Statistics module provides descriptive statistics, frequencies, cross-tabulation, scatter plots, exploratory data analysis, reports, T-tests, linear regression, correlation analysis, non-parametrics, factor analysis, cluster analysis, and reliability analysis. The Advanced Statistics module provides discriminate analysis, log-linear modeling, non-linear regression, probit analysis, and survival analysis. The Trends module provides time series analysis and foresting module including curve fitting, smoothing, spectral analysis, and Box-Jenkins procedures. A Table module is also available.

Existing Reports

Several ASCII CRE reports are available in the current CRE. The ASCII reports may be printed from the subsystem modules in which they are created. These reports should be included in the Reports subsystem of CRE.

QELIB Versions

Pioneer Software is currently developing Version 2.0 of QELIB. The developers of CRE should monitor any new releases of QELIB for future use in CRE. Future releases may include performance improvement features. QELIB is currently marketing Version 1.1.1. This version operates with the OS/2 Version 2.0 and the IBM C Set/2 compiler; however,

it does not support the 32-bit environment. A later version of the QELIB software may be updated to use the 32-bit environment.

BIBLIOGRAPHY

BIBLIOGRAPHY

Ashbaugh, Ken. (1991, May). C Resource Guide. Data Based Advisor, pp. 72-88.

Bonner, Paul. (1992, February). Q+E Database Library. Computer Shopper, p. 503.

Crisp, R.L. and P.D. Manildi, "Cleanroom Software Engineering Documentation Study," Technical Report, Department of Computer Science, University of Tennessee, Knoxville, TN, April 1992.

Fuhrer, D., Mao, H, and Poore, J.H. "OS/2 Cleanroom Environment: A Progress Report on a Cleanroom Tools Development Project." Proceedings of HICSS-25, IEEE, 1992.

IBM. (October 22, 1991) Program Announcement 191-626, pp. 1-8.

IBM. (1989) Operating System/2 Extended Edition Version 1.2, Database Manager Programming Guide and Reference Volume 1: Guide.

IBM. (1991) Extended Services for OS/2 Version 2.0, Guide to Database Manager.

Linger, R.C., Mills, H.D., & Witt, B.I. (1979). Structured Programming. Addison-Wesley Publishing Company, Inc.: Philippines.

Litwin, Paul. (1991, May). Your Guide to STATS. Data Based Advisor, pp. 110-115.

Louderback, James. (1991, October). Finish with Q+E. Data Based Advisor, pp. 80-84.

Hawkins, John L. (1991, February). Database Experts Open Windows. Data Based Advisor, pp. 65.

Moser, Karen D. 1992, June 1). Pioneer Library Links Multiple Databases. PC Week, pp. 57-58.

Nesbitt, Kenn. (1991, December). Create Database Applications with Visual Basic. Data Based Advisor, pp. 126-129.

Pioneer Software. (June 1992) Q+E Database Library (QELIB) User's Guide: Pioneer Software Systems, Inc. Raleigh, NC.

Poore, J. H. and Harlan D. Mills, An Overview of the Cleanroom Software Development Process, Formal Methods Workshop, Halifax, Nova Scotia, July, 1989.

Poore, J. H., et. al. Case Study Report on the OS/2 CLeanroom Environment Development, Technical Reports, Vols. 1-8, Department of Computer Science, University of Tennessee, Knoxville, TN, December, 1991.

Shaw, Richard Hale. (1991, May). Seeking C for Database Work. Data Based Advisor, pp. 64-68.

APPENDICES

Appendix A
Project Development Reports

Current Date: mm/dd/yyyy Increment Information Report
Project: xxxxxxxx Increment: xx
Project Start Date: mm/dd/yyyy Project Finish Date: mm/dd/yyyy
Project Desc: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Project Comments:
xxxxxxxxxxxxxxxxxxxx

Current Date: mm/dd/yyyy

Version Information Report
Project: xxxxxxxx Increment: xx

Version Number	Submit Date	Verification Changes	Compilation Failures	Execution Failures	Engineering Changes
1	mm/dd/yyyy	xxxxx	xxxxx	xxxxx	xxxxx
2	mm/dd/yyyy	xxxxx	xxxxx	xxxxx	xxxxx
3	mm/dd/yyyy	xxxxx	xxxxx	xxxxx	xxxxx
4	mm/dd/yyyy	xxxxx	xxxxx	xxxxx	xxxxx
5	mm/dd/yyyy	xxxxx	xxxxx	xxxxx	xxxxx
6	mm/dd/yyyy	xxxxx	xxxxx	xxxxx	xxxxx
7	mm/dd/yyyy	xxxxx	xxxxx	xxxxx	xxxxx
	Totals:	xxxxx	xxxxx	xxxxx	xxxxx

Version 1 Comments:

Version 2 Comments:

Figure A.2 Version Information Report

Current Date: mm/dd/yyyy

Failure Report

Project: xxxxxxxx Increment: xx

Version: xxx

Failure Number	Test Case Number	Failure Result	Test Type	Error Cause	Error Type	Trans No.	Blocked By
xxxxx	xxxxx	xxxx	xxxxxxxxxxx	xxxxxxxxxx	xxxxxxxxxxxxxx	xxxxx	xxxxx
xxxxx	xxxxx	xxxx	xxxxxxxxxxx	xxxxxxxxxx	xxxxxxxxxxxxxx	xxxxx	xxxxx
xxxxx	xxxxx	xxxx	xxxxxxxxxxx	xxxxxxxxxx	xxxxxxxxxxxxxx	xxxxx	xxxxx
xxxxx	xxxxx	xxxx	xxxxxxxxxxx	xxxxxxxxxx	xxxxxxxxxxxxxx	xxxxx	xxxxx
xxxxx	xxxxx	xxxx	xxxxxxxxxxx	xxxxxxxxxx	xxxxxxxxxxxxxx	xxxxx	xxxxx
xxxxx	xxxxx	xxxx	xxxxxxxxxxx	xxxxxxxxxx	xxxxxxxxxxxxxx	xxxxx	xxxxx
xxxxx	xxxxx	xxxx	xxxxxxxxxxx	xxxxxxxxxx	xxxxxxxxxxxxxx	xxxxx	xxxxx

Version: xxx
Version: xxx
....
....

Failures Caused by Code Written in Current Increment

Failure Number	Version Number	Tester	Test Date	Test Time	Test Site	Error Message
xxxxx	xxx	xxx	mm/dd/yyyy	hh:mm	xxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxx
xxxxx	xxx	xxx	mm/dd/yyyy	hh:mm	xxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxx
xxxxx	xxx	xxx	mm/dd/yyyy	hh:mm	xxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxx

Total Failures: xxxxxx

Failures Caused by Code Written in Previous Increment

Failure Number	Previous Increment	Tester	Test Date	Test Time	Test Site	Error Message
xxxxx	xx	xxx	mm/dd/yyyy	hh:mm	xxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxx
xxxxx	xx	xxx	mm/dd/yyyy	hh:mm	xxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxx
xxxxx	xx	xxx	mm/dd/yyyy	hh:mm	xxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxx

Total Failures: xxxxxx

Figure A.3 Failure Report

Current Date: mm/dd/yyyy Failure Report (continued)
Project: xxxxxxxx Increment: xx

Failures Caused by Code Written in an Engineering Change						
Failure Number	Engineering Change	Tester	Test Date	Test Time	Test Site	Error Message
xxxxx	xxxxx	xxx	mm/dd/yyyy	hh:mm	xxxxxxxxx	xxxxxxxxxxxxxxxxxxxxx
xxxxx	xxxxx	xxx	mm/dd/yyyy	hh:mm	xxxxxxxxx	xxxxxxxxxxxxxxxxxxxxx
xxxxx	xxxxx	xxx	mm/dd/yyyy	hh:mm	xxxxxxxxx	xxxxxxxxxxxxxxxxxxxxx
Total Failures: xxxxxx						

Failures Caused by Reused Code (written outside of project)						
Fail No.	Used Code	Tester	Test Date	Test Time	Test Site	Error Message
xxxxx	xxxxxxxxxxxxxxxxxxx	xxx	mm/dd/yyyy	hh:mm	xxxxxxxxx	xxxxxxxxxxxxxxxxx
xxxxx	xxxxxxxxxxxxxxxxxxx	xxx	mm/dd/yyyy	hh:mm	xxxxxxxxx	xxxxxxxxxxxxxxxxx
xxxxx	xxxxxxxxxxxxxxxxxxx	xxx	mm/dd/yyyy	hh:mm	xxxxxxxxx	xxxxxxxxxxxxxxxxx
Total Failures: xxxxxx						

Failure 1 Comments: ....

Failure 2 Comments: ....

Failure 3 Comments: ....

...

...

...

Figure A.4 Failure Report Continued

Current Date: mm/dd/yyyy Engineering Changes Report
Project: xxxxxxxx Increment: xx

EC Number	Version Number	Severity Code	Classification Code	Designer(s)
1	xxx	xxxxx	xxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Comments: xxxxxxxxxxxxxxxxxxxx

Current Date: mm/dd/yyyy Verification Report

Project: xxxxxxxx Increment: xx

	Commission	% of Total Commission	Omission	% of Total Omission	Total	% of Total
Computation	xxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx
Control	xxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx
Cosmetic	xxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx
Data	xxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx
Initialization	xxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx
Interface	xxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx
Totals:	xxxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx

Figure A.6 Verification Report

Current Date: mm/dd/yyyy Compilation Report

Project: xxxxxxxx Increment: xx

	Commission	% of Total Commission	Omission	% of Total Omission	Total	% of Total
Computation	xxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx
Control	xxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx
Cosmetic	xxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx
Data	xxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx
Initialization	xxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx
Interface	xxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx
Totals:	xxxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx

Figure A.7 Compilation Report

Current Date: mm/dd/yyyy Execution Report

Project: xxxxxxxx Increment: xx

	Commission	% of Total Commission	Omission	% of Total Omission	Total	% of Total
Computation	xxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx
Control	xxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx
Cosmetic	xxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx
Data	xxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx
Initialization	xxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx
Interface	xxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx
Totals:	xxxxx	xxx.xx	xxxx	xxx.xx	xxxxx	xxx.xx

Figure A.8 Execution Report

Current Date: mm/dd/yyyy

Hours Worked By Week Ending Date
Project: xxxxxxxx Increment: xx

Week Ending	Member ID	IP	BSM	Verify	Certify	EC	Usage Model	Doc.	Misc.	Total
mm/dd/yyyy	xxx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
	xxx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
	xxx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
	xxx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
	Totals:	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
mm/dd/yyyy	xxx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
	xxx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
	xxx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
	xxx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
	Totals:	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
	Report Totals:	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx

Figure A.9 Hours Worked By Week Ending Date

Current Date: mm/dd/yyyy

Hours Worked By Member ID
Project: xxxxxxxx Increment: xx

Member ID	Week Ending	IP	BSM	Verify	Certify	EC	Usage Model	Doc.	Misc.	Total
xxx	mm/dd/yyyy	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
	mm/dd/yyyy	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
	mm/dd/yyyy	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
	mm/dd/yyyy	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
	Totals:	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
xxx	mm/dd/yyyy	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
	mm/dd/yyyy	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
	mm/dd/yyyy	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
	mm/dd/yyyy	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
	Totals:	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx
	Report Totals:	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx	xxx.xx

Figure A.10 Hours Worked By Member ID

Current Date: mm/dd/yyyy Verification Sessions Report

Project: xxxxxxxx Increment: xx

Verification Session	Ver No.	Date	Errors Found	No.of Hours	Found Per Hour	Avg Att.	No. of Manhours	Found Per Manhour
1	xxx	mm/dd/yyyy	xxxxx	xx.xx	xxx.xx	xx.xx	xxxx.xx	xxxx.xx
2	xxx	mm/dd/yyyy	xxxxx	xx.xx	xxx.xx	xx.xx	xxxx.xx	xxxx.xx
3	xxx	mm/dd/yyyy	xxxxx	xx.xx	xxx.xx	xx.xx	xxxx.xx	xxxx.xx
4	xxx	mm/dd/yyyy	xxxxx	xx.xx	xxx.xx	xx.xx	xxxx.xx	xxxx.xx
5	xxx	mm/dd/yyyy	xxxxx	xx.xx	xxx.xx	xx.xx	xxxx.xx	xxxx.xx
6	xxx	mm/dd/yyyy	xxxxx	xx.xx	xxx.xx	xx.xx	xxxx.xx	xxxx.xx
7	xxx	mm/dd/yyyy	xxxxx	xx.xx	xxx.xx	xx.xx	xxxx.xx	xxxx.xx
			xxxxxx	xxxx.x	xxx.xx	xx.xx	xxxxx.xx	xxxx.xx

Average Number of Clear Box Lines Verified per Session: xxxxxxxx

Average Number of Clear Box Lines Verified per Hour: xxxxxxxx

Average Number of Clear Box Lines Verified per Manhour: xxxxxxxx

Figure A.11 Verification Sessions Report

Appendix B
Project Development Reports Data Source

Increment Information Report Data Source

Report Field Name	Data Source Table	Data Source Field Name
Project Start Date	PROJECT	PRJ_STD
Project Finish Date	PROJECT	PRJ_FND
Project Desc	PROJECT	PRJ_DESC
Project Comments	PROJECT	COMMENTS
Increment Start Date	PROJECT	PRJ_STD
Increment Finish Date	PROJECT	PRJ_FND
Increment Desc	INCR	INCR_DESC
Increment Comments	INCR	COMMENTS
New Design Language (BSM) Lines	COUNT	ND_LINES
New Source Code (CB) Lines	COUNT	NS_LINES
Old Design Language (BSM) Lines	COUNT	OD_LINES
Old Source Code (CB) Lines	COUNT	OS_LINES
Changed Design Language (BSM) Lines	COUNT	CD_LINES
Changed Source Code (CB) Lines	COUNT	CS_LINES
Deleted Design Language (BSM) Lines	COUNT	DD_LINES
Deleted Source Code (CB) Lines	COUNT	DS_LINES
Reused Code Lines	COUNT	USED_CD
Avg. No. of Lines per New Clear Box	COUNT	CB_LINES/NS_LINES
Total Design Language (BSM) Lines	TOTAL	TOT_DL
Total Source Code (CB) Lines	TOTAL	TOT_SL
Number of New Black Boxes	COUNT	BB_NEW
Number of Old Black Boxes	COUNT	BB_OLD
Number of Changed Black Boxes	COUNT	BB_CHG
Number of Deleted Black Boxes	COUNT	BB_DEL
Number of New State Boxes	COUNT	SB_NEW
Number of Old State Boxes	COUNT	SB_OLD
Number of Changed State Boxes	COUNT	SB_CHG
Number of Deleted State Boxes	COUNT	SB_DEL
Number of New Clear Boxes	COUNT	CB_NEW
Number of Old Clear Boxes	COUNT	CB_OLD
Number of Changed Clear Boxes	COUNT	CB_CHG
Number of Deleted Clear Boxes	COUNT	CB_DEL
Total Number of Black Boxes	TOTAL	TOT_BB
Total Number of State Boxes	TOTAL	TOT_SB
Total Number of Clear Boxes	TOTAL	TOT_CB
Avg. Number of Lines per Clear Box	TOTAL	TOT_CB/TOT_SL
Number of Usage States	TOTAL	TOT_US
Number of Arcs in Usage Model	TOTAL	TOT_ARCS
Number of Non-Zero Matrix Entries	TOTAL	TOT_NZ
Percent of Non-Zero Matrix Entries	TOTAL	PER_NZ

Version Information Report Data Source

Report Field Name	Data Source
Version Number	VERSION->VER_NO
Submit Date	VERSION->SUB_DT
Verification Changes	For each VERIFY record where VERIFY->VER_NO = VERSION->VER_NO, Sum the COUNT of (FOUND records where FOUND->VF_NO = VERIFY->VF_NO)
Compilation Failures	COUNT of (FAIL records where FAIL->VER_NO = VERSION->VER_NO and T_TYPE = 'COMPILATION')
Execution Failures	COUNT of (FAIL records where FAIL->VER_NO = VERSION->VER_NO and T_TYPE = 'TESTING')
Engineering Changes	COUNT of (EC records where EC->VER_NO = VERSION->VER_NO)
Version Comments	VERSION->COMMENTS

Failure Report Data Source

Report Field Name	Data Source
Version	VERSION->VER_NO
Failure Number	For Each FAIL record where FAIL->VER_NO = VERSION->VER_NO, FAIL->FAIL_NO
Test Case Number	For Each FAIL record where FAIL->VER_NO = VERSION->VER_NO, FAIL->TEST_CASE
Failure Result	For Each FAIL record where FAIL->VER_NO = VERSION->VER_NO, FAIL->F_RESULT
Test Type	For Each FAIL record where FAIL->VER_NO = VERSION->VER_NO, FAIL->T_TYPE
Error Cause	For Each FAIL record where FAIL->VER_NO = VERSION->VER_NO, FAIL->E_CAUSE
Error Type	For Each FAIL record where FAIL->VER_NO = VERSION->VER_NO, FAIL->E_TYPE
Trans No.	For Each FAIL record where FAIL->VER_NO = VERSION->VER_NO, FAIL->TRAN_NO
Blocked By	For Each FAIL record where FAIL->VER_NO = VERSION->VER_NO, FAIL->BLK_BY

Failures Caused by Code Written in Current Increment Data Source

Report Field Name	Data Source
Failure Number	For Each FAIL record where FAIL->PREV_INCR = NULL FAIL->FAIL_NO
Version Number	For Each FAIL record where FAIL->PREV_INCR = NULL FAIL->VER_NO
Tester	For Each FAIL record where FAIL->PREV_INCR = NULL FAIL->TESTER
Test Date	For Each FAIL record where FAIL->PREV_INCR = NULL FAIL->TEST_DT
Test Time	For Each FAIL record where FAIL->PREV_INCR = NULL FAIL->TEST_TM
Test Site	For Each FAIL record where FAIL->PREV_INCR = NULL FAIL->SITE
Error Message	For Each FAIL record where FAIL->PREV_INCR = NULL FAIL->ERR_MESS
Total Failures	COUNT of (FAIL records where FAIL->PREV_INCR = NULL)

Failures Caused by Code Written in Previous Increment Data Source

Report Field Name	Data Source
Failure Number	For Each FAIL record where FAIL->PREV_INCR not = NULL FAIL->FAIL_NO
Previous Increment	For Each FAIL record where FAIL->PREV_INCR not = NULL FAIL->PREV_INCR
Tester	For Each FAIL record where FAIL->PREV_INCR not = NULL FAIL->TESTER
Test Date	For Each FAIL record where FAIL->PREV_INCR not = NULL FAIL->TEST_DT
Test Time	For Each FAIL record where FAIL->PREV_INCR not = NULL FAIL->TEST_TM
Test Site	For Each FAIL record where FAIL->PREV_INCR not = NULL FAIL->SITE
Error Message	For Each FAIL record where FAIL->PREV_INCR not = NULL FAIL->ERR_MESS
Total Failures	COUNT of (FAIL records where FAIL->PREV_INCR not = NULL)

Failures Caused by Code Written in an Engineering Change Data Source

Report Field Name	Data Source
Failure Number	For Each FAIL record where FAIL->EC_NO not = NULL FAIL->FAIL_NO
Engineering Change	For Each FAIL record where FAIL->EC_NO not = NULL FAIL->EC_NO
Tester	For Each FAIL record where FAIL->EC_NO not = NULL FAIL->TESTER
Test Date	For Each FAIL record where FAIL->EC_NO not = NULL FAIL->TEST_DT
Test Time	For Each FAIL record where FAIL->EC_NO not = NULL FAIL->TEST_TM
Test Site	For Each FAIL record where FAIL->EC_NO not = NULL FAIL->SITE
Error Message	For Each FAIL record where FAIL->EC_NO not = NULL FAIL->ERR_MESS
Total Failures	COUNT of (FAIL records where FAIL->EC_NO not = NULL)

Failures Caused by Reused Code Data Source

Report Field Name	Data Source
Fail No.	For Each FAIL record where FAIL->USED_CD not = NULL FAIL->FAIL_NO
Used Code	For Each FAIL record where FAIL->USED_CD not = NULL FAIL->USED_CD
Tester	For Each FAIL record where FAIL->USED_CD not = NULL FAIL->TESTER
Test Date	For Each FAIL record where FAIL->USED_CD not = NULL FAIL->TEST_DT
Test Time	For Each FAIL record where FAIL->USED_CD not = NULL FAIL->TEST_TM
Test Site	For Each FAIL record where FAIL->USED_CD not = NULL FAIL->SITE
Error Message	For Each FAIL record where FAIL->USED_CD not = NULL FAIL->ERR_MESS
Total Failures	COUNT of (FAIL records where FAIL->USED_CD not = NULL)

Engineering Changes Report Data Source

Report Field Name	Data Source
EC Number	EC->EC_NO
Version Number	EC->VER_NO
Severity Code	EC->SEVERITY
Classification Code	EC->CLASSIFY
Designer(s)	EC->DESIGNER
Comments	EC->COMMENTS

Verification Report Data Source

Report Field Name	Data Source
Computation/Commission	COUNT of (FOUND records where E_CAUSE = 'COMMISSION' and E_TYPE = 'COMPUTATION')
Computation/ % of Total Commission	Computation/Commission divided by Sum of Computation/Commission and COUNT (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'COMPUTATION')
Computation/Omission	COUNT of (FOUND records where E_CAUSE = 'OMISSION' and E_TYPE = 'COMPUTATION')
Computation/ % of Total Omission	Computation/Omission divided by Sum of Computation/Omission and COUNT (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'COMPUTATION')
Computation/Total	Sum of Computation/Commission and Computation/Omission
Computation/ % of Total	Computation/Total divided by Sum of Computation/Total and COUNT of (FAIL records where E_TYPE = 'COMPUTATION')
Control/Commission	COUNT of (FOUND records where E_CAUSE = 'COMMISSION' and E_TYPE = 'CONTROL')
Control/ % of Total Commission	Control/Commission divided by Sum of Control/Commission and COUNT (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'CONTROL')
Control/Omission	COUNT of (FOUND records where E_CAUSE = 'OMISSION' and E_TYPE = 'CONTROL')
Control/ % of Total Omission	Control/Omission divided by Sum of Control/Omission and COUNT (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'CONTROL')
Control/Total	Sum of Control/Commission and Control/Omission
Control/ % of Total	Control/Total divided by Sum of Control/Total and COUNT of (FAIL records where E_TYPE = 'CONTROL')
Cosmetic/Commission	COUNT of (FOUND records where E_CAUSE = 'COMMISSION' and E_TYPE = 'COSMETIC')
Cosmetic/ % of Total Commission	Cosmetic/Commission divided by Sum of Cosmetic/Commission and COUNT (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'COSMETIC')
Cosmetic/Omission	COUNT of (FOUND records where E_CAUSE = 'OMISSION' and

	E_TYPE = 'COSMETIC')
Cosmetic/ % of Total Omission	Cosmetic/Omission divided by Sum of Cosmetic/Omission and COUNT (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'COSMETIC')
Cosmetic/Total	Sum of Cosmetic/Commission and Cosmetic/Omission
Cosmetic/ % of Total	Cosmetic/Total divided by Sum of Cosmetic/Total and COUNT of (FAIL records where E_TYPE = 'COSMETIC')
Data/Commission	COUNT of (FOUND records where E_CAUSE = 'COMMISSION' and E_TYPE = 'DATA')
Data/ % of Total Commission	Data/Commission divided by Sum of Data/Commission and COUNT (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'DATA')
Data/Omission	COUNT of (FOUND records where E_CAUSE = 'OMISSION' and E_TYPE = 'DATA')
Data/ % of Total Omission	Data/Omission divided by Sum of Data/Omission and COUNT (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'DATA')
Data/Total	Sum of Data/Commission and Data/Omission
Data/ % of Total	Data/Total divided by Sum of Data/Total and COUNT of (FAIL records where E_TYPE = 'DATA')
Initialization/Commission	COUNT of (FOUND records where E_CAUSE = 'COMMISSION' and E_TYPE = 'INITIALIZATION')
Initialization/ % of Total Commission	Initialization/Commission divided by Sum of Initialization/Commission and COUNT (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'INITIALIZATION')
Initialization/Omission	COUNT of (FOUND records where E_CAUSE = 'OMISSION' and E_TYPE = 'INITIALIZATION')
Initialization/ % of Total Omission	Initialization/Omission divided by Sum of Initialization/Omission and COUNT (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'INITIALIZATION')
Initialization/Total	Sum of Initialization/Commission and Initialization/Omission
Initialization/ % of Total	Initialization/Total divided by Sum of Initialization/Total and

	COUNT of (FAIL records where E_TYPE = 'INITIALIZATION')
Interface/Commission	COUNT of (FOUND records where E_CAUSE = 'COMMISSION' and E_TYPE = 'INTERFACE')
Interface/ % of Total Commission	Interface/Commission divided by Sum of Interface/Commission and COUNT (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'INTERFACE')
Interface/Omission	COUNT of (FOUND records where E_CAUSE = 'OMISSION' and E_TYPE = 'INTERFACE')
Total/Commission	COUNT of (FOUND records where E_CAUSE = 'COMMISSION')
Total/ % of Total Commission	Total/Commission divided by Sum of Total/Commission and COUNT (FAIL records where E_CAUSE = 'COMMISSION')
Total/Omission	COUNT of (FOUND records where E_CAUSE = 'OMISSION')
Total/ % of Total Omission	Total/Omission divided by Sum of Total/Omission and COUNT (FAIL records where E_CAUSE = 'OMISSION')
Total/Total	Sum of Total/Commission and Total/Omission
Total/ % of Total	Total/Total divided by Sum of Total/Total and COUNT of (FAIL records)

Compilation Report Data Source

Report Field Name	Data Source
Computation/Commission	COUNT of (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'COMPUTATION' and T_TYPE = 'COMPILATION')
Computation/ % of Total Commission	Computation/Commission divided by Sum of COUNT (FOUND records where E_CAUSE = 'COMMISSION' and E_TYPE = 'COMPUTATION') and COUNT (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'COMPUTATION')
Computation/Omission	COUNT of (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'COMPUTATION' and T_TYPE = 'COMPILATION')
Computation/ % of Total Omission	Computation/Omission divided by Sum of COUNT (FOUND records where E_CAUSE = 'OMISSION' and E_TYPE = 'COMPUTATION') and COUNT (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'COMPUTATION')
Computation/Total	Sum of Computation/Commission and Computation/Omission
Computation/ % of Total	Computation/Total divided by Sum of COUNT of (FOUND records where E_TYPE = 'COMPUTATION') and COUNT of (FAIL records where E_TYPE = 'COMPUTATION')
Control/Commission	COUNT of (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'CONTROL' and T_TYPE = 'COMPILATION')
Control/ % of Total Commission	Control/Commission divided by Sum of COUNT (FOUND records where E_CAUSE = 'COMMISSION' and E_TYPE = 'CONTROL') and COUNT (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'CONTROL')
Control/Omission	COUNT of (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'CONTROL' and T_TYPE = 'COMPILATION')
Control/ % of Total Omission	Control/Omission divided by Sum of COUNT (FOUND records where E_CAUSE = 'OMISSION' and E_TYPE = 'CONTROL') and COUNT (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'CONTROL')
Control/Total	Sum of Control/Commission and Control/Omission
Control/ % of Total	Control/Total divided by Sum of COUNT of (FOUND records where E_TYPE = 'CONTROL') and COUNT of (FAIL records where E_TYPE = 'CONTROL')
Cosmetic/Commission	COUNT of (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'COSMETIC' and T_TYPE = 'COMPILATION')

Cosmetic/ % of Total Commission	Cosmetic/Commission divided by Sum of COUNT (FOUND records where E_CAUSE = 'COMMISSION' and E_TYPE = 'COSMETIC') and COUNT (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'COSMETIC')
Cosmetic/Omission	COUNT of (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'COSMETIC' and T_TYPE = 'COMPILATION')
Cosmetic/ % of Total Omission	Cosmetic/Omission divided by Sum of COUNT (FOUND records where E_CAUSE = 'OMISSION' and E_TYPE = 'COSMETIC') and COUNT (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'COSMETIC')
Cosmetic/Total	Sum of Cosmetic/Commission and Cosmetic/Omission
Cosmetic/ % of Total	Cosmetic/Total divided by Sum of COUNT of (FOUND records where E_TYPE = 'COSMETIC') and COUNT of (FAIL records where E_TYPE = 'COSMETIC')
Data/Commission	COUNT of (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'DATA' and T_TYPE = 'COMPILATION')
Data/ % of Total Commission	Data/Commission divided by Sum of COUNT (FOUND records where E_CAUSE = 'COMMISSION' and E_TYPE = 'DATA') and COUNT (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'DATA')
Data/Omission	COUNT of (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'DATA' and T_TYPE = 'COMPILATION')
Data/ % of Total Omission	Data/Omission divided by Sum of COUNT (FOUND records where E_CAUSE = 'OMISSION' and E_TYPE = 'DATA') and COUNT (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'DATA')
Data/Total	Sum of Data/Commission and Data/Omission
Data/ % of Total	Data/Total divided by Sum of COUNT of (FOUND records where E_TYPE = 'DATA') and COUNT of (FAIL records where E_TYPE = 'DATA')
Initialization/Commission	COUNT of (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'INITIALIZATION' and T_TYPE = 'COMPILATION')
Initialization/ % of Total Commission	Initialization/Commission divided by Sum of COUNT (FOUND records where E_CAUSE = 'COMMISSION' and E_TYPE = 'INITIALIZATION') and COUNT (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'INITIALIZATION')

Initialization/Omission	COUNT of (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'INITIALIZATION' and T_TYPE = 'COMPILATION')
Initialization/ % of Total Omission	Initialization/Omission divided by Sum of COUNT (FOUND records where E_CAUSE = 'OMISSION' and E_TYPE = 'INITIALIZATION') and COUNT (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'INITIALIZATION')
Initialization/Total	Sum of Initialization/Commission and Initialization/Omission
Initialization/ % of Total	Initialization/Total divided by Sum of COUNT of (FOUND records where E_TYPE = 'INITIALIZATION') and COUNT of (FAIL records where E_TYPE = 'INITIALIZATION')
Interface/Commission	COUNT of (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'INTERFACE' and T_TYPE = 'COMPILATION')
Interface/ % of Total Commission	Interface/Commission divided by Sum of COUNT (FOUND records where E_CAUSE = 'COMMISSION' and E_TYPE = 'INTERFACE') and COUNT (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'INTERFACE')
Interface/Omission	COUNT of (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'INTERFACE' and T_TYPE = 'COMPILATION')
Interface/ % of Total Omission	Interface/Omission divided by Sum of COUNT (FOUND records where E_CAUSE = 'OMISSION' and E_TYPE = 'INTERFACE') and COUNT (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'INTERFACE')
Interface/Total	Sum of Interface/Commission and Interface/Omission
Interface/ % of Total	Interface/Total divided by Sum of COUNT of (FOUND records where E_TYPE = 'INTERFACE') and COUNT of (FAIL records where E_TYPE = 'INTERFACE')
Total/Commission	COUNT of (FAIL records where E_CAUSE = 'COMMISSION' and T_TYPE = 'COMPILATION')
Total/ % of Total Commission	Total/Commission divided by Sum of COUNT (FOUND records where E_CAUSE = 'COMMISSION') and COUNT (FAIL records where E_CAUSE = 'COMMISSION')
Total/Omission	COUNT of (FAIL records where E_CAUSE = 'OMISSION' and T_TYPE = 'COMPILATION')
Total/ % of Total Omission	Total/Omission divided by Sum of COUNT (FOUND records where E_CAUSE = 'OMISSION') and COUNT (FAIL records where E_CAUSE = 'OMISSION')

Total/Total	Sum of Total/Commission and Total/Omission
Total/ % of Total	Total/Total divided by Sum of COUNT of (FOUND records) and COUNT of (FAIL records)

Execution Report Data Source

Report Field Name	Data Source
Computation/Commission	COUNT of (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'COMPUTATION' and T_TYPE = 'TESTING')
Computation/ % of Total Commission	Computation/Commission divided by Sum of COUNT (FOUND records where E_CAUSE = 'COMMISSION' and E_TYPE = 'COMPUTATION') and COUNT (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'COMPUTATION')
Computation/Omission	COUNT of (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'COMPUTATION' and T_TYPE = 'TESTING')
Computation/ % of Total Omission	Computation/Omission divided by Sum of COUNT (FOUND records where E_CAUSE = 'OMISSION' and E_TYPE = 'COMPUTATION') and COUNT (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'COMPUTATION')
Computation/Total	Sum of Computation/Commission and Computation/Omission
Computation/ % of Total	Computation/Total divided by Sum of COUNT of (FOUND records where E_TYPE = 'COMPUTATION') and COUNT of (FAIL records where E_TYPE = 'COMPUTATION')
Control/Commission	COUNT of (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'CONTROL' and T_TYPE = 'TESTING')
Control/ % of Total Commission	Control/Commission divided by Sum of COUNT (FOUND records where E_CAUSE = 'COMMISSION' and E_TYPE = 'CONTROL') and COUNT (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'CONTROL')
Control/Omission	COUNT of (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'CONTROL' and T_TYPE = 'TESTING')
Control/ % of Total Omission	Control/Omission divided by Sum of COUNT (FOUND records where E_CAUSE = 'OMISSION' and E_TYPE = 'CONTROL') and COUNT (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'CONTROL')
Control/Total	Sum of Control/Commission and Control/Omission
Control/ % of Total	Control/Total divided by Sum of COUNT of (FOUND records where E_TYPE = 'CONTROL') and COUNT of (FAIL records where E_TYPE = 'CONTROL')
Cosmetic/Commission	COUNT of (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'COSMETIC' and T_TYPE = 'TESTING')

Cosmetic/ % of Total Commission	Cosmetic/Commission divided by Sum of COUNT (FOUND records where E_CAUSE = 'COMMISSION' and E_TYPE = 'COSMETIC') and COUNT (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'COSMETIC')
Cosmetic/Omission	COUNT of (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'COSMETIC' and T_TYPE = 'TESTING')
Cosmetic/ % of Total Omission	Cosmetic/Omission divided by Sum of COUNT (FOUND records where E_CAUSE = 'OMISSION' and E_TYPE = 'COSMETIC') and COUNT (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'COSMETIC')
Cosmetic/Total	Sum of Cosmetic/Commission and Cosmetic/Omission
Cosmetic/ % of Total	Cosmetic/Total divided by Sum of COUNT of (FOUND records where E_TYPE = 'COSMETIC') and COUNT of (FAIL records where E_TYPE = 'COSMETIC')
Data/Commission	COUNT of (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'DATA' and T_TYPE = 'TESTING')
Data/ % of Total Commission	Data/Commission divided by Sum of COUNT (FOUND records where E_CAUSE = 'COMMISSION' and E_TYPE = 'DATA') and COUNT (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'DATA')
Data/Omission	COUNT of (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'DATA' and T_TYPE = 'TESTING')
Data/ % of Total Omission	Data/Omission divided by Sum of COUNT (FOUND records where E_CAUSE = 'OMISSION' and E_TYPE = 'DATA') and COUNT (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'DATA')
Data/Total	Sum of Data/Commission and Data/Omission
Data/ % of Total	Data/Total divided by Sum of COUNT of (FOUND records where E_TYPE = 'DATA') and COUNT of (FAIL records where E_TYPE = 'DATA')
Initialization/Commission	COUNT of (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'INITIALIZATION' and T_TYPE = 'TESTING')
Initialization/ % of Total Commission	Initialization/Commission divided by Sum of COUNT (FOUND records where E_CAUSE = 'COMMISSION' and E_TYPE = 'INITIALIZATION') and COUNT (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'INITIALIZATION')

Initialization/Omission	COUNT of (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'INITIALIZATION' and T_TYPE = 'TESTING')
Initialization/ % of Total Omission	Initialization/Omission divided by Sum of COUNT (FOUND records where E_CAUSE = 'OMISSION' and E_TYPE = 'INITIALIZATION') and COUNT (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'INITIALIZATION')
Initialization/Total	Sum of Initialization/Commission and Initialization/Omission
Initialization/ % of Total	Initialization/Total divided by Sum of COUNT of (FOUND records where E_TYPE = 'INITIALIZATION') and COUNT of (FAIL records where E_TYPE = 'INITIALIZATION')
Interface/Commission	COUNT of (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'INTERFACE' and T_TYPE = 'TESTING')
Interface/ % of Total Commission	Interface/Commission divided by Sum of COUNT (FOUND records where E_CAUSE = 'COMMISSION' and E_TYPE = 'INTERFACE') and COUNT (FAIL records where E_CAUSE = 'COMMISSION' and E_TYPE = 'INTERFACE')
Interface/Omission	COUNT of (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'INTERFACE' and T_TYPE = 'TESTING')
Interface/ % of Total Omission	Interface/Omission divided by Sum of COUNT (FOUND records where E_CAUSE = 'OMISSION' and E_TYPE = 'INTERFACE') and COUNT (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'INTERFACE')
Interface/Total	Sum of Interface/Commission and Interface/Omission
Interface/ % of Total	Interface/Total divided by Sum of COUNT of (FOUND records where E_TYPE = 'INTERFACE') and COUNT of (FAIL records where E_TYPE = 'INTERFACE')
Total/Commission	COUNT of (FAIL records where E_CAUSE = 'COMMISSION' and T_TYPE = 'TESTING')
Total/ % of Total Commission	Total/Commission divided by Sum of COUNT (FOUND records where E_CAUSE = 'COMMISSION') and COUNT (FAIL records where E_CAUSE = 'COMMISSION')
Total/Omission	COUNT of (FAIL records where E_CAUSE = 'OMISSION' and T_TYPE = 'TESTING')
Total/ % of Total Omission	Total/Omission divided by Sum of COUNT (FOUND records where E_CAUSE = 'OMISSION') and COUNT (FAIL records where E_CAUSE = 'OMISSION')

Total/Total	Sum of Total/Commission and Total/Omission
Total/ % of Total	Total/Total divided by Sum of COUNT of (FOUND records) and COUNT of (FAIL records)

Hours Worked By Week Ending Date Data Source

Report Field Name	Data Source
Week Ending	HOURS->WE_DATE
Member ID	HOURS->MEMBER
IP	HOURS->IP_HRS
BSM	HOURS->BSM_HRS
Verify	HOURS->VF_HRS
Certify	HOURS->CT_HRS
EC	HOURS->EC_HRS
Usage Model	HOURS->UM_HRS
Doc.	HOURS->DOC_HRS
Misc.	HOURS->MISC_HRS
Total	Sum of HOURS->IP_HRS HOURS->BSM_HRS HOURS->VF_HRS HOURS->CT_HRS HOURS->EC_HRS HOURS->UM_HRS HOURS->DOC_HRS HOURS->MISC_HRS

Hours Worked By Member ID Data Source

Report Field Name	Data Source
Member ID	HOURS->MEMBER
Week Ending	HOURS->WE_DATE
IP	HOURS->IP_HRS
BSM	HOURS->BSM_HRS
Verify	HOURS->VF_HRS
Certify	HOURS->CT_HRS
EC	HOURS->EC_HRS
Usage Model	HOURS->UM_HRS
Doc.	HOURS->DOC_HRS
Misc.	HOURS->MISC_HRS
Total	Sum of HOURS->IP_HRS HOURS->BSM_HRS HOURS->VF_HRS HOURS->CT_HRS HOURS->EC_HRS HOURS->UM_HRS HOURS->DOC_HRS HOURS->MISC_HRS

Verification Sessions Report Data Source

Report Field Name	Data Source
Verification Session	VERIFY->VF_NO
Ver No.	VERIFY->VER_NO
Date	VERIFY->VF_DATE
Errors Found	COUNT of (FOUND records where FOUND->VF_NO = VERIFY->VF_NO)
No. of Hours	VF_FN_TM - VF_ST-TM (Minutes should be converted (minutes/60) to get decimal representation before substraction.)
Found Per Hour	(Errors Found) divided by (No. of Hours)
Avg. Att	Sum of (ATTEND->PER_TIME/100), for each ATTEND record where ATTEND->VF_NO = VERIFY->VF_NO
No. of Manhours	(Avg. Att) multiplied by (No. of Hours)
Found Per Manhour	(Errors Found) divided by (No. of Manhours)

Average Number of Clear Box Lines Verified per Session:
(COUNT->NS->LINES) divided by (Count of VERIFY records)

Average Number of Clear Box Lines Verified per Hour:
(COUNT->NS->LINES) divided by (Total No. of Hours)

Average Number of Clear Box Lines Verified per Manhour:
(COUNT->NS->LINES) divided by (Total No. of Manhours)

Interface/ % of Total Omission	Interface/Omission divided by Sum of Interface/Omission and COUNT (FAIL records where E_CAUSE = 'OMISSION' and E_TYPE = 'INTERFACE')
Interface/Total	Sum of Interface/Commission and Interface/Omission
Interface/ % of Total	Interface/Total divided by Sum of Interface/Total and COUNT of (FAIL records where E_TYPE = 'INTERFACE')

Appendix C
List of Abbreviations

List of Abbreviations

ANSI	-	American National Standards Institute
API	-	Application Programming Interface
BCD	-	Binary Coded Decimal
BSM	-	Box Structure Method (subsystem)
C	-	C Code
CASEPM	-	Computer Aided Software Engineering For Presentation Manager
CDL	-	C Design Language
CDLP	-	C Design Language Preprocessor (subsystem)
Cleanroom	-	Cleanroom Software Development Method
CM	-	Certification Model (subsystem)
CMR	-	Certification Model Report (file)
CRE	-	Cleanroom Software Engineering Environment
CRM	-	Certification Reliability Model (file)
DB	-	Query Manager Database (file)
DBS	-	Database (subsystem)
DLL	-	Dynamic Link Library
DOS	-	Disk Operating System
ERR	-	C Design Language Error (file)
EST	-	Testing Matrix Estimation Report (file)
FRE	-	Analytical Results (file)
FTF	-	From-To-Frequencies (file)
FTP	-	From-To-Probabilities (file)
GUI	-	Graphical User Interface
MMN	-	Markov Model Notation (file)
MTBF	-	Mean Time Between Failures
MTTF	-	Mean Time To Failure
PI	-	Project Identifier (subsystem)
PLN	-	Planning (subsystem)
PLY	-	Policy (file)
QELIB	-	Q+E Database Library
SEQ	-	Test Case Sequences (file)
SER	-	Markov Usage Model Errors (file)
SQL	-	Structured Query Language
SYM	-	Symbol (file)
TCG	-	Test Case Generator (subsystem)
TMC	-	Testing Markov Chain (file)
TST	-	Testing Failure (file)
TTF	-	Time To Failure (file)
UME	-	Usage Model Editor (subsystem)
UND	-	Undefined Symbol Error (file)
VFQ	-	Verification Questions (subsystem)

VITA

Russell Lee Crisp was born in Cleveland, Tennessee on May 19, 1962. He graduated from Meigs County High School in Decatur, Tennessee in May 1980. The following September he attended Tennessee Technological University in Cookeville, Tennessee where he received the Bachelor of Arts degree with a major in Computer Science in May 1984.

After graduation, Mr. Crisp accepted a position as a computing analyst with Martin Marietta Energy Systems Inc. (MMES), operators of government installations for the United States Department of Energy in Oak Ridge, Tennessee. He was a member of the Engineering Management Information Systems (EMIS) section for eight years. As a member of the EMIS section, he worked primarily with IBM mainframe systems using a large IDMS database. He also worked for approximately two years in the PC environment where he designed and coded FoxBase applications.

In 1992, Mr. Crisp transferred to the Financial Data Collection and Management section where he accepted the Section Head position. His primary responsibility includes management of section personnel who are responsible for the Cost Accounting System (CAS) data collection applications. The Financial Data Collection and Management section primarily implements and maintains IBM mainframe systems utilizing the DB2 database architecture.

Mr. Crisp entered The Graduate School of the University of Tennessee, Knoxville in September 1988. He received the Master of Science degree with a major in Computer Science in December 1992.