AN ANALYSIS TOOL IN A KNOWLEDGE BASED
SOFTWARE ENGINEERING ENVIRONMENT

THESIS

David W. Fautheree
Captain, USAF

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

86 10 2 175

AFIT/GCS/ENG/86M-2

AN ANALYSIS TOOL IN A KNOWLEDGE BASED
SOFTWARE ENGINEERING ENVIRONMENT

THESIS

David W. Fautheree
Captain, USAF

AFIT/GCS/ENG/86M-2

AN ANALYSIS TOOL IN A

KNOWLEDGE BASED SOFTWARE ENGINEERING ENVIRONMENT

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science, Computer Systems

David W. Fautheree, B.S.

Captain, USAF

March 1986

List of Acronyms

AFIT	Air Force Institute of Technology
AFIT/ENG	AFIT School of Engineering, Department of Electrical and Computer Engineering
DBMS	Data Base Management System
DFD	Data Flow Diagram
DEC	Digital Equipment Corporation
HIPO	Hierarchy plus Input, Process, Output
ISL	Information Sciences Laboratory
KBSEE	Knowledge Based Software Engineering Environment
KBSMA	Knowledge Based Software Module Analysis tool
SADT	Structured Analysis and Design Technique
SDW	Software Development Workbench
SDWE	SDW Executive

List of Figures

AFIT/GCS/ENG/86M-2

Abstract

This thesis investigation presents the conceptual level
development of a knowledge based software engineering
environment. A variety of existing software tools are
integrated into the environment as well as newly developed
knowledge based tools, such as the software module analysis
tool designed and implemented for this project. The
environment is an extension of concepts from the AFIT
Software Development Workbench (SDW).

System development follows the software engineering
lifecycle of requirements analysis, design, implementation,
and operation as well as exploratory programming/rapid
protoyping techniques.

Table of Contents

I. Introduction

Thesis Objectives

The principal objective of this thesis effort is the conceptual level development of a knowledge-based software engineering environment. A variety of existing software tools should also be integrated. Another objective is the development an incorporation of the first knowledge-based tool. This tool should analyze software modules using the fundamental software engineering design principals of coupling and cohesion (DeMarco, 1979) (Peters, 1981) (Woffinden, 1985).

The requirements analysis, design, implementation, and operation of the Knowledge Based Software Engineering Environment (KBSEE) and the new analysis tool should be thoroughly documented.

Background

This section introduces the software development process, then provides a description and short history of AFIT's Software Development Workbench (SDW). Finally, knowledge based systems are introduced, providing the background on which the remainder of this investigation is based. Since a detailed description of these areas is well

I - 1

beyond the scope of this section, numerous references are provided.

The Software Development Lifecycle. The software development lifecycle has been characterized in many different ways (Myers, 1975) (DeMarco, 1979) (Peters, 1981). In this thesis investigation, the cycle is divided into five phases. They are the requirements definition phase, the design phase, the implementation phase, the integration phase, and the maintenance phase. Since software development is an evolving process, each phase receives feedback information from later phases as well as input from earlier phases. Hence, distinctions between lifecycle phases are not always entirely clear. Configuration control of all documentation at each phase in the lifecycle are essential to prevent confusion, especially in the maintenance/operation phase.

The requirements definition phase emphasizes what the system should do. In the design phase, requirements are assigned to various hardware and software components, which are then refined into interacting modules. During the implementation phase, the defined modules are written in a formal computer language and tested individually and as groups. In the integration phase, the hardware and software components are assembled into a system and are subjected to

testing as a whole. Finally, the system is used and modified as necessary during the maintenance phase.

For a detailed discussion of the requirements phase, see (DeMarco, 1979). The design phase is covered in (Myers, 1975) and (Peters, 1981). Various aspects of the the implementation phase are covered in (Aho and others, 1974), (Horowitz and Sahni, 1984), and (Wirth, 1976).

The lifecycle phases discussed above are based on the classical software engineering lifecycle model. Recently, rapid prototyping and exploratory programming approaches have been introduced (Sheil, 1983) and (Martin, 1985). Sheil and Martin advocate the use of powerful design tools which allow the software developer to quickly write and modify source code. In the rapid prototyping approach, the requirements and specifications are not rigidly determined before starting the design. General concepts are explored in small prototypes to show feasibility of implementation and correctness of design. From these explorations, requirements and designs are either proven or modified. Eventually, the entire system design is solidified. In short, the lifecycle phases evolve from the exploratory design/implementation phases.

The Software Development Workbench. The Software Development Workbench (SDW), which resides on the AFIT Information Sciences Laboratory (ISL) Digital Equipment Corporation (DEC) VAX-11/780 computer, was conceived and designed to help the software engineer manage the inherent complexity of developing computer software. The SDW consists of "an integrated set of automated tools to assist the software engineer in the development of quality and maintainable software" (Hadfield and Lamont, 1983:171).

The original work on the SDW was done by 2Lt Steven M. Hadfield for his master's thesis (Hadfield, 1982). In his thesis, 2Lt Hadfield provided motivation for the development of an interactive and automated software development environment. He maintained that such an environment should be integrated, traceable, flexible, and user-friendly. The original SDW was the result of his implementation efforts. Since that time, several AFIT graduate students have added additional tools to the original SDW system:

Rose, 1982

Gatewood, 1983

Thomas, 1984

Shomper, 1984

Moore, 1984

Wolfe, 1985.

<u>Knowledge-Based Systems</u>. The study of knowledge-based systems ("expert systems") is in part an outgrowth of artificial intelligence (AI), which involves the study of automated problem solving, construction of symbolic representation of knowledge, natural language communication, and machine learning. Knowledge-based systems are computer programs consisting of a knowledge base, situation data, and an implicit or explicit control structure (Harmon, 1985:49). The knowledge base consists of a knowledge representation scheme, usually in the form of production rules, semantic networks, frames, or a hybrid scheme. Problem solving knowledge and techniques are contained in the knowledge base. Situation data is stored in memory and contains information about the specific problem being solved. The control structure provides the reasoning mechanism (inference) to solve the problem. The general architecture is shown in Figure 1.1.

The six components on the left side of the figure reflect the capabilities for knowledge acquisition, debugging and experimenting with the knowledge base, running test cases, generating summaries of conclusions, explaining the reasoning that led to the conclusion, and evaluating system performance. The main computation engine is the search/inference component, which searches the knowledge base for
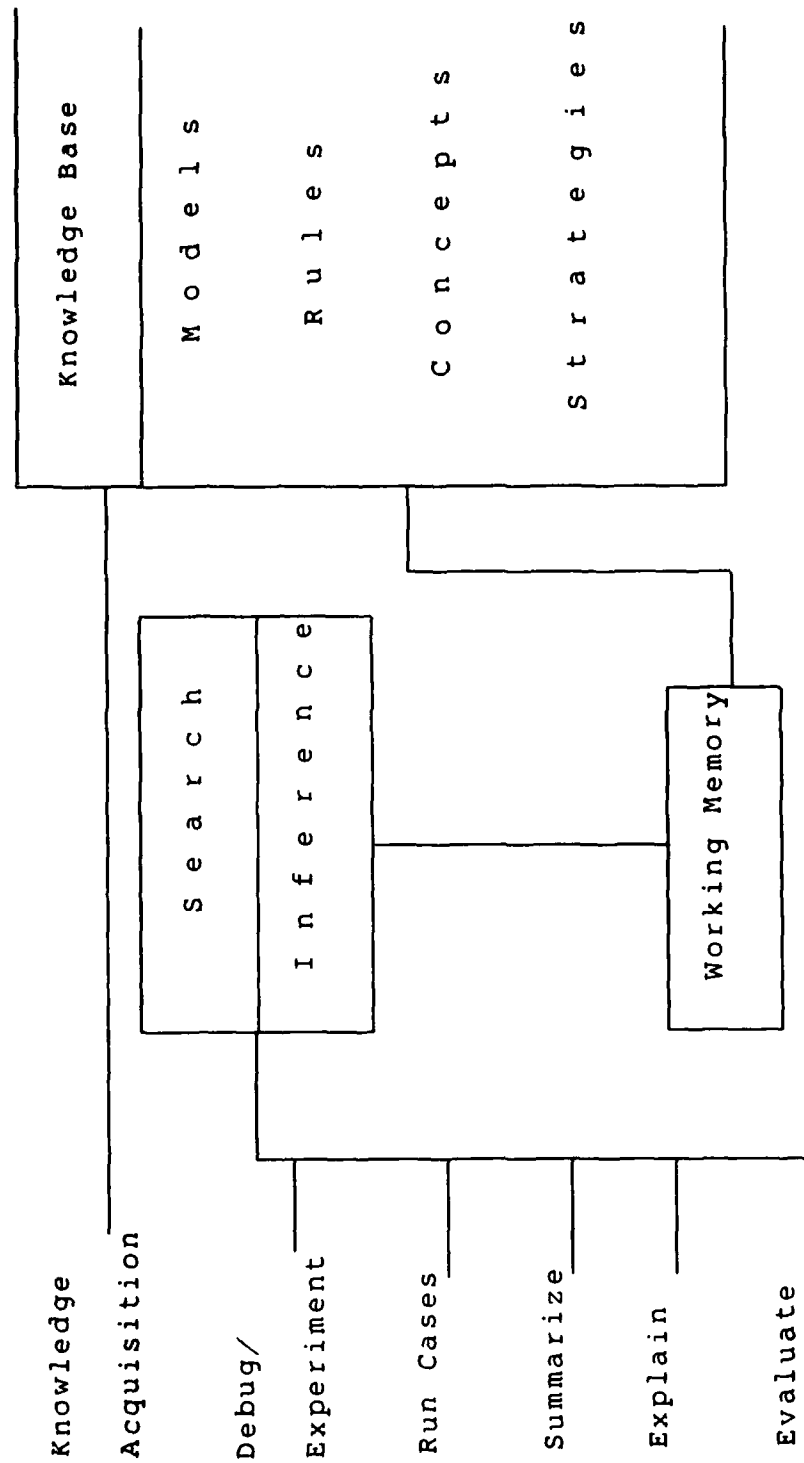
Figure 1.1 - A General Architecture of a Knowledge Based System

applicable knowledge and makes inferences based on current problem data stored in memory.

The knowledge base is the main repository for specific knowledge about the domain. Concepts are declarative representations of domain objects, with both abstract classes and concrete instances. Complex interrelationships are represented and used in making inferences and in constructing similarities.

Conceptual knowledge includes the basic terms of the problem domain. Rules are empirical associations linking: causes and effects; evidence and likely hypotheses; and situations and desirable actions. Models are collections of interrelated rules, usually associated with a particular problem hypothesis or overall diagnostic conclusion. Strategies are rules and procedures which aid the use of the rest of the knowledge base; i.e., guiding search and resolving conflicts when several equally plausible rules apply to a given situation. (Rychener, 1984).

Knowledge Based Software Engineering Environments

Software systems are increasing in size (measured in lines of executable code) at a rate considerably faster than programmer productivity (measured in lines of executable code produced per man-hour) (Myers, 1978). To combat this

software productivity problem, researchers are developing new architectures for software engineering environments using automatic programming and knowledge based system technology (Kinnucan, 1985) (Kowalsky, 1984) (Ramanathan, 1984) (Sheil, 1983) (Teitelbaum and Reps, 1981) (Waters, 1982) (Wess, 1984).

These new architectures use one of two paradigms: "program analysis" and "program synthesis" (Barr and Feigenbaum, 1982:295-379). In both paradigms, systems are represented in some formal language, with specified formal transformations on that representations. These transformations eventually produce executable programs. In the analysis paradigm, existing programs are examined to gain understanding of their overall function and the overall programming task is divided into elementary parts using the software engineering method of step-wise refinement. In the synthesis paradigm, the problem is formally specified in a very high order language. A major problem with the analysis method is that requirements change, causing repeated re-analysis. Programming in very high order specification languages is usually as least as difficult as programming in a high order programming language (Frenkel, 1985).

There are several texts introducing the field of Artificial Intelligence and knowledge based systems. Two useful

standard texts are (Nilsson, 1980) and (Rich, 1983). (Hayes-Roth, 1983) and (Forgy, 1984) discuss various expert system architectures and design methodologies.

Problem and Scope

The SDW tool set is not complete and none of the tools represent a knowledge-based architecture. Currently the only program specifically included in the SDW as a design tool is AUTOIDEF. AUTOIDEF allows one to create Integrated Computer Aided Manufacturing Definition (IDEF) models. AUTOIDEF is described in the Interim AUTOIDEF System User's Reference Manual (UM 170133010, 1982). One of the model types in AUTOIDEF, $IDEF_0$, can be used to create Structured Analysis and Design Technique (SADT) charts. Unfortunately, AUTOIDEF does not provide for any automated consistency checking. For more information about AUTOIDEF see (UM 170133010, 1982). (Peters, 1981:62-64) describes SADT.

Another tool provided by the SDW which can be applied during the design phase, but not specifically included as a design tool, is SYSFL (Mihaloew, undated). SYSFL is a graphics editor which provides standard flowcharting symbols as primitives. SYSFL can be used during the design phase to create structure charts and flowcharts.

None of the design tools currently in the SDW have

sufficient power to support the type of rapid prototyping and exploratory programming advocated by Sheil and Martin. The SDW does not currently fully support an integrated knowledge-based tool set. Furthermore, the SDW human-computer interface is a hierarchical menu/command interpreter system and often requires more keystrokes to use than to execute the tools directly from a single level menu/interpreter.

A new KBSEE and integrated set of tools would solve these problems and extend the SDW concepts in an entire new environment for future development. The knowledge-based software module analysis tool is a good example of the type of tool that can be developed in a KBSEE, a useful addition to the tool set, and a first step towards a complete, integrated, AI based software environment at AFIT.

Standards

Since this investigation emphasizes the use of a knowledge based software engineering environment and a tool for the analysis of software modules, the standard for determining success is whether or not the environment can be used effectively for software development and whether or not the tool correctly analyzes the modules and provides useful information to the software engineer. The tool uses data

elements identified as design elements in AFIT/ENG Development Documentation Guidelines and Standards (AFIT/ENG, 1984) as its primary source of information. However, these data elements do not provide sufficient information to allow the tool to complete its analysis. Efficient, effective interaction with outside elements, including the user, is an important consideration. If the tool easily obtains its required data, then this project should be considered successful. An efficient human-computer interface is an important consideration. Systems which process data efficiently and correctly tend to not be used if they are difficult or clumsy to use.

Approach

The development of the KBSEE and the analysis tool, KBSMA, follow the standard software development cycle described in the Background section. Exploratory programming and rapid prototyping also contribute to the success of this project, since these techniques allow various designs to be more quickly tested, evaluated, modified, abandoned, or adopted than the standard software development process normally allows. This increase in productivity is due to the power of employed software tools and the nature of the implementation languages, which allow separate development

and testing. The separate development and testing of small, exploratory systems directly result in more rapid design of the larger system. If a large system is developed using the conventional software engineering lifecycle approach, detailed design and implementation occur late in the cycle. Problems occurring this late in the development cycle result in major modifications of the entire system. In rapid prototyping/exploratory programming, concepts are adopted or rejected before the nature of the entire system is firmly set (Sheil, 1983). When a concept is adopted, the system is then optimized. Since the KBSEE with the KBSMA can be categorized as a reasonably large and complex software system, exploratory programming/rapid prototyping approaches can (and do) yield productivity benefits.

The development of the KBSEE is very closely related to the SDW. The goals are essentially the same: the development of an organized environment consisting of off-the-shelf software tools. The approach of the KBSEE differs from the SDW in the primary emphasis on the human-computer interface and the incorporation of knowledge based systems. The KBSEE uses principles discussed in (Hansen, 1971) which presents "User Engineering Principles for Interactive Systems" and (Teitelman, 1977), which discusses interactions through terminal displays. Other knowledge based environments

require the use of exotic hardware, i.e., Lisp machines, or exotic languages. The KBSEE approach is to help software engineers develop systems using tools and methods they are already employing.

Thesis Overview

System level requirements are defined, then the specific requirements of the KBSEE and the knowledge-based software module analysis tool are examined in Chapter 2. The design and implementation of the analysis tool is based on earlier design project for EENG 749, Advanced Topics in Artificial Intelligence. The design of the KBSEE and the analysis tool is described in Chapter 3. Chapter 4 discusses the tool's implementation. In Chapter 5, the implemented system is evaluated using the standards described in this chapter and recommendations for future investigation are provided. Complete documentation is included as appendices to this thesis.

II. <u>Requirements Definition</u>

<u>Introduction</u>

The objective of the requirements definition phase is to formalize what the system is to do into a concise, clear, and consistent statement (Peters, 1981). To accomplish this goal, the system must be viewed from three different perspectives: customer, user, and designer. The customer states the requirements by functional description of the task the system is to accomplish. The user states requirements in the form of system operations. The designer considers the views of both the customer and the user in the design specifications. In all three views, requirements are stated in some requirements definition language. The language may be graphical or lexical or a combination of both.

This chapter presents a broad functional requirements definition for a knowledge-based software engineering environment and a software module analysis tool in that environment. First, the overall requirements for the Knowledge Based Software Engineering Environment (KBSEE) are presented. Next, more detailed subsystem requirements are discussed. Finally, the detailed requirements for the analysis tool are described.

The requirements are presented in the form of data flow

diagrams (DFDs). DFDs were chosen for use in this inves-
tigation for their simplicity. Other representations, such
as Structured Analysis and Design Technique (SADT) charts
and Systematic Activity Modeling Method (SAMM) activity
cells (Peters, 1981:133-138), show more information, but are
more difficult to create and maintain. DFDs are part of the
structured analysis concepts developed by (Stevens,
1974:115-139). In structured analysis, DFDs are used to
develop a specification and a design. This method is very
widely used, with its popularity stemming from its ease of
use (Peters, 1981:139-148). The use of exploratory program-
ming and rapid prototyping in this investigation requires
simplicity in requirements definition since requirements and
conceptual designs are implemented before and during system
specification development. Simplicity in requirements
definition was considered more important than detail.

DFDs consist of four basic elements: processes, data
flows, data stores, and sources/sinks. Processes transform
data and are represented by circles. Data flows are paths
between elements are are represented by arrows. Data
stores, represented by line segments or parallel lines, are
files or data bases. Sources and sinks are entities outside
the system which originate and collect data, respectively.
Sinks and sources are represented by rectangles. For a

detailed description of the DFD language, see (DeMarco, 1979).

System Specifications Development

While the automatic programming research discussed in the previous chapter has produced interesting results, a system which can generate quality, general purpose software directly from requirements is beyond the current state-of-the-art. Requirements are stated in some representational language. The goal of an automatic programmer is to transform the requirements language into a system executable language. To accomplish this goal, the requirements language must either be represented in a language close to the implementation language, requiring relatively simple transformations, or be represented in a simpler requirements language, and require complex transformations. The first representation requires a notation that is nearly as complex as a programming language, such as C or Lisp, and is therefore on the same order of difficulty. The second representation requires a system of automatic transformations similar to those accomplished by an experienced system software engineer. If software development is a problem for human software engineers (Myers, 1975) (Boehm, 1976) (DeMarco, 1979) (Peters, 1981) (Sheil, 1983) (Martin, 1985), then

the development of software to automate software development would compound the problem, except in small, limited cases.

A more practical approach, using current technology, is to develop an environment which aids the software engineer develop software products using existing tools and methodologies. Software design would be done by humans, with the more mundane tasks being automated, i.e. entering operating system commands, remembering file names and their location within the file system. The environment would allow the human software engineer to devote nearly all attention on design, not minor project details. Knowledge based tools can also enhance productivity by using facts, rules, and models stored in the knowledge base to help the developer design consistent, quality code.

The initial specifications of the KBSEE are derived from analysis of the user requirements. One requirement is that the environment be used to develop software using existing tools. It is not reasonable to develop an entire tool set within the scope of a single thesis investigation. Also, users know and understand the interactions and operations of existing software tools. If the environment uses existing tools, then the user does not need to learn how to use the new tools as well as how to use the new environment.

Software development is a repetitive process. Software

modules are usually added to the system incrementally. First, the designed module is implemented by adding it to the source file. The calling module calls the new module using a prescribed interface. The source file is then compiled and linked to form an executable program. The program is then executed with a series of test situations to ensure that the newly implemented module works properly. If there are errors in the implementation of the module or its interface, the source file is modified with a corrected version of the module and the process is repeated. The user normally repetitively enters the same sequence of commands, i.e., EDIT FOO, COMPILE FOO, LINK FOO, RUN FOO. The environment should have a simple human-computer command interface that does not require the user to re-enter the same commands.

Software systems usually consist of many separate files. These files are usually organized in some manner within the file system. Most software developers organize their projects so that all required components are in the same portion of the file system. For example, a project may be located in a file directory with subdirectories for source modules, data files, and executable files. Software engineers should not have to devote time managing project files. The environment should keep track of what project

the user is currently developing and where the associated files are located.

The environment should be as simple in organization as possible, but convey a large amount of information, i.e., available tools, environment status, current project information, and user interaction.

This section describes such an environment in terms of its functional requirements: the Knowledge Based Software Engineering Environment (KBSEE) developed for this thesis effort. Figure 2.1 shows the top level Data Flow Diagram (DFD) for the system. This figure illustrates the scope of the KBSEE: it accepts command and data from the user and produces some software product. Figure 2.2 shows the decomposition of the KBSEE into major subsystems, the project manager, the display manager, the command interpreter, and a set of software tools. These major subsystems correspond to the broad requirements stated above. The remainder of this section discusses each of the subsystems in turn. Detailed requirements for the software module analysis tool are presented as a subsection of the software tool set subsystem.
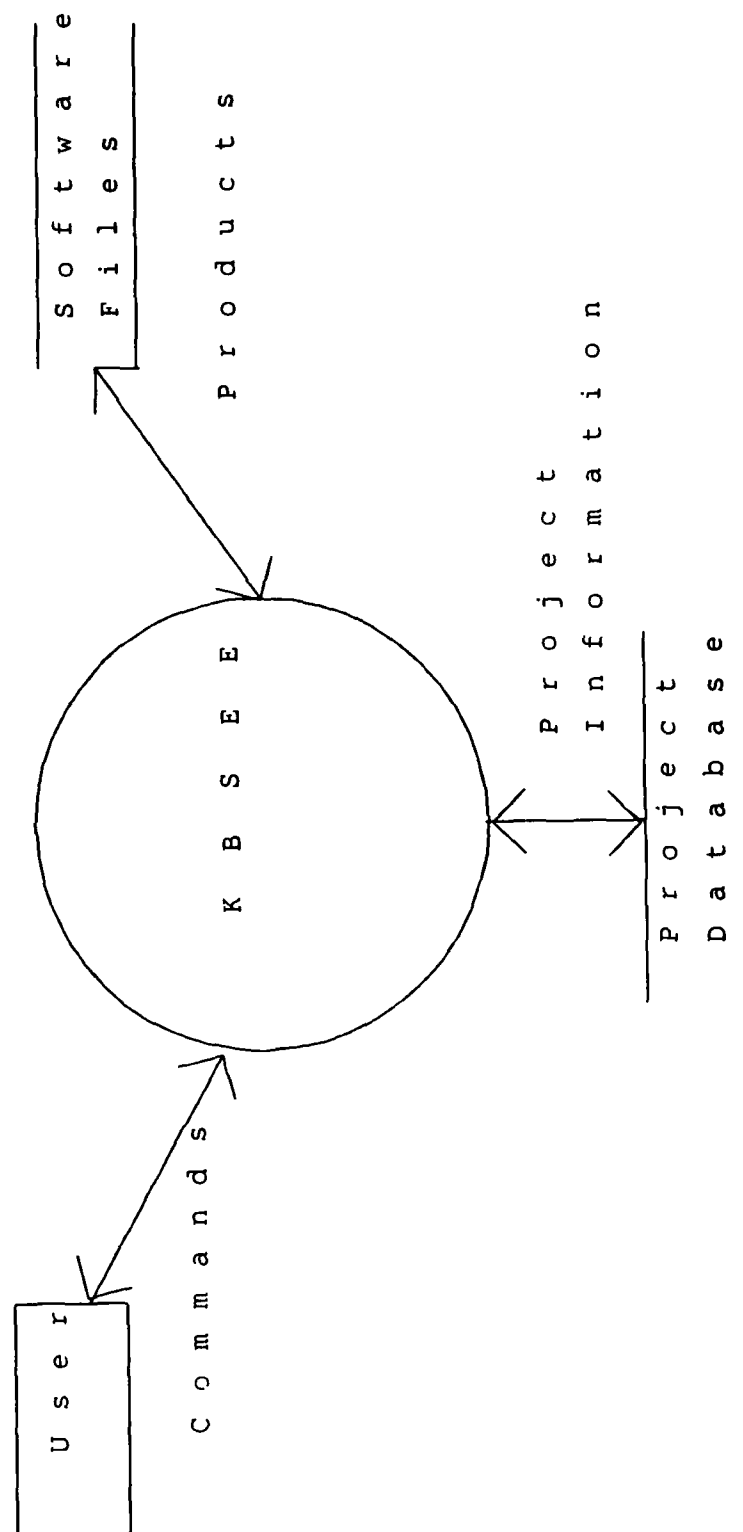
Figure 2.1 - KBSEE Top Level DFD
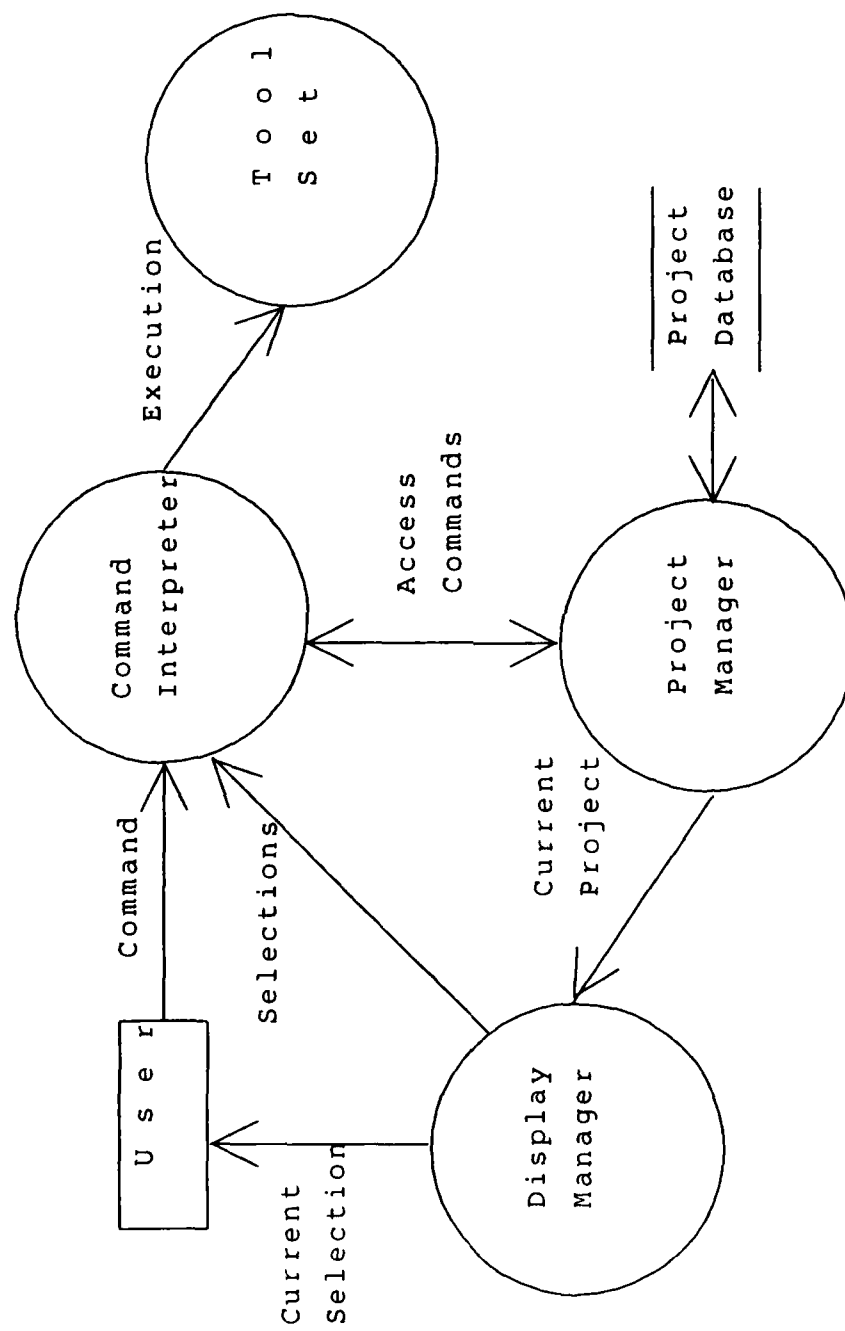
Figure 2.2 - KBSEE Major Subsystem DFD

Project Manager Design Specifications

The project manager maintains the project database, which keeps track of the location and names of files used in various software development projects and maintains a profile of the user's preferred tools and commands. The project manager's functions allow the user to not have to repeatedly re-enter commands and remember locations of project files. Since software engineers usually develop systems over a period of time ranging from hours to years, the project manager must store the information in a fairly permanent medium. The project manager must load information from the permanent medium into working memory, where it can be used by other KBSEE functions. Figure 2.3 illustrates the data flow for the project manager, which consists of four subfunctions: LOAD PROJECT, SAVE PROJECT, LOAD PROFILE, and SAVE PROFILE.

When a save project command is entered by the user, the SAVE PROJECT subfunction stores the current working project information into a user's permanent, centralized project database. This is necessary so that the user does not have to specify filenames and locations each time the environment is used.

Once the project information is stored into the project database, a method for retrieval is necessary. The LOAD
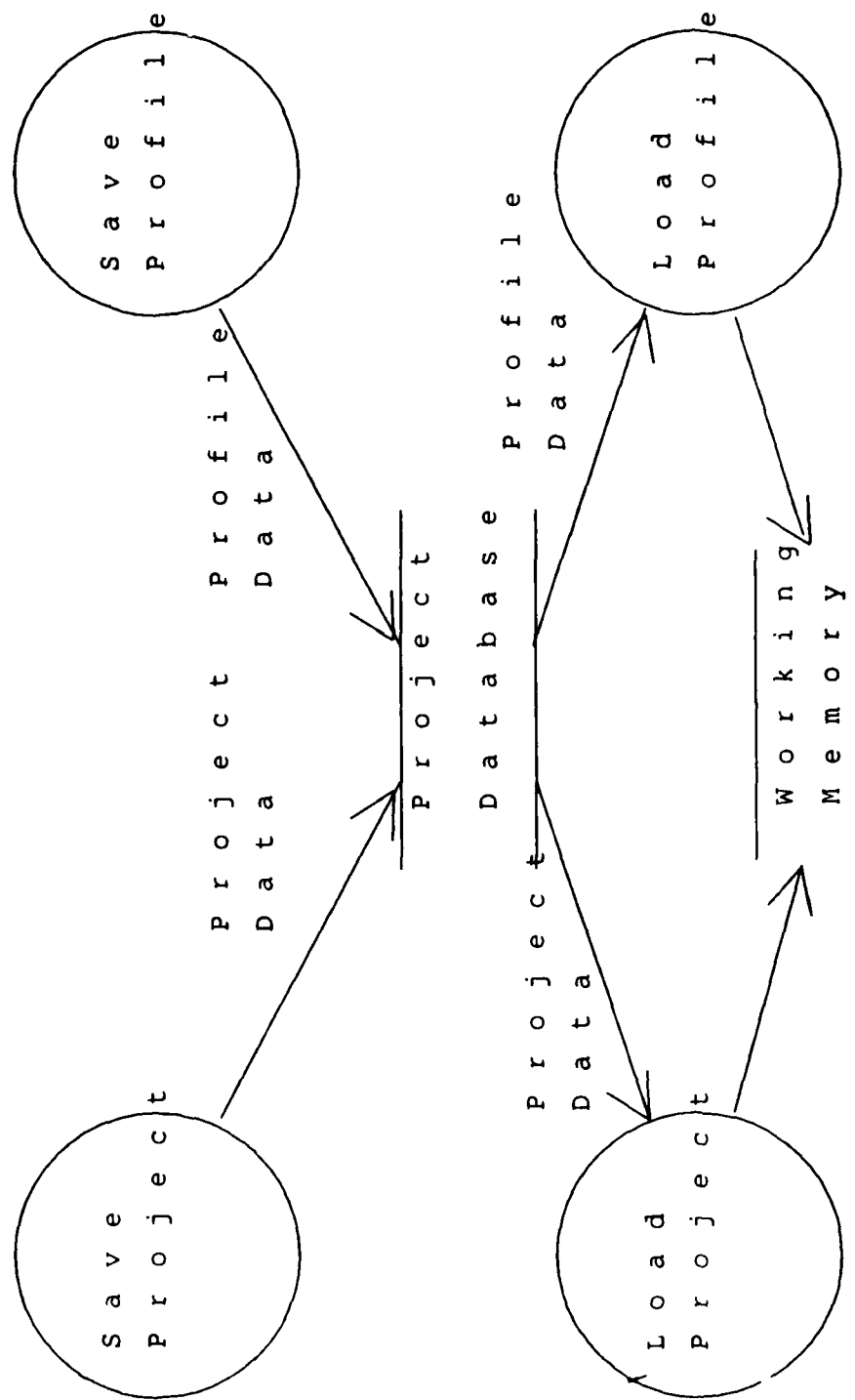
Figure 2.3 - KBSEE Project Manager DFD

PROJECT function provides this method. If another project is already in working storage, it is overwritten by the new project.

The LOAD PROFILE and SAVE PROFILE subfunctions work similarly to LOAD PROJECT and SAVE PROJECT, except they maintain information about specific user preferences, such as which editor or compiler to use and what commands are used to build an executable software product. These two functions allow the user to not have to repeatedly re-enter operating system commands. SAVE PROJECT and LOAD PROJECT save the information into a permanent local file, rather than a centralized database.

Display Manager Design Specifications

The display manager organizes and maintains the terminal screen of the KBSEE. This function is extremely important in an interactive environment (Teitelman, 1971) (Hansen, 1971). The display manager displays the set of available tools, presents information about the current project, and provides positive feedback to the user about the environment status. The display manager also provides a medium for user interaction with the KBSEE.

Command Interpreter Design Specifications

The command interpreter controls the overall execution of the KBSEE. Figure 2.4 depicts the data flow for the command interpreter. As is shown in the diagram, the interpreter has two subfunctions: GET COMMAND and PROCESS COMMAND.

When the user enters a command, the GET COMMAND routine validates the command. If the selection can be correctly interpreted by the GET COMMAND process, the PROCESS COMMAND routine is called. This routine controls the actual execution of the user's selected KBSEE routine.

Tool Set Requirements

Tool sets consist of a wide variety of tools. Conventional tools (i.e., compilers, editors, linkers) have been available for many years and are well understood. They are generally available off-the-shelf from the computer manufacturer or a software vendor. Knowledge based tools have been introduced comparatively recently, usually in a research environment. Knowledge based systems in general have only recently been in widespread use in industry (Harmon, 1985), which accounts for their limited application. Knowledge based tools are usually customized for the particular application for which they are to be used and for the environment in which they will execute.
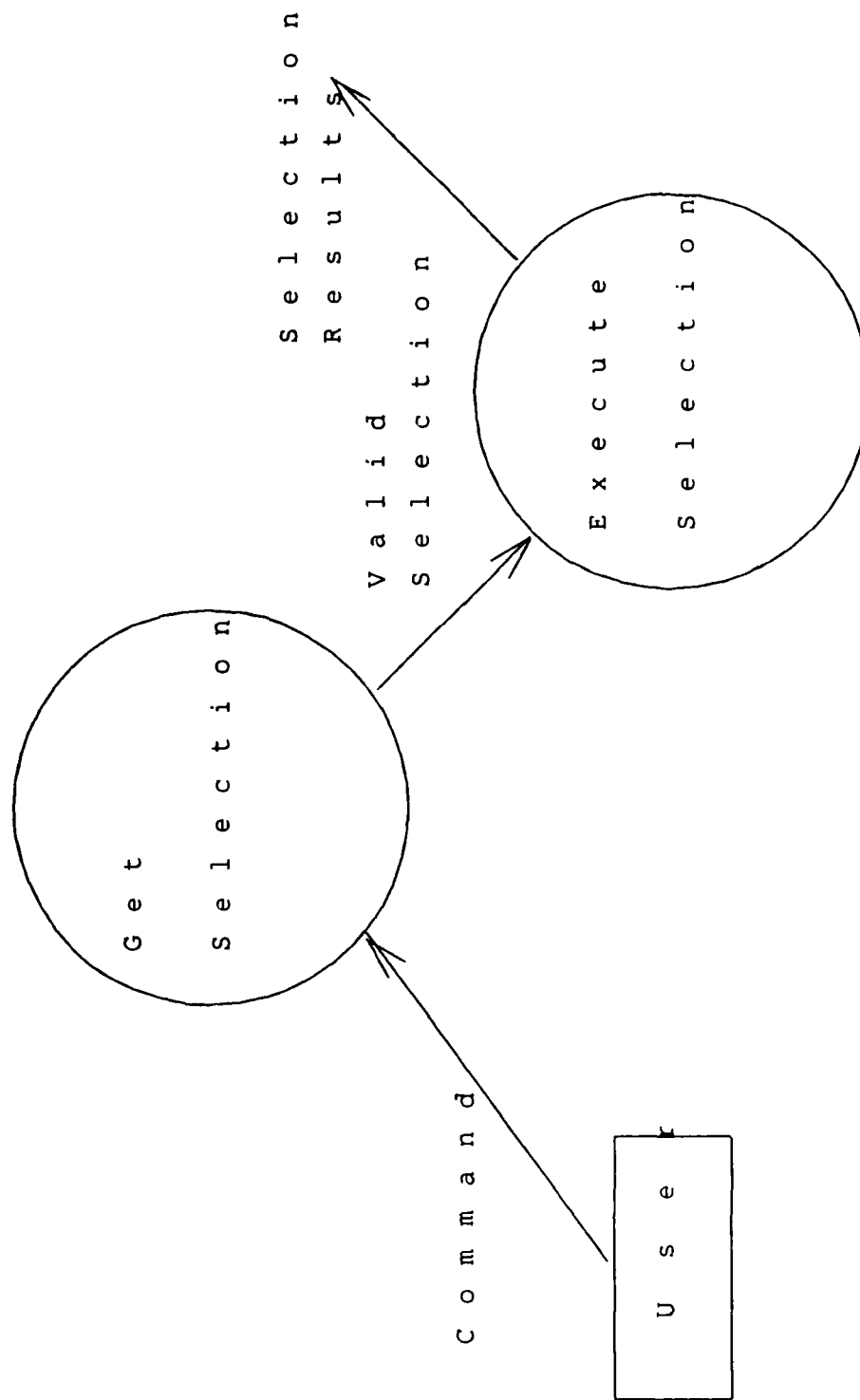
Figure 2.4 - KBSEE Command Interpreter DFD

The selection of the tool set should result in suffi-
cient tools to generate executable code. The tool set
selection should also be based on established, conventional
concepts as well as knowledge based methods. The SDW
provides a useful analysis of tool set requirements
(Hadfield, 1982).

An optimal tool set would be one that generates
executable code and provides tools for documentation, all
with minimal redundancy of effort. Which tools comprise an
optimal tool set is an issue currently unresolved
(Woffinden, 1985). Indeed, even the selection of a
minimally sufficient set of tools is the subject of some
controversy in the Ada environment (AJPO, 1980).

Figure 2.5 shows the DFD for a "typical" tool, the
compiler. The tool accepts input from some source, in this
case a programming language source code file, and transforms
it into some form of output, such as an object code file.

Figure 2.6 presents the DFD for a knowledge based tool:
the knowledge based software module analysis tool, KBSMA.
The analysis tool obtains information about the software
module to be analyzed from available sources. This
information is in the form of module and variable data
dictionary entries. Information about the module is then
placed in memory. The inference mechanism searches the

knowledge base for any applicable production rules. If any inferences can be made, the tool then produces its analysis of the current software module.

Conclusion

This chapter has presented the results of the functional requirements analysis and design specification phase of this investigation. Since much of the requirements were presented in the form of data flow diagrams, the symbology of DFDs were discussed. The next chapter describes the design phase of this thesis effort.
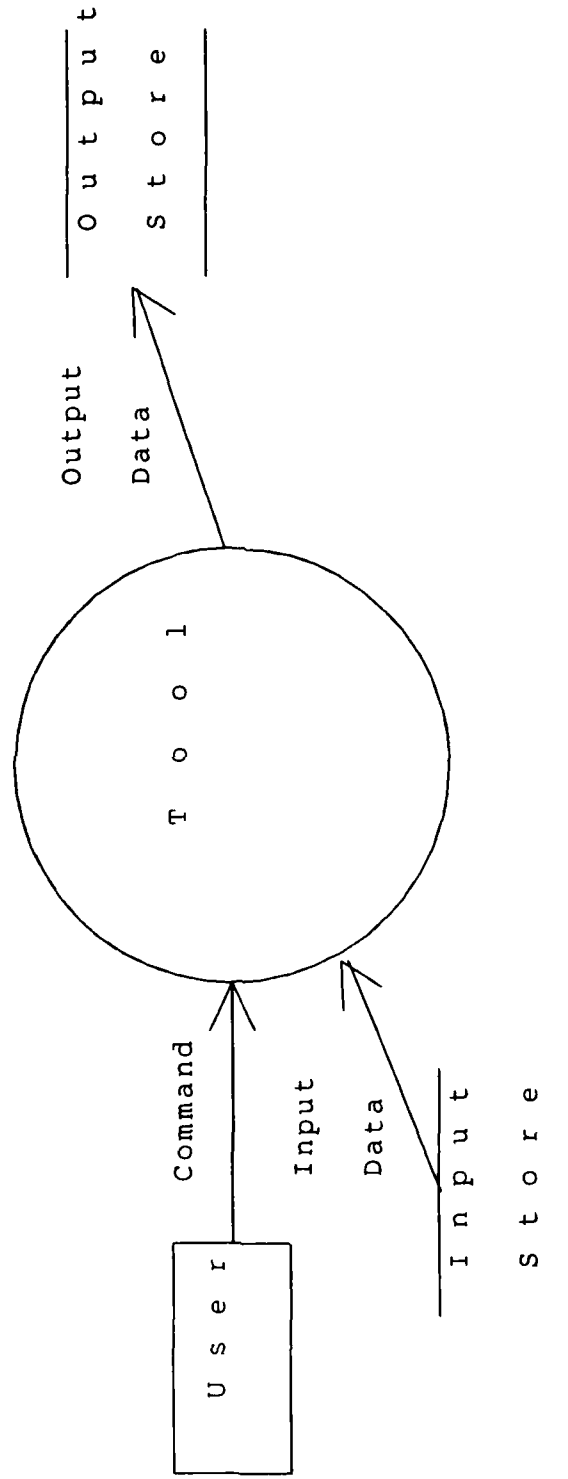
Figure 2.5 - DFD for Typical KBSEE Tool

III. <u>Design</u>

<u>Introduction</u>

In this chapter, the design of the Knowledge Based Software Engineering Environment, KBSEE, and the Knowledge Based Software Module Analyzer, KBSMA, are described and justified. The description begins with the overall functional system design. Then, the design is decomposed into functions of increasing detail. Detailed descriptions at the lowest level of KBSEE's design and KBSMA's design are given in Appendix B, Structure Listing - KBSEE.

The system-level design is presented in the form of a data flow diagram to maintain a consistent format between the requirements definition/system specification phase and the design phase. The lower level designs are in the form of structure listings, included as Appendix B. Structure listings show the overall hierarchical structure of the design. Structured listings are simpler in layout and more concise than structure charts, while conveying the same information. Also, they can be easily generated by an automated tool available on the host computer system. Details, such as passed parameters and data types are shown in the data dictionary entries of Appendix A. Structure charts usually accompany Data Flow Diagrams when using the

structured analysis design methodology (Peters, 1981:139). Other notations which can be used to document software design include Leighton Diagrams, structure charts, and HIPO charts. These notations often include redundant details already shown in the Data Dictionary and are generally not used in conjunction with DFDs. Changing notation and methodology in the middle of the software development lifecycle could allow inconsistencies when one design language is translated into another. These other methods are discussed in (Peters, 1981:44-62).

System Design

The system level design of KBSEE is based on the results of the requirements definitions/system specification phase. During this phase, the system was decomposed into three functional elements: the Project Manager, the Command Interpreter, and the Tool Set. The basic design is shown in figure 3.1, with elements DISPLAY MANAGER, COMMAND INTERPRETER, and TOOL SET. The Project Manager consists of the centralized project database, the user profile store, the current project save store, and ACCESS processes for each of the stores. The Tool Set consists of executable programs and associated data stores.

The COMMAND INTERPRETER reads input commands from the
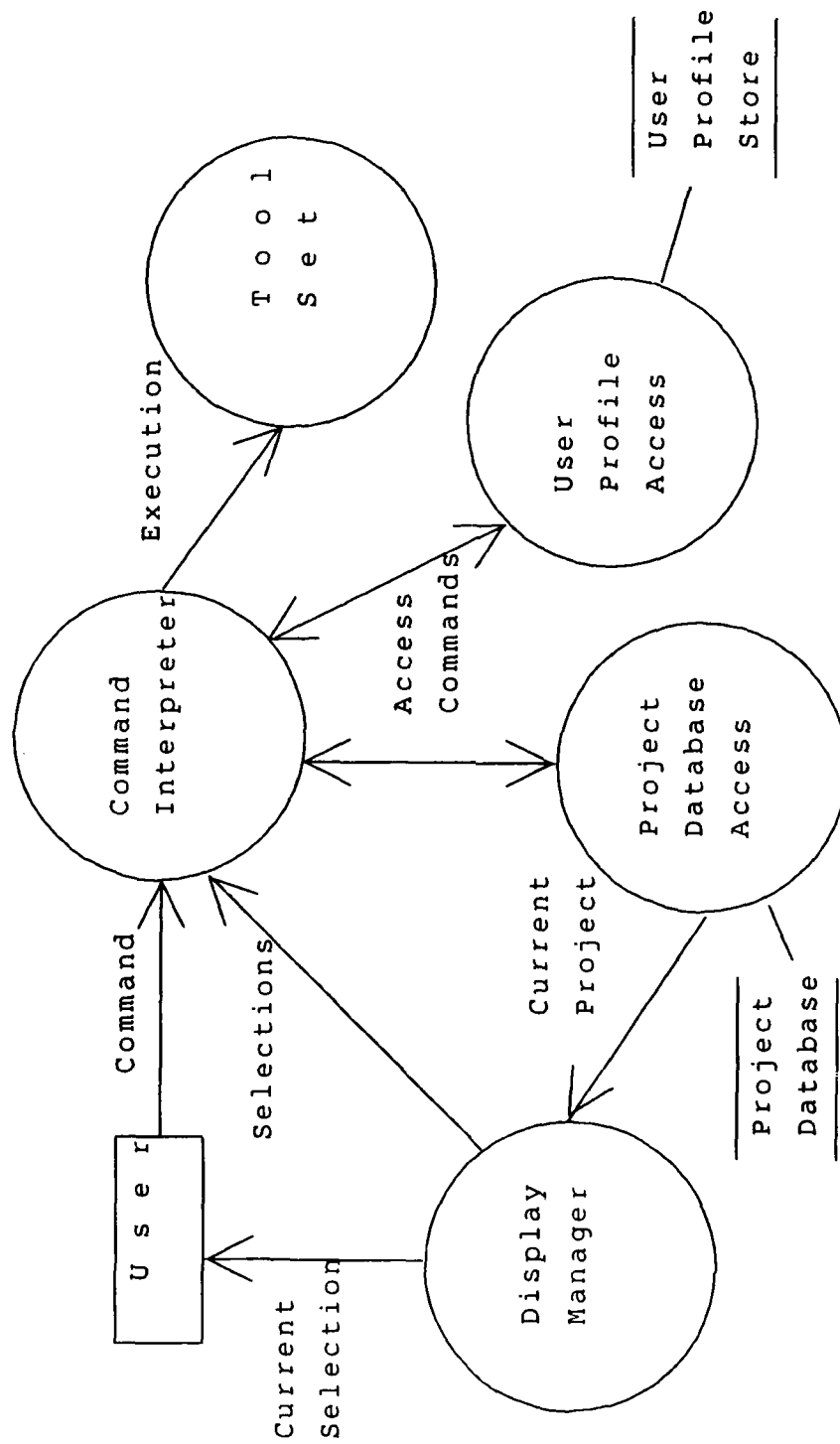
Figure 3.1 - DFD for KBSEE System Design

user, then invokes the DISPLAY MANAGER to output the user's current selection from a list of valid choices. The DISPLAY MANAGER also obtains the current project and workfile from the PROJECT MANAGER and loads the user profile for the current workfile. Having separate profiles for each workfile allows multiple workfiles to be used within a single project. Since there may be several executable program files within a single project, each workfile should have its own profile store for its appropriate edit, compile, or link command. If the workfile is a text file, then the edit command would contain the appropriate text processing utility command and the compile and link command portion of the store would be empty.

Once the user selects the EXECUTION COMMAND, the COMMAND INTERPRETER uses the profile information to execute the selected tool from the TOOL SET with minimal user intervention. Depending upon the selection, the tool may access its own local storage during tool use.

Display Manager Design. The design of the terminal display is based on concepts from (Teitelman, 1971) and (Hansen, 1971). The key principles from Hansen are:

    1) Use selection, not entry.

    2) Use names, not numbers.

    3) Display inertia.

4) Organize commands.

5) Rapid execution of common operations.

To accomplish principle 1, the terminal displays a menu of choices, with the current choice highlighted. The user moves the highlight to the desired choice and selects that item. To accomplish principle 2, the menu choices are the names of the items, not numbers; i.e., if the current selection is EDIT, then EDIT is displayed, not an arbitrary number from a list of choices.

To accomplish principle 3, the basic layout of the display remains the same in all operations. The user always knows what choices are available from a single screen. The user does not have to select a submenu to execute a desired tool. This also prevents having to exit a submenu, return to the main menu, and select another submenu to execute two different tools from different tool categories. This submenu/main menu/submenu interaction is reflected in the current SDW. Display inertia is also maintained in the KBSEE by using multiple windows. A window is a portion of the screen that can be accessed as a single entity. The concept of multiple windows originated with XEROX Company's Palo Alto Research Center (PARC) (Teitelman, 1971). A multiple window system allows essential information to be displayed dynamically on the screen in a window, without changing other windows on other parts of the display screen.

Principles 4 and 5, the efficient organization of commands and rapid execution of common commands, respectively, are accomplished by placing the most commonly used tools (editor, compiler, linker) at a location in the menu where they can be selected with a minimum of user interaction. Since most developers execute an edit, a compile, and a link in succession, more rapid execution can be gained if a single command accomplishes all three in sequence and if the KBSEE makes this command the current selection upon entering the environment.

Figure 3.2 shows the layout of the KBSEE display. The display is designed in accordance with the user engineering principles discussed in the previous paragraphs. The upper portion of the screen is divided into four windows which are used to list selections. The Defaults window displays the current project, workfile, and location within the file structure. The Status window gives the user positive feedback about the process the KBSEE is currently executing. Messages to and from the KBSEE are input and output in the Messages window. The initial display and update of these windows is the primary purpose of the display manager.

Since data can be read directly from the windows, the display manager is also invoked by the command interpreter for reading and highlighting the current menu selection.

```
------------------------------------------------------------------
|                          Main Menu                             |
|    KBSEE - A Knowledge Based Software Engineering Environment   |
|                                                                |
|    Build Program    Work File       Debugger      Spawn to CLI |
|    Edit             User Profile    Analyzer      Exit         |
|    Compile          Introduction    Librarian                  |
|    Link                             Printer                    |
|    Run                              Text Formatting            |
|    Display Errors                                              |
|    Start Project                                               |
|    Select Project                                              |
|    List Projects                                              |
|    Display Project                                             |
|                                                                |
|                                                                |
|          Defaults                ><            Status        > |
|PROJECT:                                                        |
|WORK FILE:                                                      |
|DIRECTORY:                                                      |
|                            Messages                            |
|                                                                |
|                                                                |
|                                                                |
------------------------------------------------------------------
```

Figure 3.2 - KBSEE Terminal Display Layout

This method is used because of the minimal number of user keystrokes involved in its use.

Display Manager Data Structures. The display manager uses several data structures in the maintenance and operation of the windows. The primary data structure in the menu_data structure, which maintains the current vertical and horizontal cursor positions, the maximum and minimum vertical positions, and the current menu item. This structure is used only for windows used for displaying menus. The display manager uses the structure to read the menu selection directly from the display. The maximum and minimum vertical values prevent attempts to read above or below the window. For example, if the menu has only two selections, an attempt to move the cursor to a position above or below the two selections would be prevented by a comparison to values in the data structure. By design, this is prevented by "scrolling" to the appropriate maximum or minimum value (i.e., if the current selection is the one at the top of the menu and the user attempts to move the current selection up, the display manager sets the current selection to the one at the bottom and vice-versa).

The display manager also uses a data structure to store the current menu. Since the design uses four menus, the display manager needs to keep track of which one is cur-

rently in use. Whenever the user moves to a different menu, the value of current menu structure is updated.

Command Interpreter Design. The primary function of the command interpreter is to translate commands from the user into a form appropriate for use by the KBSEE. To accomplish this task, it must compare user inputs in the current keystroke data structure to a set of valid user commands. One of Hansen's user engineering principles is to minimize memorization. The more words there are in a command language, the more words the user has to remember. Since the KBSEE is designed to use selection, rather than entry, there should be a single command for selection. The simplest command consists of a single entity. Therefore, the selection command should be a single keystroke. Other commands are necessary to change the current selection, request help, and remove the current window. These commands should also be specified by a single keystroke. A table of the KBSEE command language is shown in Table 3.1 below:

```
+-----------------------------------------------------------------+
|   Key                 Action                                    |
|   UP Arrow            Move to selection above current one       |
|   DOWN Arrow          Move to selection below current one       |
|   LEFT Arrow          Move to menu left of current menu         |
|   RIGHT Arrow         Move to menu right of current menu        |
|   Select              Select current menu item for execution    |
|   Remove              Remove menus and exit system              |
|   Help                Select help                               |
+-----------------------------------------------------------------+
```

Table 3.1 - Keyboard Command Map

Other keyboard inputs are ignored by the interpreter. As discussed in a previous section, the interpreter interacts with the display manager for menu and menu selection information. When the user invokes the Select keyboard command, the current menu item field in the menu data structure is passed to the command interpreter for execution.

Project Manager Design

As shown in Figure 3.3, the project manager maintains three data stores, a centralized database of all projects being developed by a user, a user profile for each project work file, and a current project save store. When the KBSEE is initially invoked, the project manager LOAD PROJECT function is invoked. The LOAD PROJECT function attempts to read the current project save store. If there is no current project save store (as is the case for a new user), the user must supply project information to the project manager. If the save store is present, the current project name and current work file name are loaded. The LOAD PROJECT function then accesses the project database for the location of the current work file. This process also validates the integrity of the database and save store. The LOAD PROJECT function then sets the KBSEE current location within the file system to the location obtained from the project
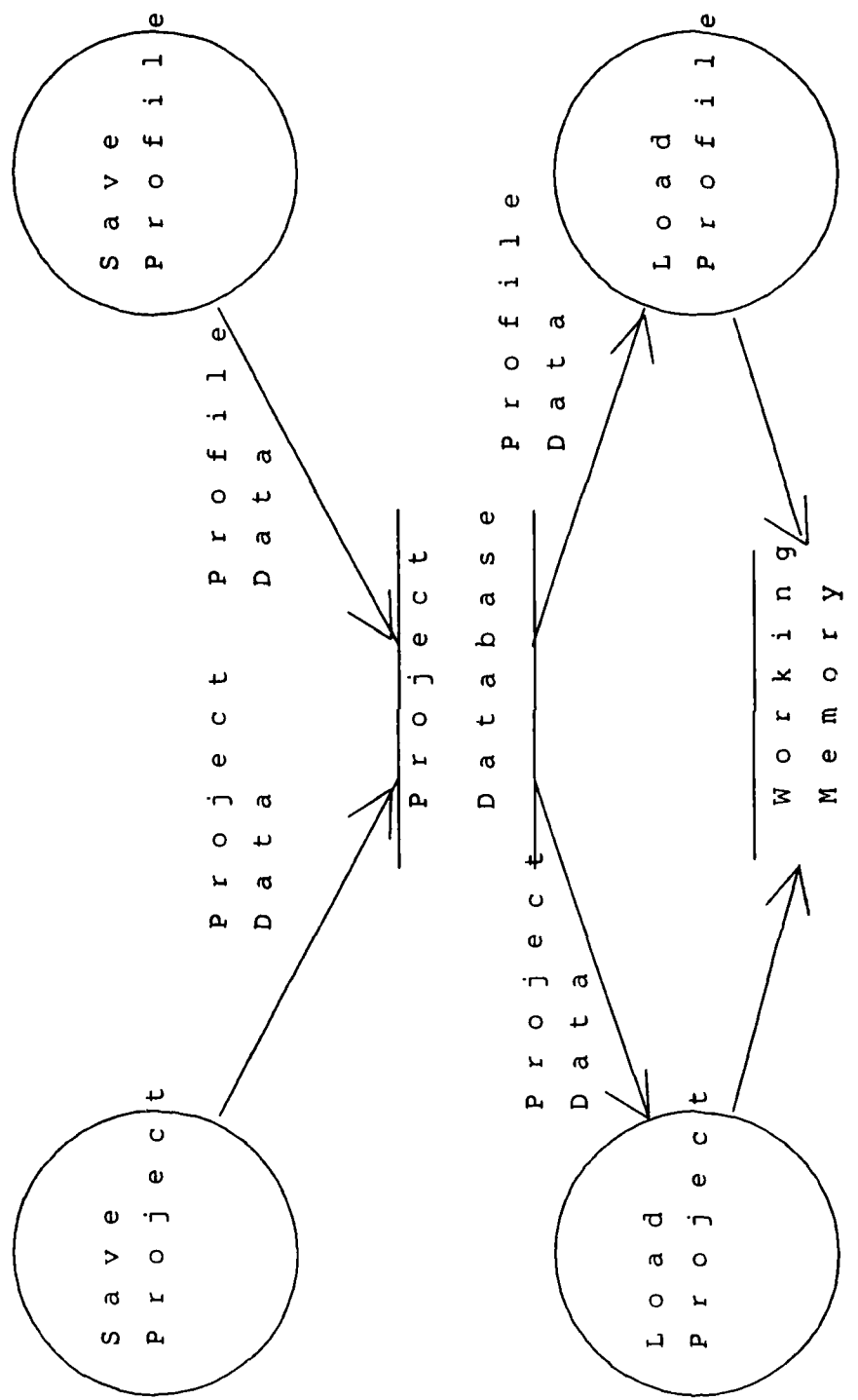
Figure 3.3 - KBSEE Project Manager DFD

database. This ensures that all the files associated with a project are co-located. Access functions to the user profile store and the project database are provided through KBSEE menu item selections. These selections are: Start Project, Select Project, List Projects, Display Project, Change Work File, and Change User Profile. The access function SAVE CURRENT PROJECT is invoked automatically when the user exits the KBSEE. This allows the project manager to automatically establish the project, work file, location, and profile whenever the user invokes the KBSEE.

Project Manager Data Structures. The project database consists of records as shown below in Figure 3.4:

```
+--------------------------------------------------------------+
| Project Name | Location | Work File Name                     |
+--------------------------------------------------------------+
```

Figure 3.4 - Project Database Record

For multiple work files within a project, the record is repeated with the same project name, but the location and work file name may be different.

The project manager also maintains a store for user profile information. The record format used by this store is shown below in Figure 3.5:

```
+--------------------------------------------------------------+
| Edit Command | Compile Command | Link Command                |
+--------------------------------------------------------------+
```

Figure 3.5 - User Profile Record

The project save store record format is shown below in Figure 3.6:

```
+-------------------------------------------------------+
| Project Name | Work File Name                         |
+-------------------------------------------------------+
```

Figure 3.6 - Project Save Store Record

Analysis Tool Design

Figure 3.7 shows the design of the analysis tool, KBSMA, which follows the general architecture described in Figure 1.1. (Harmon, 1985:178) characterizes the development of small (less than 200 rules) knowledge based system as a sequence of six steps:

1) Select a tool and make an implicit commitment to a particular paradigm.

2) Identify the problem and analyze the knowledge to be included in the system.

3) Design the system. Typically, this involves drafting a few rules.

4) Develop a prototype and test it.

5) Expand, test, and revise the system until implementation is complete.

6) Maintain and update system as needed.

Most existing tools for developing knowledge based systems (i.e., OPS-5, M.1, S.1, KEE, ART) handle the most common paradigm: diagnosis and prescription. (Harmon, 1985:92-133) contains an excellent survey of commercial tools and languages for developing knowledge based systems.
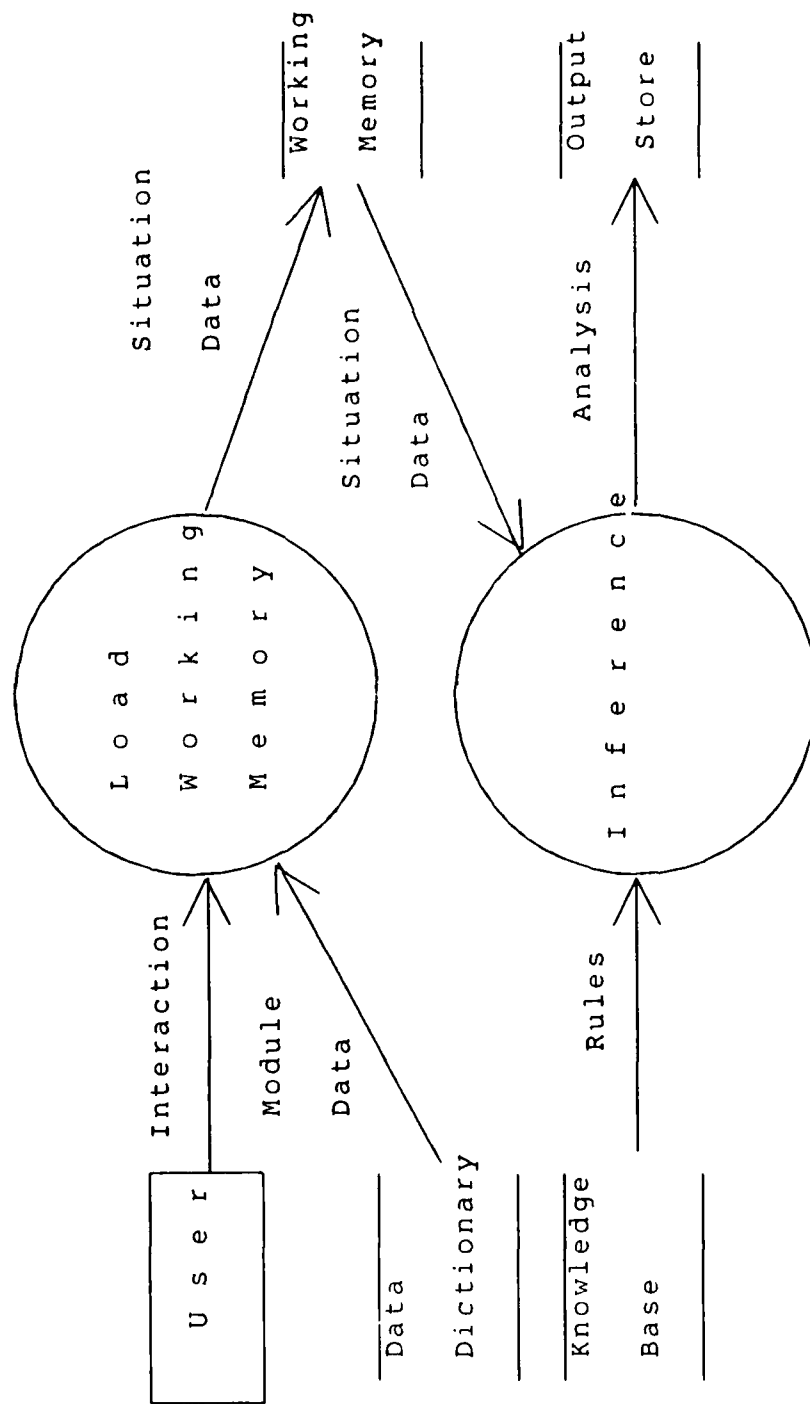
Figure 3.7 - DFD for KBSMA

The KBSMA problem area is of the diagnosis/prescription type. Software modules are diagnosed and recommendations for improvement are prescribed. This categorization implies that the KBSMA can be designed without regard to the actual implementation language, which will be a representation scheme for rules in some form. The next step in the design of the KBSMA is the design of the knowledge base.

Knowledge Base Design. The development of a knowledge base often requires the skills of a knowledge engineer. A knowledge engineer is essentially a person skilled in the translation of rules, principles, and models used by a domain expert into some knowledge representation scheme. Whether or not a knowledge engineer is required in the design and implementation of a knowledge based system depends upon the communication skills of the knowledge source (domain expert), the technical abilities of the system sponsor/system developer, and the relative difficulty in representing the domain knowledge in a form usable by the target knowledge based architecture. In the case of the KBSMA, the knowledge base was developed entirely without the aid of a knowledge engineer. There is no need for an intermediary between the domain expert, the system developer, and the customer, when all these functions are performed by the same person. A knowledge engineer would not have con-

tributed to the development of the KBSMA; in fact, filtering information through a knowledge engineer would have hindered the effort, rather than helped.

In order for the KBSMA to perform its function, information about the software modules to be analyzed must be made available to the inference mechanism. The best source of this information is a centralized data dictionary. Since the development of a data dictionary system is beyond the scope of this investigation, the information is provided manually in a format identical to the AFIT/ENG Software Development Guidelines and Standards, Data Dictionary Entry for a Module and used in SDW data dictionary research efforts (Thomas, 1984) (Wolfe, 1985). This format is implemented in the module data structure. The module data structure contains the following information:

        Module name
        Project name
        Module number
        Description
        Passed variables
        Return value
        Global variables used
        Global variables changed
        Files read
        Files written
        Calling modules
        Modules called
        Version
        Date
        Author
        Filename
        Coupling type
        Cohesion type

Recommendation

The fields for the coupling, cohesion, and recommendation are not from the data dictionary entries, but are used to hold this information once they have been determined by the inference mechanism.

Since the AFIT/ENG Data Dictionary Entry for a Data Element does not provide all the information necessary for determining coupling and cohesion, another data structure is necessary. The variable data structure contains fields for the name of the variable, its type, and whether or not it is a control variable. The control field must be provided by the user since there is no way to directly infer whether or not a variable is used for control by analyzing the data dictionary entries.

Small prototypes of the KBSMA showed that coupling can be inferred automatically from the data dictionary entries. However, cohesion cannot be determined by examination of the data dictionary entries. Ideally, cohesion should be deduced automatically from a description of the function of the module (i.e., "Natural Language" interpretation) or directly from the code (i.e., an intelligent interpreter or overall program structure analyzer). Since the development of either of these are beyond the scope of this thesis effort, the KBSMA prompts the user until module cohesion is determined.

Production Rules. The design of the production rules for determining coupling and cohesion is discussed in this section.

Coupling - There are four types of coupling: Data, Stamp, Control, and Common. The rules for determining each type are discussed in the following sections.

Common Coupling

IF
> The module coupling type has not been determined
> AND
> A global variable is used

THEN
> Set the coupling type to Common
> AND
> Recommend passing the data item as a parameter.

Control Coupling

IF
> The module coupling type has not been determined
> AND
> The module passes a parameter
> AND
> The parameter is a control variable

THEN
> Set the coupling type to Control
> AND
> Recommend modifying the module to not use imported control information.

Stamp Coupling

IF
> The module coupling type has not been determined
> AND
> The module passes a parameter
> AND
> The parameter is not of type Record
> AND
> The parameter is not a control variable

THEN
> Set the coupling type to Stamp
> AND
> Recommend passing only required data item.

Data Coupling

IF

The module coupling type has not been determined
AND
The module passes a parameter
AND
The parameter is of type Primitive
AND
The parameter is not a control variable

THEN

Set the coupling type to Data
AND
Recommend no improvement necessary.

<u>Cohesion</u> - Four questions are sufficient to determine module cohesion. The answers to these questions are stored in the cohesion answers data structure.

<u>Cohesion Questions</u> - These rules prompt the user for the information required to determine cohesion.

One Function

IF

The module cohesion type has not been determined

THEN

Ask if the module is performing only one function
AND
Place answer in the one function field

Related Activites

IF

The module cohesion type has not been determined
AND
The module is not performing only one function

THEN

Ask if what relates the activities of the function
AND
Place answer in the activities related field.

Sequence Important

IF

The module cohesion type has not been determined
AND
The relation between activities is Data or Control

THEN

Ask if sequence is important
AND
Place answer in the sequence important field.

Same Category
IF

The module cohesion type has not been determined
AND
The activity relation is not Control or Data
THEN

Ask if the activities are in the same general category
AND
Place the answer in the same category field.

Note that as few as one question or as many as all four may be asked. In any case, the minimum number of questions that allow the KBSMA to infer the cohesion is asked.

Functional Cohesion
IF

The module is performing only one function
THEN

Set cohesion type to functional

Sequential Cohesion
IF

The module is not performing only one function
AND
The activities of the module are related by Data
AND
The sequence is important
THEN

Set cohesion type to sequential

Communicational Cohesion
IF

The module is not performing only one function
AND
The activities of the module are related by Data
AND
The sequence is not important
THEN

Set cohesion type to communicational

Procedural Cohesion

IF

The module is not performing only one function
AND
The activities of the module are related by Control
AND
The sequence is important

THEN

Set cohesion type to procedural

Temporal Cohesion

IF

The module is not performing only one function
AND
The activities of the module are related by Control
AND
The sequence is not important

THEN

Set cohesion type to temporal

Logical Cohesion

IF

The module is not performing only one function
AND
The activities of the module are not related
AND
The activities are in the same general category

THEN

Set cohesion type to logical

Coincidental Cohesion

IF

The module is not performing only one function
AND
The activities of the module are not related
AND
The activities are in the same general category

THEN

Set cohesion type to logical

The coupling rules show how stored data can be used by an inference mechanism without user interaction. The cohesion rules demonstrate how a knowledge based system can interact with the user to obtain information necessary to make an inference.

Conclusion

This chapter has described the design of the KBSEE and the KBSMA. Data Flow Diagrams and Structured Listings were used in the design documentation. General issues in knowledge based system design were also discussed. Chapter IV continues the description of this investigation by presenting the implementation phase.

IV. Implementation

Introduction

This chapter discusses the implementation of the Knowledge Based Software Engineering Environment, KBSEE, and the Knowledge Based Software Module Analysis tool, KBSMA. First, the system level implementation is presented, followed by a more detailed discussion of each subsystem. The final section of this chapter describes the integration of tools into the KBSEE in general and the KBSMA in particular. Source code listings are contained in Appendix C.

System Implementation

KBSEE is implemented on the AFIT Information Sciences Laboratory's DEC VAX-11/780 superminicomputer under the VAX/VMS operating system version 4.2. The ISL VAX was chosen as the target machine for several reasons. First, it was available and not overloaded compared to the other computer systems at AFIT. Second, there are a wide variety of languages and tools available on the system. Table 4.1 shows the languages and tools available on the ISL VAX. Finally, it is the host machine of the SDW (Hadfield, 1982:17-18), on which KBSEE is based.

After deciding upon the host computer system, the selec-

Compilers	Text Formaters	Data Management
Ada	Runoff	CDD
C	Massll	Datatrieve
Fortran		Ingres
Pascal		TOTAL
Assemblers	**Graphics**	**Screen Management**
VAX-11 Macro	GKS	FMS
	DecGraph	TDMS
	DecSlide	
	Plotln	
	SYSFL	
Editors	**Interpreters**	**Miscellaneous**
EDT	VAX LISP	Linker
TPU	NIL	Debugger
AI		Librarian
OPS-5		CMS
Art		PMS

Table 4.1 - Languages and Tools on the ISL VAX

tion of an programming language was the next step in the implementation phase. Available languages include C, FORTRAN, Pascal, Ada, LISP, Prolog, and the Digital Command Language (DCL). Of these, C was selected.

The most important reason for choosing the C programming language was the nature of the language. The C language is a general-purpose programming language which is manageable because of its small size, flexible because of its ample supply of operators, and powerful in its utilization of modern control flow and data structures. It is a simple language, but one rich in its variety of Run-Time Libraries of functions and macros (Kernighan, 1978) (Helms, 1984).

The second reason for choosing C is that it is highly portable, while providing access to the powerful VAX/VMS environment. It would not be very difficult to rehost a software system written in C to a new target computer system.

The VAX C programming language contains a Run-Time Library called Curses (DEC, 1985:26.1-26.39). The Curses library contains very powerful functions for controlling the display of terminal screens. Rather than developing operating system specific or terminal specific routines for screen management, greater program portability is obtained by using the Curses library, which is implemented in many C compilers on a wide variety of host computers.

Furthermore, C was chosen because it handles strings better than FORTRAN or Pascal, generates code which compiles faster and runs faster than Ada (a much larger language) (MacLennan, 1983), and because it generates executable code that can be moved and run on any VAX/VMS computer whether or not a the target system has a C compiler. This transportability cannot be accomplished by using a LISP or Prolog interpreter. LISP and Prolog programs require a LISP or Prolog interpreter for their execution. DCL is also interpreted (by the DEC Command Language Interpreter) and runs very slow compared to an executable program. DCL programs are actually series of executable programs, so the VAX/VMS operating system must load and execute each DCL command separately.

The particular implementation of C used is DEC's VAX C version 2.0. VAX C is a full and complete implementation of the C language, as defined in (Kernighan, 1978). It also provides access to the very rich VAX Run Time Library (in addition to the C Run Time Library discussed in previous paragraphs). The VAX-11 Run Time Library includes routines to screen management, keyboard management, data conversions, system management, and mathematics. The one keyboard management routine and one system management routine were used in the implementation of KBSEE. This is discussed in detail below.

Display Manager Implementation

The display manager uses functions and macros from the VAX-11 C Curses Run-Time Library to initialize, display, and maintain the user's terminal screen. For a detailed description of Curses and its functions and macros, see (DEC, 1985). The Curses functions are used to divide the terminal screen into windows, perform input and output in the windows, display and remove windows, and update the entire display.

Command Interpreter Implementation

As described in Chapters II and III, the command interpreter obtains valid menu selections from the display manager. This is implemented through the Curses function winch, which inputs a character from a given location on the window. The command interpreter processes user command via keyboard input. The VAX-11 Run-Time Library procedures SMG$CREATE_VIRTUAL_KEYBOARD and SMG$READ_STRING are used for reading keyboard function keys and controlling keyboard input. These routines can read function keys on any terminal defined in the VAX/VMS terminal definition table. Currently, DEC VT-52, DEC VT-100, and DEC VT-200 series terminals are defined. The KBSEE uses the numeric keypad of the VT100 and the edit keypad of the VT-200 for single keystroke commands, as shown in Figure 4.1.

VT 100 Series

VT 200 Series
Help = HELP (F15)

Figure 4.1 - KBSEE Terminal Keyboard Layout

Commands are validated, then appropriate execution routines are invoked for the individual menu selections. If the command is for an external tool or program, the VAX-11 Run-Time Library procedure LIB$SPAWN is used to spawn a subprocess for that tool or program. Details of actual spawn procedures used for the various external routines are shown in Appendix E.

Project Manager Implementation

The project manager is a collection of data files with simple access functions for reading and writing to these data files. The project database is implemented as a standard sequential file, which is searched on two fields, project name and work file name. This implementation uses standard C Run-Time Library routines. Use of a fully indexed file would speed search in a large file, but would cause loss of transportability since indexed file implementations are not standardized in the C language definition. The project database should contain less than 100 records, which should not require a noticable delay in searching. There is a project database for each KBSEE user, located in the default login directory with file name KBSEE.PROJECTS.

The user profile information store is implemented as a

standard sequential file, so the access functions use standard C I/O routines. Since the file contains only three records, an indexed file would not be any faster since all records in the file are processed whenever the file is accessed. There is a user profile store for each workfile in a project, located in the same directory as the workfile. The name of the user profile store is based on the name of the workfile. For example, if the name of the workfile is FOO.BAR, then the name of the profile store would be FOO.PROFILE.

The project save store is also implemented as a standard sequential file. The file contains only one record, with two data items, so other file formats would not speed access and would result in loss of transportability. The save store is located in each user's default login directory with name KBSEE.SAVE.

Tool Set Implementation

The tool set consists of VAX/VMS compilers, linkers, and editors, the librarian, and symbolic debugger. Other tools may be implemented in a manner similar to the tools supplied by DEC. Tools are usually executable programs invoked by either a run command or a Command Language Interpreter (CLI) command sequence. In either case, the executive spawns a

subprocess in order to execute the tool. Since many tools write to the screen, care is taken to avoid interfering with the KBSEE display manager windows. This is accomplished by using DCL command procedure file for the spawned subprocess. The first line in the command procedure is a clear screen command to erase the KBSEE menus. After the tool is invoked and writes to the terminal screen, another clear screen command is issued. Once the spawned process returns control to the KBSEE, the display manager function UPDATE_DISPLAY is called to refresh the display of the KBSEE windows and menus on the terminal screen. Tools which receive input from the keyboard must have the VAX/VMS logical name SYS$INPUT defined to be the terminal keyboard. By default, SYS$INPUT is defined to be the command procedure file for executable programs invoked from a command procedure. So, prior to the actual tool invocation in the command procedure, the line DEFINE/USER_MODE SYS$INPUT SYS$COMMAND must appear. The /USER_MODE command qualifier makes the definition valid only for the next program being executed. If this qualifier did not appear on the DEFINE command, input would continue to come from the keyboard, even when it should not. For tools which are expected to generate error messages, the VAX/VMS logical name SYS$ERROR should be defined to be a file. This will allow the user to scroll though files whenever con-

venient, rather than loose the error messages when the screen is cleared after the tool is finished executing.

KBSMA Implementation

The KBSMA is implemented in a rule based production system called OPS-5. Most commercial tools are rule based systems (i.e., AL/X, ES/P Advisor, INSIGHT, M.1, Personal Consultant, SeRIES-PC, EXPERT, KES, OPS-5, and S.1) or hybrid systems containing rules within frames or a semantic network (ART, KEE, and LOOPS) (Harmon, 1985:129-134). Since all these commercial systems contain rules in some sort, the implementation decision was dependant on the tools available on the ISL VAX. At the time of KBSMA design and implementation, OPS-5 was the only commercial knowledge based system development tool available on the ISL VAX. Also, the prototype system developed for the Advanced Topics in Artificial Intelligence course, EENG 749, was implemented in OPS-5 on the AFIT/SI SSC VAX/UNIX system.

OPS is an acronym for Official Production System. The particular implementation of OPS-5 used is DEC VAX-11 OPS-5 version 1.0 (DEC, 1984). It generates executable images which can be run in a manner identical to the code generated by other compilers. This differs from many other implementations of OPS-5 which require the user to first

invoke a LISP interpreter, then load OPS-5 (as is the case with the implementation on the AFIT/SI SSC VAX using Franz LISP). The code generated by the VAX-11 OPS-5 compiler is optimized for faster execution.

Rules are easily transformed from their English descriptions into OPS-5 syntax. The KBSMA user's manual in Appendix F explains the process of transforming rules into OPS-5 productions.

The KBSMA module and variable data structures are implemented as OPS-5 LITERALIZE structures. LITERALIZE is an OPS-5 keyword that specifies that the next item in the list is the name of a list data structure with the remaining items in the list used as field names within the list; i.e., (literalize FOO BAR1 BAR2) would create a data structure named FOO with items BAR1 and BAR2. The implementation of the data structures are shown in Appendix D.

The module data structure is nearly a one-for-one translation of the AFIT/ENG guideline standard (AFIT/ENG, 1984). The exceptions are the parameters passed and globals used fields. To allow up to three of passed parameters and global variables, there are three repeated fields for each: passed-parameter-1, passed-parameter-2, passed-parameter-3, global-1, global-2, and global-3. This is necessary because although OPS-5 can represent more than one item in a field

as a list, it cannot separate list items for separate handling. This meant a tradeoff was necessary between a general purpose data representation of a field with multiple items with a very complicated access scheme requiring invokation of external system routines or a simple repetition within the data structure, allowing less flexibility, but a much simpler access scheme. Simplicity was considered more important since the scheme could easily be extended and is transportable. The user's manual contained in Appendix F discusses implementation-specific details.

The knowledge base portion of the KBSMA is located in the OPS-5 source file KBSMA.OPS which is compiled into the executable program KBSMA.EXE. The knowledge base contains the data structure definitions, production rules, and a control procedure - startup. The startup procedure defines an environment for OPS-5 execution which includes the type of output, halt mechanism, and search/inference strategy. The startup procedure also loads the situation data into working memory. Situation data is contained in a file name KBSMA_INSTANCES.DAT. It contains data values for each of the modules to be analyzed in the variable and module data structures. The separate instance data allows the knowledge base to be used to analyze any number of different modules.

The command procedure to invoke the KBSMA has the following lines:

```
$CLS

$DEFINE/USER SYS$INPUT SYS$COMMAND

$RUN KBSMA

$CLS

$EXIT
```

The name of the command procedure is KBSEE_ANALYZE.COM, which is located in the KBSEE system directory, KBSEE$SYS-TEM.

Conclusion

This chapter discussed the implementation of the Knowledge Based Software Engineering Environment, KBSEE, and the Knowledge Based Software Module Analyzer, KBSMA. The choice of host computer system was discussed, as was the choice of implementation language. Detailed implementation decisions were presented for each KBSEE major function and data structure. Implementation and integration of new tools were discussed, with the KBSMA presented as a detailed example of tool implementation. The next chapter, Chapter V, concludes this thesis effort and provides recommendations for further investigation.

V. Conclusion and Recommendations

Introduction

This thesis investigation has described the development of a Knowledge Based Software Engineering Environment, KBSEE, and a Knowledge Based Software Module Analysis tool, KBSMA. KBSEE is consists of several interacting components, including a set of software development tools such as compilers, editors, linkers, and debuggers. The KBSMA analyzes software modules for the software engineering parameters of coupling and cohesion. The primary goal of this investigation is an easily useable environment for developing software.

This final chapter presents a short summary of the system development, followed by an analysis relating the developed system to the standards described in Chapter 1. Finally, recommendations for future investigations are presented.

Development Summary

KBSEE and KBSMA were built using a variation of the classic software development life cycle along with exploratory programming/rapid prototyping. First, an extensive literature search was conducted to gain a better

understanding of knowledge based systems, software development environments, the software development process and its problems, and how these problems might be diminished through automation. The information gleaned from this search, along with prior knowledge from experience and observation, was used to perform the requirements analysis phase. In this phase, sets of requirements were defined for the environment as a whole, for each subsystem, and for the analysis tool.

After generating the initial sets of requirements, prototypical systems were developed and implemented. A cyclic process of design and implementation was performed until a satisfactory system was completed. Feedback from the prototypes provided useful information about the completeness and consistency of the requirements, which were modified, if necessary. Throughout implementation, routines were tested as they were developed. This informal testing was conducted on the isolated module to show that it performed as intended and on the module as it was integrated into the environment to ensure that its system interface behaved in the intended manner. Although no formal testing was performed, the informal tests do suggest that the KBSEE and KBSMA are reasonably error-free.

Analysis of the Current System

The KBSEE is built upon SDW concepts and is designed to free the user from having to remember file names and locations and re-enter commonly used commands. It has a more efficient human-computer interface than does the SDW and performs its functions considerably faster. Its tool set is sufficient for building software systems in a variety of languages, including C, FORTRAN, Pascal, and Assembler.

The software analysis tool determines a module's coupling and cohesion. It can obtain information through interaction with the user and through access to the situation file.

The KBSEE is not completely implemented and needs to be extended. It needs a centralized data dictionary and appropriate access functions to avoid having to interact with the user for module information and to maintain consistency checks throughout software development. Not all menu selections are implemented, although what is currently implemented does show the utility of the KBSEE and the KBSMA, so both portions of this investigation should be considered successful. The next section discusses recommendations for future investigation.

Recommendations for Future Investigation

Not all the requirements have been fulfilled by the design and not all the design has been implemented. The lack of a centralized data dictionary and access functions is the primary reason for the incompleteness. Also, formal testing needs to be accomplished. Complete testing is not especially difficult, but is extremely time-consuming and even then, does not guarantee the absence of errors.

As discussed earlier, future work needs to be done in the area of a data dictionary and access functions for the KBSEE. Hopefully, the system would be based on the AFIT/ENG guidelines to ensure that the tool would be useful to the KBSEE and KBSMA. A new data dictionary system could also be the basis for new tools for consistency checking and automated graphic output of structure charts, DFDs, SADTs, etc.

The analysis of software modules for parameters other than coupling and cohesion would be an excellent topic for future investigation; i.e., determining the time/space complexity of a module and then using module complexity to determine an entire program's overall complexity is a difficult, but interesting problem. Determining the time complexity would require analyzing the loops and branches of the module's algorithms. For example, if the current level

is a loop and the current level is within the scope of a previous loop, then the time complexity of the current level is of order n-squared. To obtain the type of information required for this analysis would require parsing of the programming language syntax. Even then, there are subtleties that would require human analysis. Such an investigation would probably require techniques from computational analysis, artificial intelligence, and compiler theory.

The list of useful software development tools that could be developed for the KBSEE, or any other environment, is virtually endless. A future investigation of what tools should be in an environment would prove invaluable and would, in turn, stimulate more work on software tools.

The incorporation of Artificial Intelligence technology into software engineering environments could be very useful, as is demonstrated by the KBSMA developed in this effort. Future work needs to be done to clarify the role of AI in software engineering problems.

Conclusion

This thesis effort has shown the utility of combining Software Engineering and Artificial Intelligence technologies. The benefits of using off-the-shelf tools in

a reasonably efficient environment were also demonstrated. All in all, this investigation proved to be a success.

Data Dictionary

for the

Knowledge Based

Software Engineering Environment

(KBSEE)

```
***********************************************************
   NAME: copyright_win
   TYPE: WINDOW (defined by Curses package)
   SCOPE: Global

   USE: stores window data for copyright window

   REFERENCED BY MODULES: bld_copyright

***********************************************************


***********************************************************
   NAME: curr_dat
   TYPE: record of type menu_data
   SCOPE: Global

   USE: stores menu position information for the current menu

   REFERENCED BY MODULES: chk_left_right, chk_move, get_item,
main, chk_main_1, chk_main_2, chk_main_3, chk_main_4

***********************************************************


***********************************************************
   NAME: curr_proj
   TYPE: record
   SCOPE: Global

      RECORD ITEM NAME: project_name
      RECORD ITEM TYPE: array of 20 characters
      RECORD ITEM USE: stores name of current project

      RECORD ITEM NAME: location
      RECORD ITEM TYPE: array of 64 characters
      RECORD ITEM USE: stores location of current project

      RECORD ITEM NAME: work_file
      RECORD ITEM TYPE: array of 32 characters
      RECORD ITEM USE: stores name of current work file

   REFERENCED BY MODULES: chk_save_file, exec_start_project,
get_project, save_project

***********************************************************
```

```
************************************************************
   NAME: curr_win
   TYPE: WINDOW (defined by Curses package)
   SCOPE: Global

   USE: stores window data for current window

   REFERENCED BY MODULES: chk_left_right, chk_main_sel,
chk_move, get_item, main

************************************************************

************************************************************
   NAME: def_head
   TYPE: WINDOW (defined by Curses package)
   SCOPE: Global

   USE: stores window data for default window header

   REFERENCED BY MODULES: bld_def, display_menus,
update_display

************************************************************

************************************************************
   NAME: def_win
   TYPE: WINDOW (defined by Curses package)
   SCOPE: Global

   USE: stores window data for default window

   REFERENCED BY MODULES: bld_def, display_menus,
update_display

************************************************************

************************************************************
   NAME: exit_flag
   TYPE: integer
   SCOPE: Global, used Local as alias exit_now

   USE: flag set to exit system

   REFERENCED BY MODULES: main

************************************************************
```

Data Dictionary - KBSEE

```
************************************************************
   NAME: found
   TYPE: integer
   SCOPE: Local

   USE: flag set to when a comparison is true

   REFERENCED BY MODULES: get_project

************************************************************

************************************************************
   NAME: help_win
   TYPE: WINDOW (defined by Curses package)
   SCOPE: Global

   USE: stores window data for help window

   REFERENCED BY MODULES: bld_help

************************************************************

************************************************************
   NAME: i
   TYPE: integer
   SCOPE: Local

   USE: loop control variable

   REFERENCED BY MODULES: chk_save_file, exec_start_project,
get_project, save_project

************************************************************

************************************************************
   NAME: inp
   TYPE: single character
   SCOPE: Local

   USE: temporary storate location for string to be converted
into a descriptor type for use in RTL routines

   REFERENCED BY MODULES: main

************************************************************
```

Data Dictionary - KBSEE

```
************************************************************
     NAME: input
     TYPE: descriptor
     SCOPE: Local

     USE: descriptor for a string for use in RTL routines

     REFERENCED BY MODULES: main

************************************************************

************************************************************
     NAME: kid
     TYPE: unsigned integer
     SCOPE: Global

     USE: keyboard identifier used by Screen Mgt RTL routines

     REFERENCED BY MODULES: main

************************************************************

************************************************************
     NAME: m_len
     TYPE: integer
     SCOPE: Local

     USE: length of modifiers for RTL routine

     REFERENCED BY MODULES: main

************************************************************

************************************************************
     NAME: modifiers
     TYPE: integer
     SCOPE: Local

     USE: I/O RTL routine modifiers

     REFERENCED BY MODULES: main

************************************************************
```

Data Dictionary - KBSEE

```
*************************************************************
   NAME: main_data_1
   TYPE: record of type menu_data
   SCOPE: Global

   USE: stores menu position information for main_menu_1

   REFERENCED BY MODILES: chk_left_right, main

*************************************************************

*************************************************************
   NAME: main_data_2
   TYPE: record of type menu_data
   SCOPE: Global

   USE: stores menu position information for main_menu_2

   REFERENCED BY MODILES: chk_left_right, main

*************************************************************

*************************************************************
   NAME: main_data_3
   TYPE: record of type menu_data
   SCOPE: Global

   USE: stores menu position information for main_menu_3

   REFERENCED BY MODILES: chk_left_right, main

*************************************************************

*************************************************************
   NAME: main_data_4
   TYPE: record of type menu_data
   SCOPE: Global

   USE: stores menu position information for main_menu_4

   REFERENCED BY MODILES: chk_left_right, main

*************************************************************
```

```
*********** ,*********************************************
   NAME: main_menu_1
   TYPE: WINDOW (defined by Curses package)
   SCOPE: Global

   USE: stores window data for main_menu_1

   REFERENCED BY MODULES: chk_left_right, chk_main_sel,
bld_menu_1, display_menus, update_display, main

   ***********************************************************


   ***********************************************************
   NAME: main_menu_2
   TYPE: WINDOW (defined by Curses package)
   SCOPE: Global

   USE: stores window data for main_menu_2

   REFERENCED BY MODULES: chk_left_right, chk_main_sel,
bld_menu_2, display_menus, update_display

   ***********************************************************


   ***********************************************************
   NAME: main_menu_3
   TYPE: WINDOW (defined by Curses package)
   SCOPE: Global

   USE: stores window data for main_menu_3

   REFERENCED BY MODULES: chk_left_right, chk_main_sel,
bld_menu_3, display_menus, update_display

   ***********************************************************


   ***********************************************************
   NAME: main_menu_4
   TYPE: WINDOW (defined by Curses package)
   SCOPE: Global

   USE: stores window data for main_menu_4

   REFERENCED BY MODULES: chk_left_right, chk_main_sel,
bld_menu_4, display_menus, update_display

   ***********************************************************
```

```
************************************************************
   NAME: menu_data
   TYPE: record type definition
   SCOPE: Global

     RECORD ITEM NAME: min_y
     RECORD ITEM TYPE: integer
     RECORD ITEM USE: minimum vertical range of menu items

     RECORD ITEM NAME: max_y
     RECORD ITEM TYPE: integer
     RECORD ITEM USE: maximum vertical range of menu items

     RECORD ITEM NAME: curr_y
     RECORD ITEM TYPE: integer
     RECORD ITEM USE: cursor position within vertical range

     RECORD ITEM NAME: curr_x
     RECORD ITEM TYPE: integer
     RECORD ITEM USE: cursor position within horizontal range

     RECORD ITEM NAME: menu_item
     RECORD ITEM TYPE: array of 17 characters
     RECORD ITEM USE: name of current menu item

   REFERENCED BY DATA STRUCTURES: curr_dat, main_data_1,
main_data_2, main_data_3, main_data_4

************************************************************

************************************************************
   NAME: menu_hdr_1
   TYPE: WINDOW (defined by Curses package)
   SCOPE: Global

   USE: stores window data for main menu header 1

   REFERENCED BY MODULES: bld_menu_hdr, display_menus,
update_display
************************************************************
```

```
****************************************************************
    NAME: menu_hdr_2
    TYPE: WINDOW (defined by Curses package)
    SCOPE: Global

    USE: stores window data for main menu header 2

    REFERENCED BY MODULES: bld_menu_hdr, display_menus,
update_display

****************************************************************


****************************************************************
    NAME: mesg_head
    TYPE: WINDOW (defined by Curses package)
    SCOPE: Global

    USE: stores window data for message window header

    REFERENCED BY MODULES: bld_mesg, display_menus,
update_display

****************************************************************


****************************************************************
    NAME: mesg_win
    TYPE: WINDOW (defined by Curses package)
    SCOPE: Global

    USE: stores window data for message window

    REFERENCED BY MODULES: bld_mesg, display_menus,
update_display, chk_main_sel, chk_save_file,
exec_start_project, get_project, chk_main_1, chk_main_2,
chk_main_3

****************************************************************


****************************************************************
    NAME: position
    TYPE: integer
    SCOPE: Local

    USE: current cursor position inside menu string

    REFERENCED BY MODULES: get_item
****************************************************************
```

Data Dictionary - KBSEE

```
*****************************************************************
   NAME: proj
   TYPE: array of 20 characters
   SCOPE: Local

   USE: name of project read from project database

   REFERENCED BY MODULES: get_project

*****************************************************************

*****************************************************************
   NAME: ret_val
   TYPE: integer (used as boolean)
   SCOPE: Local

   USE: return value set true if save file exits

   REFERENCED BY MODULES: chk_save_file

*****************************************************************

*****************************************************************
   NAME: status
   TYPE: integer
   SCOPE: Local

   USE: return status of RTL routines

   REFERENCED BY MODULES: main

*****************************************************************

*****************************************************************
   NAME: sts_head
   TYPE: WINDOW (defined by Curses package)
   SCOPE: Global

   USE: stores window data for status window header

   REFERENCED BY MODULES: bld_sts, display_menus,
update_display
*****************************************************************
```

Data Dictionary - KBSEE

```
**************************************************************
    NAME: sts_win
    TYPE: WINDOW (defined by Curses package)
    SCOPE: Global

    USE: stores window data for status window

    REFERENCED BY MODULES: bld_sts, display_menus,
update_display

**************************************************************

**************************************************************
    NAME: terminator
    TYPE: short integer
    SCOPE: Local

    USE: keyboard function key terminator for SMG RTL routine

    REFERENCED BY MODULES: chk_left-right, chk_move,
chk_main_sel, main

**************************************************************

**************************************************************
    NAME: tmp_x
    TYPE: integer
    SCOPE: Local

    USE: temporary store for cursor horizontal position

    REFERENCED BY MODULES: get_item

**************************************************************
```

Structure Listing

for the

Knowledge Based

Software Engineering Environment

(KBSEE)

Level	Module	Number	Couples
0	main	0.0	
1	init	1.0	
2	bld_help	1.0.0	
2	bld_copyright	1.0.1	
2	bld_main	1.0.2	
3	bld_menu_hdr	1.0.2.0	
3	bld_menu_1	1.0.2.1	
3	bld_menu_2	1.0.2.2	
3	bld_menu_3	1.0.2.3	
3	bld_menu_4	1.0.2.4	
2	bld_mesg	1.0.3	
2	bld_sts	1.0.4	
2	bld_def	1.0.5	
1	display_menus	1.1	
1	chk_save_file	1.2	
1	get_project	1.3	
1	get_profile	1.4	
1	get_item	1.5	curr_win, curr_dat
1	chk_move	1.6	terminator
1	chk_left_right	1.7	terminator
1	chk_main_sel	1.8	exit_now, termi..ator
2	chk_main_1	1.8.0	
3	exec_bld_program	1.8.0.0	

Structure Listing - KBSEE

Level	Module	Number Couples
3	exec_edit	1.8.0.1
3	exec_compile	1.8.0.2
3	exec_link	1.8.0.3
3	exec_run	1.8.0.4
3	exec_display_error	1.8.0.5
3	exec_start_project	1.8.0.6
3	exec_select_project	1.8.0.7
3	exec_list_projects	1.8.0.8
3	exec_display_project	1.8.0.9
2	chk_main_2	1.8.1
3	exec_work_file	1.8.1.0
3	exec_user_profile	1.8.1.1
3	exec_introduction	1.8.1.2
2	chk_main_3	1.8.2
3	exec_debugger	1.8.2.0
3	exec_analyer	1.8.2.1
3	exec_librarian	1.8.2.2
3	exec_printer	1.8.2.3
3	exec_text_format	1.8.2.4
2	chk_main_4	1.8.3
3	exec_spawn	1.8.3.0
3	exec_exit	1.8.3.1
1	save_project	1.9

Level Module	Number Couples
0 update_display	0.1

Source Code Listing

for the

Knowledge Based

Software Engineering Environment

(KBSEE)

```
/***********************************************************************
*                                                                     *
*     DATE: 2/20/86                                                   *
*     VERSION: 1.0                                                    *
*                                                                     *
*         TITLE: KBSEE Main Executive Routines                        *
*         FILENAME: kbsee.c                                           *
*         COORDINATOR: Capt Dave Fautheree                            *
*         PROJECT: KBSEE (M.S. Thesis)                                *
*         OPERATING SYSTEM: VAX/VMS version 4.2                       *
*         LANGUAGE: VAX-11 C                                          *
*         USE: CC KBSEE                                               *
*         CONTENTS:                                                   *
*                   chk_main_1 - process selections for main menu 1   *
*                   chk_main_2 - process selections for main menu 2   *
*                   chk_main_3 - process selections for main menu 3   *
*                   exec_analyzer - execute KBSMA analysis tool       *
*                   chk_main_4 - process selections for main menu 4   *
*                   main - main executive routine                     *
*                                                                     *
*         FUNCTION: Implements main executive functions               *
*                   and command interpretation.                       *
*                                                                     *
***********************************************************************/
```
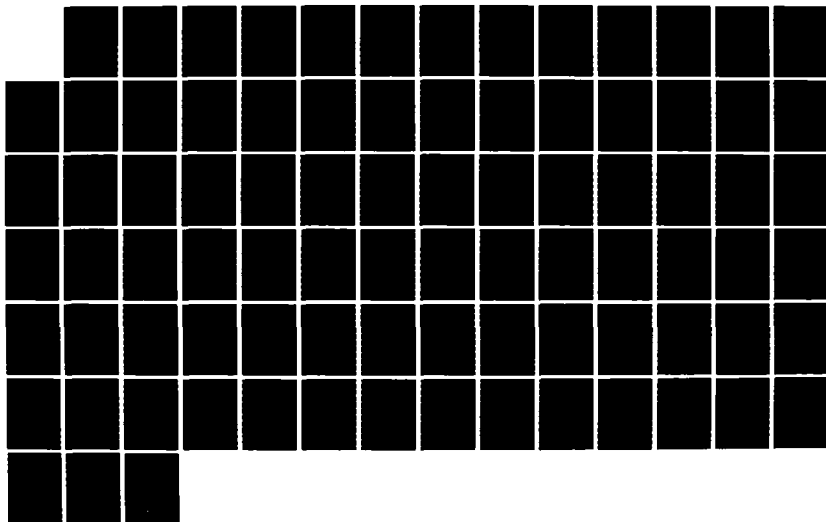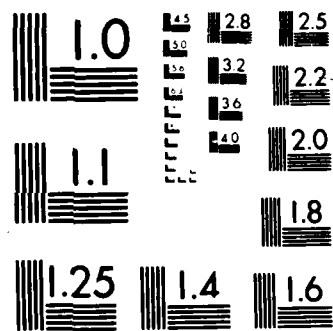
```
#include    curses              /* Curses Screen Management Definitions */
#include    descrip             /* Descriptor Definitions */
#include    iodef               /* I/O Status Definitions */

#define         ITEM_LENGTH      16
#define         AND              &&
#define         OR               ||
#define         EQ               ==
#define         NEQ              !=

#define         SIZE_NAME        20
#define         SIZE_LOC         64
#define         SIZE_WORK_FILE   32

struct curr_proj
        {
            char     project_name[SIZE_NAME];
            char     location[SIZE_LOC];
            char     work_file[SIZE_WORK_FILE];
        };

struct menu_data
        {
            int      min_y;
            int      max_y;
            int      curr_y;
            int      curr_x;
            char     menu_item[ITEM_LENGTH + 1];
        };

struct menu_data    main_data_1 = { 0, 9, 0, 0, ""};
struct menu_data    main_data_2 = { 0, 2, 0, 0, ""};
struct menu_data    main_data_3 = { 0, 4, 0, 0, ""};
struct menu_data    main_data_4 = { 0, 1, 0, 0, ""};

WINDOW *curr_win;
struct menu_data    curr_dat;

WINDOW *menu_hdr_1,
       *menu_hdr_2,
       *main_menu_1,
       *main_menu_2,
       *main_menu_3,
       *main_menu_4,
       *mesg_head,
       *mesg_win,
```

C - 3

```
        *sts_head,
        *sts_win,
        *def_win,
        *def_head,
        *help_win;
        *copyright_win;

int        SMG$CREATE_VIRTUAL_KEYBOARD();
int        SMG$READ_STRING();
int        LIB$SPAWN();

char    *project_file;
int        exit_flag = FALSE;
unsigned    kid;

#include    "KBSEE_EXEC.C"            /* Include Executive Modules */
#include    "KBSEE_PROJ.C"           /* Include Project Mgt Modules */
```

```
/******************************************************************
*                                                                *
*      DATE: 2/20/86                                             *
*      VERSION: 1.0                                              *
*                                                                *
*      NAME: chk_main_1                                          *
*      MODULE NUMBER: 1.8.0                                      *
*         DESCRIPTION: checks menu selections for main menu 1    *
*         PASSED VARIABLES:                                      *
*         RETURNS:                                               *
*         GLOBAL VARIABLES USED: curr_dat, mesg_win              *
*         GLOBAL VARIABLES CHANGED:                              *
*         FILES READ:                                           *
*         FILES WRITTEN:                                        *
*         HARDWARE INPUT:                                       *
*         HARDWARE OUTPUT:                                      *
*      MODULES CALLED: exec_bld_program, exec_edit, exec_compile, exec_link, *
*                      exec_run, exec_display_error, exec_start_project,     *
*                      exec_select_project, exec_list_projects,             *
*                      exec_display_project                                 *
*                                                                *
*      CALLING MODULES: chk_main_sel                            *
*                                                                *
*         AUTHOR: Capt Dave Fautheree                           *
*         HISTORY:                                              *
*                                                                *
*                                                                *
******************************************************************/

chk_main_1()
{
    if (strcmp(curr_dat.menu_item, "Build Program    ") EQ 0)
        wprintw(mesg_win, "\nBuilding Program");
    else
    if (strcmp(curr_dat.menu_item, "Edit             ") EQ 0)
        wprintw(mesg_win, "\nEditing Program");
    else
    if (strcmp(curr_dat.menu_item, "Compile          ") EQ 0)
        wprintw(mesg_win, "\nCompiling Program");
    else
    if (strcmp(curr_dat.menu_item, "Link             ") EQ 0)
        wprintw(mesg_win, "\nLinking Program");
    else
    if (strcmp(curr_dat.menu_item, "Run              ") EQ 0)
        wprintw(mesg_win, "\nRunning Program");
    else
```

```
if (strcmp(curr_dat.menu_item, "Display Errors   ") EQ 0)
    wprintw(mesg_win, "\nDisplaying Errors");
else
if (strcmp(curr_dat.menu_item, "Start Project    ") EQ 0)
    exec_start_project();
else
if (strcmp(curr_dat.menu_item, "Select Project   ") EQ 0)
    wprintw(mesg_win, "\nSelect Project");
else
if (strcmp(curr_dat.menu_item, "List Projects    ") EQ 0)
    wprintw(mesg_win, "\nList Projects    ");
else
if (strcmp(curr_dat.menu_item, "Display Project ") EQ 0)
    wprintw(mesg_win, "\nDisplay Project ");
}
```

```
/**************************************************************************
*                                                                        *
*     DATE: 2/20/86                                                       *
*     VERSION: 1.0                                                        *
*                                                                        *
*     NAME: chk_main_2                                                    *
*     MODULE NUMBER: 1.8.1                                                *
*        DESCRIPTION: checks menu selections for main menu 2             *
*        PASSED VARIABLES:                                                *
*        RETURNS:                                                         *
*        GLOBAL VARIABLES USED: curr_dat, mesg_win                        *
*        GLOBAL VARIABLES CHANGED:                                        *
*        FILES READ:                                                      *
*        FILES WRITTEN:                                                   *
*        HARDWARE INPUT:                                                  *
*        HARDWARE OUTPUT:                                                 *
*     MODULES CALLED: exec_work_file, exec_user_profile, exec_introduction *
*                                                                        *
*     CALLING MODULES: chk_main_sel                                       *
*                                                                        *
*        AUTHOR: Capt Dave Fautheree                                      *
*        HISTORY:                                                         *
*                                                                        *
*                                                                        *
**************************************************************************/

chk_main_2()
{
    if (strcmp(curr_dat.menu_item, "Work File       ") EQ 0)
        wprintw(mesg_win, "\nWork File      ");
    else
    if (strcmp(curr_dat.menu_item, "User Profile    ") EQ 0)
        wprintw(mesg_win, "\nUser Profile   ");
    else
    if (strcmp(curr_dat.menu_item, "Introduction    ") EQ 0)
        wprintw(mesg_win, "\nIntroduction   ");
}
```

```
/*****************************************************************************
 *                                                                           *
 *    DATE: 2/20/86                                                          *
 *    VERSION: 1.0                                                           *
 *                                                                           *
 *    NAME: chk_main_3                                                       *
 *    MODULE NUMBER: 1.8.2                                                   *
 *       DESCRIPTION: checks menu selections for main menu 3                *
 *       PASSED VARIABLES:                                                   *
 *       RETURNS:                                                            *
 *       GLOBAL VARIABLES USED: curr_dat, mesg_win                          *
 *       GLOBAL VARIABLES CHANGED:                                          *
 *       FILES READ:                                                         *
 *       FILES WRITTEN:                                                      *
 *       HARDWARE INPUT:                                                     *
 *       HARDWARE OUTPUT:                                                    *
 *    MODULES CALLED: exec_debugger, exec_analyzer, exec_librarian,         *
 *                    exec_printer, exec_text_format                        *
 *                                                                           *
 *    CALLING MODULES: chk_main_sel                                         *
 *                                                                           *
 *       AUTHOR: Capt Dave Fautheree                                        *
 *       HISTORY:                                                            *
 *                                                                           *
 *                                                                           *
 *****************************************************************************/

chk_main_3()
{
   if (strcmp(curr_dat.menu_item, "Debugger        ") EQ 0)
      wprintw(mesg_win, "\nDebugger       ");
   else
   if (strcmp(curr_dat.menu_item, "Analyzer        ") EQ 0)
      exec_analyzer();
   else
   if (strcmp(curr_dat.menu_item, "Librarian       ") EQ 0)
      wprintw(mesg_win, "\nLibrarian      ");
   else
   if (strcmp(curr_dat.menu_item, "Printer         ") EQ 0)
      wprintw(mesg_win, "\nPrinter        ");
   else
   if (strcmp(curr_dat.menu_item, "Text Formatting ") EQ 0)
      wprintw(mesg_win, "\nText Formatting");
```

```
/***************************************************************************
 *                                                                         *
 *    DATE: 2/20/86                                                        *
 *    VERSION: 1.0                                                         *
 *                                                                         *
 *    NAME: exec_analyzer                                                  *
 *    MODULE NUMBER: 1.8.2.1                                               *
 *       DESCRIPTION: executes KBSMA analysis tool                         *
 *       PASSED VARIABLES:                                                 *
 *       RETURNS:                                                          *
 *       GLOBAL VARIABLES USED:                                            *
 *       GLOBAL VARIABLES CHANGED:                                         *
 *       FILES READ:                                                       *
 *       FILES WRITTEN:                                                    *
 *       HARDWARE INPUT:                                                   *
 *       HARDWARE OUTPUT:                                                  *
 *    MODULES CALLED: update_display                                       *
 *                                                                         *
 *    CALLING MODULES: chk_main_3                                          *
 *                                                                         *
 *       AUTHOR: Capt Dave Fautheree                                       *
 *       HISTORY:                                                          *
 *                                                                         *
 *                                                                         *
 ***************************************************************************/

exec_analyzer()
{
   char    inp;
           $DESCRIPTOR(input, "@kbsee$system:kbsee_analyze.com");

   lib$spawn(&input,0,0,0,0,0,0,0,0,0,0,0);

   update_display();
}
```

```
/********************************************************************
 *                                                                  *
 *      DATE: 2/20/86                                               *
 *      VERSION: 1.0                                                *
 *                                                                  *
 *      NAME: chk_main_4                                            *
 *      MODULE NUMBER: 1.8.3                                        *
 *         DESCRIPTION: checks menu selections for main menu 4      *
 *         PASSED VARIABLES:                                        *
 *         RETURNS:                                                 *
 *         GLOBAL VARIABLES USED: curr_dat, mesg_win               *
 *         GLOBAL VARIABLES CHANGED:                                *
 *         FILES READ:                                             *
 *         FILES WRITTEN:                                          *
 *         HARDWARE INPUT:                                         *
 *         HARDWARE OUTPUT:                                        *
 *      MODULES CALLED: exec_spawn, exec_exit                       *
 *                                                                  *
 *      CALLING MODULES: chk_main_sel                               *
 *                                                                  *
 *         AUTHOR: Capt Dave Fautheree                             *
 *         HISTORY:                                                *
 *                                                                  *
 *                                                                  *
 ********************************************************************/

chk_main_4()
{
   if (strcmp(curr_dat.menu_item, "Spawn to CLI     ") EQ 0)
       wprintw(mesg_win, "\nSpawn to CLI    ");
   else
   if (strcmp(curr_dat.menu_item, "Exit             ") EQ 0)
       wprintw(mesg_win, "\nExit            ");
}
```

```
/***************************************************************************
 *                                                                         *
 *      DATE: 2/20/86                                                      *
 *      VERSION: 1.0                                                       *
 *                                                                         *
 *      NAME: main                                                        *
 *      MODULE NUMBER: 0.0                                                 *
 *         DESCRIPTION: main executive routine                            *
 *         PASSED VARIABLES:                                              *
 *         RETURNS:                                                       *
 *         GLOBAL VARIABLES USED: curr_dat, curr_win, kid, main_data_1,   *
 *                                main_menu_1                              *
 *         GLOBAL VARIABLES CHANGED: curr_win, curr_dat                   *
 *         FILES READ:                                                    *
 *         FILES WRITTEN:                                                 *
 *         HARDWARE INPUT:                                                *
 *         HARDWARE OUTPUT:                                               *
 *      MODULES CALLED: init, display_menus, chk_save_file, get_project,  *
 *                      get_profile, get_item, chk_move, chk_left_right,   *
 *                      chk_main_sel                                       *
 *                                                                         *
 *      CALLING MODULES:                                                  *
 *                                                                         *
 *         AUTHOR: Capt Dave Fautheree                                    *
 *         HISTORY:                                                       *
 *                                                                         *
 *                                                                         *
 ***************************************************************************/

main()
{
    int      status;
    char     inp;
             $DESCRIPTOR(input, inp);
    int          m_len = 1;
    unsigned     modifiers;
    short int    terminator;

    status =    SMG$CREATE_VIRTUAL_KEYBOARD(&kid);
    modifiers = (IO$M_ESCAPE + IO$M_NOECHO + IO$M_PURGE);

    init();

    display_menus();

    curr_win = main_menu_1;
```

```
    curr_dat = main_data_1;

    if (chk_save_file())
    {
        get_project();
/*      get_profile();
*/
    }

    while(!exit_flag)
    {
        wrefresh(mesg_win);
        wrefresh(sts_win);
        do
        {
            get_item(curr_win, &curr_dat);
            status = SMG$READ_STRING
                    (&kid,&input,0,&m_len,&modifiers,0,0,0,&terminator,0);
            chk_move(terminator);

        }   while((status) AND
                (terminator NEQ SMG$K_TRM_SELECT) AND
                (terminator NEQ SMG$K_TRM_KP7) AND
                (terminator NEQ SMG$K_TRM_KP3) AND
                (terminator NEQ SMG$K_TRM_RIGHT) AND
                (terminator NEQ SMG$K_TRM_KP1) AND
                (terminator NEQ SMG$K_TRM_LEFT) AND
                (terminator NEQ SMG$K_TRM_REMOVE) AND
                (terminator NEQ SMG$K_TRM_PF3) AND
                (terminator NEQ SMG$K_TRM_HELP) AND
                (terminator NEQ SMG$K_TRM_KP4));

        chk_left_right(terminator);
        chk_main_sel(&exit_flag, terminator);
    }
    endwin();

    save_project();

    printf("B C N U\n");
}
```

```
/********************************************************************************
*                                                                              *
*       DATE: 2/20/86                                                          *
*       VERSION: 1.0                                                           *
*                                                                              *
*           TITLE: KBSEE Executive Routines                                    *
*                      for Command Interpreter and Display Manager             *
*           FILENAME: kbsee_exec.c                                             *
*           COORDINATOR: Capt Dave Fautheree                                   *
*           PROJECT: KBSEE (M.S. Thesis)                                       *
*           OPERATING SYSTEM: VAX/VMS version 4.2                              *
*           LANGUAGE: VAX-11 C                                                 *
*           USE: Include file for KBSEE.C                                      *
*           CONTENTS:                                                          *
*                      bld_copyright - builds copyright window                 *
*                      bld_def - builds defaults window                        *
*                      bld_help - builds help window                          *
*                      bld_main - builds main menu windows                     *
*                      bld_menu_hdr - builds main menu header                  *
*                      bld_menu_1 - builds main menu left column               *
*                      bld_menu_2 - builds main menu second column from left   *
*                      bld_menu_3 - builds main menu third column from left    *
*                      bld_menu_4 - builds main menu column on right           *
*                      bld_mesg - builds messages window                       *
*                      bld_sts - builds status window                         *
*                      chk_left_right - interprets left/right arrow keys       *
*                      chk_main_sel - calls executor routines for menus        *
*                      chk_move - interprets up/down arrow keys & moves choice *
*                      display_menus - displays windows on screeen            *
*                      get_item - highlights current item choice               *
*                      init - initializes windows                             *
*                      update_display - updates screen after spawned process   *
*           FUNCTION: Implements executive functions for display management    *
*                      and command interpretation.                            *
*                                                                              *
********************************************************************************/
```

```
/***************************************************************************
*                                                                         *
*      DATE: 2/20/86                                                       *
*      VERSION: 1.0                                                        *
*                                                                         *
*      NAME: bld_copyright                                                 *
*      MODULE NUMBER: 1.0.1                                                *
*         DESCRIPTION: builds copyright window                             *
*         PASSED VARIABLES:                                                *
*         RETURNS:                                                         *
*         GLOBAL VARIABLES USED: copyright_win                             *
*         GLOBAL VARIABLES CHANGED: copyright_win                          *
*         FILES READ:                                                      *
*         FILES WRITTEN:                                                   *
*         HARDWARE INPUT:                                                  *
*         HARDWARE OUTPUT:                                                 *
*      MODULES CALLED:                                                     *
*                                                                         *
*      CALLING MODULES: init                                              *
*                                                                         *
*         AUTHOR: Capt Dave Fautheree                                      *
*         HISTORY:                                                         *
*                                                                         *
*                                                                         *
****************************************************************************/

bld_copyright()
{
    copyright_win = newwin(24, 80, 0, 0);
    wsetattr(copyright_win, _REVERSE);
    box(copyright_win, ' ', ' ');
    wclrattr(copyright_win, _REVERSE);
    mvwaddstr(copyright_win, 0, 32, "Copyright Notice");
    wsetattr(copyright_win, _BOLD);
    mvwaddstr(copyright_win, 4, 10,
            "KBSEE - A Knowledge Based Software Engineering Environment");
    wclrattr(copyright_win, _BOLD);
    mvwaddstr(copyright_win, 7, 24,
            "Air Force Institute of Technology");
    mvwaddstr(copyright_win, 9, 16,
            "Department of Electrical and Computer Engineering");
    mvwaddstr(copyright_win, 11, 11,
            "Information Sciences/Artificial Intelligence Laboratories");
    mvwaddstr(copyright_win, 18,  3, "AFIT/ENG");
    mvwaddstr(copyright_win, 19,  3, "ATTN: Dr Gary B. Lamont");
    mvwaddstr(copyright_win, 20,  3, "Wright-Patterson AFB, OH 45433");
```

```
    wsetattr(copyright_win, _BOLD);
    mvwaddstr(copyright_win, 18, 46, "(C) Copyright 1985 by");
    mvwaddstr(copyright_win, 19, 50, "David W. Fautheree");
    mvwaddstr(copyright_win, 20, 50, "Gary B. Lamont");
    wclrattr(copyright_win, _BOLD);
    wrefresh(copyright_win);
    sleep(4);
    wrefresh(copyright_win);
    delwin(copyright_win);
}
```

```
/*****************************************************************************
*                                                                           *
*       DATE: 2/20/86                                                        *
*       VERSION: 1.0                                                         *
*                                                                           *
*       NAME: bld_def                                                        *
*       MODULE NUMBER: 1.0.5                                                  *
*          DESCRIPTION: builds defaults window                               *
*          PASSED VARIABLES:                                                 *
*          RETURNS:                                                          *
*          GLOBAL VARIABLES USED:  def_head, def_win                         *
*          GLOBAL VARIABLES CHANGED: def_head, def_win                       *
*          FILES READ:                                                       *
*          FILES WRITTEN:                                                    *
*          HARDWARE INPUT:                                                   *
*          HARDWARE OUTPUT:                                                  *
*       MODULES CALLED:                                                      *
*                                                                           *
*       CALLING MODULES: init                                                *
*                                                                           *
*          AUTHOR: Capt Dave Fautheree                                       *
*          HISTORY:                                                          *
*                                                                           *
*                                                                           *
*****************************************************************************/

bld_def()
{
    def_head = newwin(1, 40, 16, 0);
    def_win = newwin(3, 40, 17, 0);
    box(def_head, ' ', ' ');
    mvwaddstr(def_head, 0, 15, "Defaults");
    mvwaddstr(def_head, 0, 0, "<");
    mvwaddstr(def_head, 0, 39, ">");
    mvwaddstr(def_win, 0, 1, "PROJECT:");
    mvwaddstr(def_win, 1, 1, "WORK FILE:");
    mvwaddstr(def_win, 2, 1, "DIRECTORY:");
    wmove(def_win, 0, 0);
}
```

```
/*****************************************************************************
*                                                                           *
*      DATE: 2/20/86                                                         *
*      VERSION: 1.0                                                          *
*                                                                           *
*      NAME: bld_help                                                        *
*      MODULE NUMBER: 1.0.1                                                  *
*         DESCRIPTION: builds help window                                    *
*         PASSED VARIABLES:                                                  *
*         RETURNS:                                                           *
*         GLOBAL VARIABLES USED: help_win                                    *
*         GLOBAL VARIABLES CHANGED: help_win                                 *
*         FILES READ:                                                        *
*         FILES WRITTEN:                                                     *
*         HARDWARE INPUT:                                                    *
*         HARDWARE OUTPUT:                                                   *
*      MODULES CALLED:                                                       *
*                                                                           *
*      CALLING MODULES: init                                                 *
*                                                                           *
*         AUTHOR: Capt Dave Fautheree                                        *
*         HISTORY:                                                           *
*                                                                           *
*                                                                           *
*****************************************************************************/

bld_help()
{
    help_win = newwin(10, 20, 2, 50);
    wsetattr(help_win, _REVERSE);
    box(help_win, ' ', ' ');
    wclrattr(help_win, _REVERSE);
    mvwaddstr(help_win, 0, 7, "Help");
}
```

```
/***********************************************************************
*                                                                     *
*      DATE: 2/20/86                                                  *
*      VERSION: 1.0                                                   *
*                                                                     *
*      NAME: bld_main                                                 *
*      MODULE NUMBER: 1.0.2                                           *
*         DESCRIPTION: builds main menu windows                      *
*         PASSED VARIABLES:                                          *
*         RETURNS:                                                   *
*         GLOBAL VARIABLES USED:                                     *
*         GLOBAL VARIABLES CHANGED:                                  *
*         FILES READ:                                                *
*         FILES WRITTEN:                                             *
*         HARDWARE INPUT:                                            *
*         HARDWARE OUTPUT:                                           *
*      MODULES CALLED: bld_menu_hdr, bld_menu_1,2,3,4                *
*                                                                     *
*      CALLING MODULES: init                                         *
*                                                                     *
*         AUTHOR: Capt Dave Fautheree                                *
*         HISTORY:                                                   *
*                                                                     *
*                                                                     *
***********************************************************************/

bld_main()
{
    bld_menu_hdr();
    bld_menu_1();
    bld_menu_2();
    bld_menu_3();
    bld_menu_4();
}
```

```
/*********************************************************************
 *                                                                   *
 *      DATE: 2/20/86                                                *
 *      VERSION: 1.0                                                 *
 *                                                                   *
 *      NAME: bld_menu_hdr                                           *
 *      MODULE NUMBER: 1.0.2.0                                       *
 *         DESCRIPTION: builds main menu header                      *
 *         PASSED VARIABLES:                                         *
 *         RETURNS:                                                  *
 *         GLOBAL VARIABLES USED:  menu_hdr_1, menu_hdr_2            *
 *         GLOBAL VARIABLES CHANGED: menu_hdr_1, menu_hdr_2          *
 *         FILES READ:                                              *
 *         FILES WRITTEN:                                            *
 *         HARDWARE INPUT:                                           *
 *         HARDWARE OUTPUT:                                          *
 *      MODULES CALLED:                                             *
 *                                                                   *
 *      CALLING MODULES: bld_main                                   *
 *                                                                   *
 *         AUTHOR: Capt Dave Fautheree                              *
 *         HISTORY:                                                  *
 *                                                                   *
 *                                                                   *
 *********************************************************************/

bld_menu_hdr()
{
    menu_hdr_1 = newwin(1, 80, 0, 0);
    menu_hdr_2 = newwin(1, 80, 1, 0);
    wsetattr(menu_hdr_1, _REVERSE);
    box(menu_hdr_1, ' ', ' ');
    wclrattr(menu_hdr_1, _REVERSE);
    mvwaddstr(menu_hdr_1, 0, 35, "Main Menu");
    wsetattr(menu_hdr_2, _BOLD);
    mvwaddstr(menu_hdr_2, 0, 10,
            "KBSEE - A Knowledge Based Software Engineering Environment");
    wclrattr(menu_hdr_2, _BOLD);
}
```

```
/*************************************************************************
*                                                                       *
*       DATE: 2/20/86                                                   *
*       VERSION: 1.0                                                     *
*                                                                       *
*       NAME: bld_menu_1                                                 *
*       MODULE NUMBER: 1.0.2.1                                          *
*          DESCRIPTION: builds main menu left column                    *
*          PASSED VARIABLES:                                            *
*          RETURNS:                                                     *
*          GLOBAL VARIABLES USED: main_menu_1                           *
*          GLOBAL VARIABLES CHANGED: main_menu_1                        *
*          FILES READ:                                                  *
*          FILES WRITTEN:                                               *
*          HARDWARE INPUT:                                              *
*          HARDWARE OUTPUT:                                             *
*       MODULES CALLED:                                                 *
*                                                                       *
*       CALLING MODULES: bld_main                                       *
*                                                                       *
*          AUTHOR: Capt Dave Fautheree                                  *
*          HISTORY:                                                     *
*                                                                       *
*                                                                       *
*************************************************************************/

bld_menu_1()
{
    main_menu_1 = newwin(11, 16, 3, 8);
    mvwaddstr(main_menu_1, 0, 0, "Build Program  ");
    mvwaddstr(main_menu_1, 1, 0, "Edit           ");
    mvwaddstr(main_menu_1, 2, 0, "Compile        ");
    mvwaddstr(main_menu_1, 3, 0, "Link           ");
    mvwaddstr(main_menu_1, 4, 0, "Run            ");
    mvwaddstr(main_menu_1, 5, 0, "Display Errors ");
    mvwaddstr(main_menu_1, 6, 0, "Start Project  ");
    mvwaddstr(main_menu_1, 7, 0, "Select Project ");
    mvwaddstr(main_menu_1, 8, 0, "List Projects  ");
    mvwaddstr(main_menu_1, 9, 0, "Display Project");
}
```

```
/************************************************************************
*                                                                      *
*     DATE: 2/20/86                                                    *
*     VERSION: 1.0                                                     *
*                                                                      *
*     NAME: bld_menu_2                                                 *
*     MODULE NUMBER: 1.0.2.2                                           *
*        DESCRIPTION: builds main menu second column from left         *
*        PASSED VARIABLES:                                             *
*        RETURNS:                                                      *
*        GLOBAL VARIABLES USED: main_menu_2                            *
*        GLOBAL VARIABLES CHANGED: main_menu_2                         *
*        FILES READ:                                                   *
*        FILES WRITTEN:                                                *
*        HARDWARE INPUT:                                               *
*        HARDWARE OUTPUT:                                              *
*     MODULES CALLED:                                                  *
*                                                                      *
*     CALLING MODULES: bld_main                                        *
*                                                                      *
*        AUTHOR: Capt Dave Fautheree                                  *
*        HISTORY:                                                      *
*                                                                      *
*                                                                      *
************************************************************************/

bld_menu_2()
{
    main_menu_2 = newwin(11, 16, 3, 24);
    mvwaddstr(main_menu_2, 0, 0, "Work File     ");
    mvwaddstr(main_menu_2, 1, 0, "User Profile  ");
    mvwaddstr(main_menu_2, 2, 0, "Introduction  ");
}
```

```
/**********************************************************************
 *                                                                    *
 *      DATE: 2/20/86                                                 *
 *      VERSION: 1.0                                                  *
 *                                                                    *
 *      NAME: bld_menu_3                                              *
 *      MODULE NUMBER: 1.0.2.3                                        *
 *         DESCRIPTION: builds main menu third column from left       *
 *         PASSED VARIABLES:                                          *
 *         RETURNS:                                                   *
 *         GLOBAL VARIABLES USED: main_menu_3                         *
 *         GLOBAL VARIABLES CHANGED: main_menu_3                      *
 *         FILES READ:                                                *
 *         FILES WRITTEN:                                             *
 *         HARDWARE INPUT:                                            *
 *         HARDWARE OUTPUT:                                           *
 *      MODULES CALLED:                                               *
 *                                                                    *
 *      CALLING MODULES: bld_main                                     *
 *                                                                    *
 *         AUTHOR: Capt Dave Fautheree                                *
 *         HISTORY:                                                   *
 *                                                                    *
 *                                                                    *
 **********************************************************************/

bld_menu_3()
{
    main_menu_3 = newwin(11, 16, 3, 40);
    mvwaddstr(main_menu_3, 0, 0, "Debugger        ");
    mvwaddstr(main_menu_3, 1, 0, "Analyzer        ");
    mvwaddstr(main_menu_3, 2, 0, "Librarian       ");
    mvwaddstr(main_menu_3, 3, 0, "Printer         ");
    mvwaddstr(main_menu_3, 4, 0, "Text Formatting");
}
```

```
/************************************************************************
*                                                                      *
*      DATE: 2/20/86                                                    *
*      VERSION: 1.0                                                     *
*                                                                      *
*      NAME: bld_menu_4                                                 *
*      MODULE NUMBER: 1.0.2.4                                           *
*         DESCRIPTION: builds main menu right column                    *
*         PASSED VARIABLES:                                             *
*         RETURNS:                                                      *
*         GLOBAL VARIABLES USED: main_menu_4                            *
*         GLOBAL VARIABLES CHANGED: main_menu_4                         *
*         FILES READ:                                                   *
*         FILES WRITTEN:                                                *
*         HARDWARE INPUT:                                               *
*         HARDWARE OUTPUT:                                              *
*      MODULES CALLED:                                                  *
*                                                                      *
*      CALLING MODULES: bld_main                                        *
*                                                                      *
*         AUTHOR: Capt Dave Fautheree                                   *
*         HISTORY:                                                      *
*                                                                      *
*                                                                      *
************************************************************************/

bld_menu_4()
{
    main_menu_4 = newwin(11, 16, 3, 56);
    mvwaddstr(main_menu_4, 0, 0, "Spawn to CLI    ");
    mvwaddstr(main_menu_4, 1, 0, "Exit            ");
}
```

```
/***************************************************************************
*                                                                         *
*      DATE: 2/20/86                                                      *
*      VERSION: 1.0                                                       *
*                                                                         *
*      NAME: bld_mesg                                                     *
*      MODULE NUMBER: 1.0.3                                               *
*         DESCRIPTION: builds messages window                            *
*         PASSED VARIABLES:                                              *
*         RETURNS:                                                       *
*         GLOBAL VARIABLES USED: mesg_head, mesg_win                     *
*         GLOBAL VARIABLES CHANGED: mesg_head, mesg_win                  *
*         FILES READ:                                                    *
*         FILES WRITTEN:                                                 *
*         HARDWARE INPUT:                                                *
*         HARDWARE OUTPUT:                                               *
*      MODULES CALLED:                                                   *
*                                                                         *
*      CALLING MODULES: init                                            *
*                                                                         *
*         AUTHOR: Capt Dave Fautheree                                   *
*         HISTORY:                                                      *
*                                                                         *
*                                                                         *
****************************************************************************/

bld_mesg()
{
    mesg_win = newwin(3, 80, 21, 0);
    mesg_head = newwin(1, 80, 20, 0);
    wsetattr(mesg_head, _REVERSE);
    box(mesg_head, ' ', ' ');
    wclrattr(mesg_head, _REVERSE);
    mvwaddstr(mesg_head, 0, 36, "Messages");
    wmove(mesg_win, 0, 0);
    scrollok(mesg_win, TRUE);
}
```

```
/*****************************************************************************
*                                                                           *
*     DATE: 2/20/86                                                          *
*     VERSION: 1.0                                                           *
*                                                                           *
*     NAME: bld_sts                                                          *
*     MODULE NUMBER: 1.0.4                                                   *
*        DESCRIPTION: builds status window                                   *
*        PASSED VARIABLES:                                                   *
*        RETURNS:                                                            *
*        GLOBAL VARIABLES USED: sts_head, sts_win                            *
*        GLOBAL VARIABLES CHANGED: sts_head, sts_win                         *
*        FILES READ:                                                         *
*        FILES WRITTEN:                                                      *
*        HARDWARE INPUT:                                                     *
*        HARDWARE OUTPUT:                                                    *
*     MODULES CALLED:                                                        *
*                                                                           *
*     CALLING MODULES: init                                                  *
*                                                                           *
*        AUTHOR: Capt Dave Fautheree                                         *
*        HISTORY:                                                            *
*                                                                           *
*                                                                           *
*****************************************************************************/

bld_sts()
{
    sts_head = newwin(1, 40, 16, 40);
    sts_win = newwin(3, 40, 17, 40);
    box(sts_head, ' ', ' ');
    mvwaddstr(sts_head, 0, 16, "Status");
    mvwaddstr(sts_head, 0, 0, "<");
    mvwaddstr(sts_head, 0, 39, ">");
    wmove(sts_win, 0, 0);
    scrollok(sts_win, TRUE);
}
```

```
/****************************************************************************
*                                                                          *
*      DATE: 2/20/86                                                       *
*      VERSION: 1.0                                                        *
*                                                                          *
*      NAME: chk_left_right                                               *
*      MODULE NUMBER: 1.1                                                  *
*         DESCRIPTION: changes menu when left/right arrow keys are pressed  *
*         PASSED VARIABLES: terminator                                    *
*         RETURNS:                                                        *
*         GLOBAL VARIABLES USED: curr_win, curr_dat                       *
*         GLOBAL VARIABLES CHANGED: curr_win, curr_dat                    *
*         FILES READ:                                                     *
*         FILES WRITTEN:                                                  *
*         HARDWARE INPUT:                                                 *
*         HARDWARE OUTPUT:                                                *
*      MODULES CALLED:                                                    *
*                                                                          *
*      CALLING MODULES: main                                             *
*                                                                          *
*         AUTHOR: Capt Dave Fautheree                                     *
*         HISTORY:                                                        *
*                                                                          *
*                                                                          *
****************************************************************************/


chk_left_right(terminator)
short int terminator;
{
     if ((terminator EQ SMG$K_TRM_KP3) OR
         (terminator EQ SMG$K_TRM_RIGHT))
     {
         wmove(curr_win, curr_dat.curr_y, curr_dat.curr_x);
         wclrattr(curr_win, _REVERSE);
         wprintw(curr_win, "%s", curr_dat.menu_item);
         wrefresh(curr_win);
         if (curr_win EQ main_menu_1)
         {
             curr_win = main_menu_2;
             curr_dat = main_data_2;
         }
         else
         if (curr_win EQ main_menu_2)
         {
             curr_win = main_menu_3;
```

```
        curr_dat = main_data_3;
    }
    else
    if (curr_win EQ main_menu_3)
    {
        curr_win = main_menu_4;
        curr_dat = main_data_4;
    }
    else
    if (curr_win EQ main_menu_4)
    {
        curr_win = main_menu_1;
        curr_dat = main_data_1;
    }
}
else
if ((terminator EQ SMG$K_TRM_KP1) OR
    (terminator EQ SMG$K_TRM_LEFT))
{
    wmove(curr_win, curr_dat.curr_y, curr_dat.curr_x);
    wclrattr(curr_win, _REVERSE);
    wprintw(curr_win, "%s", curr_dat.menu_item);
    wrefresh(curr_win);
    if (curr_win EQ main_menu_1)
    {
        curr_win = main_menu_4;
        curr_dat = main_data_4;
    }
    else
    if (curr_win EQ main_menu_2)
    {
        curr_win = main_menu_1;
        curr_dat = main_data_1;
    }
    else
    if (curr_win EQ main_menu_3)
    {
        curr_win = main_menu_2;
        curr_dat = main_data_2;
    }
    else
    if (curr_win EQ main_menu_4)
    {
        curr_win = main_menu_3;
        curr_dat = main_data_3;
    }
```

```
      }
  }
```

```
/*************************************************************************
 *                                                                       *
 *      DATE: 2/20/86                                                     *
 *      VERSION: 1.0                                                      *
 *                                                                       *
 *      NAME: chk_main_sel                                                *
 *      MODULE NUMBER: 1.8                                                *
 *         DESCRIPTION: calls appropriate menu selection processing routines *
 *         PASSED VARIABLES: exit_now, terminator                        *
 *         RETURNS:                                                       *
 *         GLOBAL VARIABLES USED: curr_win, main_menu_1,2,3,4            *
 *         GLOBAL VARIABLES CHANGED:                                     *
 *         FILES READ:                                                    *
 *         FILES WRITTEN:                                                 *
 *         HARDWARE INPUT:                                                *
 *         HARDWARE OUTPUT:                                               *
 *      MODULES CALLED: chk_main_1, chk_main_2, chk_main_3, chk_main_4   *
 *                                                                       *
 *      CALLING MODULES: main                                            *
 *                                                                       *
 *         AUTHOR: Capt Dave Fautheree                                   *
 *         HISTORY:                                                       *
 *                                                                       *
 *                                                                       *
 *************************************************************************/

chk_main_sel(exit_now, terminator)
int *exit_now;
short int terminator;
{
    if ((terminator EQ SMG$K_TRM_KP7) OR (terminator EQ SMG$K_TRM_SELECT))
    {
        if (curr_win EQ main_menu_1)
        {
            chk_main_1();
        }
        else
        if (curr_win EQ main_menu_2)
        {
            chk_main_2();
        }
        else
        if (curr_win EQ main_menu_3)
        {
            chk_main_3();
        }
```

```
        else
        if (curr_win EQ main_menu_4)
        {
            chk_main_4();
        }
    }
    else
    if ((terminator EQ SMG$K_TRM_KP4) OR (terminator EQ SMG$K_TRM_HELP))
    {
        wprintw(mesg_win, "\nHelp selected");
    }
    else
    if ((terminator EQ SMG$K_TRM_PF3) OR (terminator EQ SMG$K_TRM_REMOVE))
    {
        *exit_now = TRUE;
    }
}
```

```
/*******************************************************************************
 *                                                                             *
 *      DATE: 2/20/86                                                          *
 *      VERSION: 1.0                                                           *
 *                                                                             *
 *      NAME: chk_move                                                         *
 *      MODULE NUMBER: 1.6                                                     *
 *         DESCRIPTION: moves current menu item selection up or down          *
 *         PASSED VARIABLES: terminator                                       *
 *         RETURNS:                                                           *
 *         GLOBAL VARIABLES USED: curr_win, curr_dat                          *
 *         GLOBAL VARIABLES CHANGED: curr_dat                                 *
 *         FILES READ:                                                        *
 *         FILES WRITTEN:                                                     *
 *         HARDWARE INPUT:                                                    *
 *         HARDWARE OUTPUT:                                                   *
 *      MODULES CALLED:                                                       *
 *                                                                             *
 *      CALLING MODULES: main                                                 *
 *                                                                             *
 *         AUTHOR: Capt Dave Fautheree                                        *
 *         HISTORY:                                                           *
 *                                                                             *
 *                                                                             *
 *******************************************************************************/

chk_move(terminator)
short int terminator;
{
    if ((terminator EQ SMG$K_TRM_KP2) OR (terminator EQ SMG$K_TRM_DOWN))
    {
        wmove(curr_win, curr_dat.curr_y, curr_dat.curr_x);     /* UnReverse Ite
*/
        wclrattr(curr_win, _REVERSE);
        wprintw(curr_win, "%s", curr_dat.menu_item);
        curr_dat.curr_y = curr_dat.curr_y + 1;
        if (curr_dat.curr_y > curr_dat.max_y)        /* Wrap to top of menu */
            curr_dat.curr_y = curr_dat.min_y;
    }
    else
    if ((terminator EQ SMG$K_TRM_KP5) OR (terminator EQ SMG$K_TRM_UP))
    {
        wmove(curr_win, curr_dat.curr_y, curr_dat.curr_x);     /* UnReverse Ite
*/
        wclrattr(curr_win, _REVERSE);
        wprintw(curr_win, "%s", curr_dat.menu_item);
```

```
        curr_dat.curr_y = curr_dat.curr_y - 1;
        if (curr_dat.curr_y < curr_dat.min_y)            /* Wrap to bottom of men
    */

            curr_dat.curr_y = curr_dat.max_y;

    }
}
```

```
/*********************************************************************
 *                                                                   *
 *    DATE: 2/20/86                                                  *
 *    VERSION: 1.0                                                   *
 *                                                                   *
 *    NAME: display_menus                                            *
 *    MODULE NUMBER: 1.1                                             *
 *       DESCRIPTION: displays newly initialized menus on screen     *
 *       PASSED VARIABLES:                                           *
 *       RETURNS:                                                    *
 *       GLOBAL VARIABLES USED: menu_hdr_1,2 main_menu_1,2,3,4 sts_win,head *
 *                              mesg_win,head def_win,head            *
 *       GLOBAL VARIABLES CHANGED:                                  *
 *       FILES READ:                                                *
 *       FILES WRITTEN:                                             *
 *       HARDWARE INPUT:                                            *
 *       HARDWARE OUTPUT:                                           *
 *    MODULES CALLED:                                               *
 *                                                                   *
 *    CALLING MODULES: init                                          *
 *                                                                   *
 *       AUTHOR: Capt Dave Fautheree                                 *
 *       HISTORY:                                                    *
 *                                                                   *
 *                                                                   *
 *********************************************************************/

display_menus()
{
    refresh();
    wrefresh(menu_hdr_1);
    wrefresh(menu_hdr_2);
    wrefresh(main_menu_1);
    wrefresh(main_menu_2);
    wrefresh(main_menu_3);
    wrefresh(main_menu_4);
    wrefresh(mesg_head);
    wrefresh(mesg_win);
    wrefresh(sts_head);
    wrefresh(sts_win);
    wrefresh(def_win);
    wrefresh(def_head);
}
```

```
/*******************************************************************************
 *                                                                             *
 *      DATE: 2/20/86                                                          *
 *      VERSION: 1.0                                                           *
 *                                                                             *
 *      NAME: get_item                                                         *
 *      MODULE NUMBER: 1.5                                                     *
 *         DESCRIPTION: reads and highlights current selection from menu       *
 *         PASSED VARIABLES:                                                   *
 *         RETURNS:                                                            *
 *         GLOBAL VARIABLES USED:                                             *
 *         GLOBAL VARIABLES CHANGED:                                          *
 *         FILES READ:                                                        *
 *         FILES WRITTEN:                                                     *
 *         HARDWARE INPUT:                                                    *
 *         HARDWARE OUTPUT:                                                   *
 *      MODULES CALLED:                                                       *
 *                                                                             *
 *      CALLING MODULES: main                                                 *
 *                                                                             *
 *         AUTHOR: Capt Dave Fautheree                                        *
 *         HISTORY:                                                           *
 *                                                                             *
 *                                                                             *
 *******************************************************************************/

get_item(win, menu_dat)
WINDOW *win;
struct menu_data *menu_dat;
{
    int position = 0;
    int tmp_x = menu_dat->curr_x;

    while(position < ITEM_LENGTH)
    {
        wmove(win, menu_dat->curr_y, tmp_x);
        menu_dat->menu_item[position] = winch(win);
        tmp_x += 1;
        position += 1;
    }
        menu_dat->menu_item[position] = '\0';

        wmove(win, menu_dat->curr_y, menu_dat->curr_x);
        wsetattr(win, _REVERSE);
        wprintw(win, "%s", menu_dat->menu_item);
        wclrattr(win, _REVERSE);
```

```
    wrefresh(win);
}
```

```
/*******************************************************************************
 *                                                                             *
 *      DATE: 2/20/86                                                          *
 *      VERSION: 1.0                                                           *
 *                                                                             *
 *      NAME: init                                                            *
 *      MODULE NUMBER: 1.0                                                     *
 *         DESCRIPTION: initializes windows                                    *
 *         PASSED VARIABLES:                                                   *
 *         RETURNS:                                                            *
 *         GLOBAL VARIABLES USED:                                             *
 *         GLOBAL VARIABLES CHANGED:                                          *
 *         FILES READ:                                                        *
 *         FILES WRITTEN:                                                     *
 *         HARDWARE INPUT:                                                    *
 *         HARDWARE OUTPUT:                                                   *
 *      MODULES CALLED:                                                       *
 *                                                                             *
 *      CALLING MODULES: main                                                 *
 *                                                                             *
 *         AUTHOR: Capt Dave Fautheree                                         *
 *         HISTORY:                                                            *
 *                                                                             *
 *                                                                             *
 *******************************************************************************/

init()
{
    initscr();

    bld_help();
    bld_copyright();
    bld_main();
    bld_mesg();
    bld_sts();
    bld_def();
}
```

```
/******************************************************************
*                                                                *
*     DATE: 2/20/86                                              *
*     VERSION: 1.0                                              *
*                                                                *
*     NAME: update_display                                      *
*     MODULE NUMBER: 0.1                                         *
*         DESCRIPTION: updates windows after spawn              *
*         PASSED VARIABLES:                                      *
*         RETURNS:                                               *
*         GLOBAL VARIABLES USED: menu_hdr_1,2 main_menu_1,2,3,4 sts_win,head *
*                                mesg_win,head def_win,head      *
*         GLOBAL VARIABLES CHANGED:                             *
*         FILES READ:                                            *
*         FILES WRITTEN:                                         *
*         HARDWARE INPUT:                                        *
*         HARDWARE OUTPUT:                                       *
*     MODULES CALLED:                                            *
*                                                                *
*     CALLING MODULES: executive routines with spawns           *
*                                                                *
*         AUTHOR: Capt Dave Fautheree                           *
*         HISTORY:                                               *
*                                                                *
*                                                                *
******************************************************************/

update_display()
{
    touchwin(stdscr);
    touchwin(menu_hdr_1);
    touchwin(menu_hdr_2);
    touchwin(main_menu_1);
    touchwin(main_menu_2);
    touchwin(main_menu_3);
    touchwin(main_menu_4);
    touchwin(mesg_head);
    touchwin(mesg_head);
    touchwin(mesg_head);
    touchwin(mesg_win);
    touchwin(sts_head);
    touchwin(sts_win);
    touchwin(def_win);
    touchwin(def_head);
}
```

```
/*******************************************************************************
*                                                                             *
*      DATE: 2/20/86                                                          *
*      VERSION: 1.0                                                           *
*                                                                             *
*         TITLE: KBSEE Project Manager Routines                              *
*         FILENAME: kbsee_proj.c                                             *
*         COORDINATOR: Capt Dave Fautheree                                   *
*         PROJECT: KBSEE (M.S. Thesis)                                       *
*         OPERATING SYSTEM: VAX/VMS version 4.2                              *
*         LANGUAGE: VAX-11 C                                                 *
*         USE: Include file for KBSEE.C                                      *
*         CONTENTS:                                                          *
*                 bld_copyright - builds copyright window                    *
*         FUNCTION: Implements executive functions for display management    *
*                 and command interpretation.                                *
*                                                                             *
*******************************************************************************/
```

```
/********************************************************************
 *                                                                  *
 *      DATE: 2/20/86                                               *
 *      VERSION: 1.0                                                *
 *                                                                  *
 *      NAME: chk_save_file                                         *
 *      MODULE NUMBER: 1.2                                          *
 *         DESCRIPTION: checks for save file and loads it, if found *
 *         PASSED VARIABLES:                                        *
 *         RETURNS: TRUE if file exists                             *
 *         GLOBAL VARIABLES USED: curr_proj, mesg_win, sts-win      *
 *         GLOBAL VARIABLES CHANGED: curr_proj                      *
 *         FILES READ: kbsee.save                                   *
 *         FILES WRITTEN:                                           *
 *         HARDWARE INPUT:                                          *
 *         HARDWARE OUTPUT:                                         *
 *      MODULES CALLED:                                             *
 *                                                                  *
 *      CALLING MODULES: main                                       *
 *                                                                  *
 *         AUTHOR: Capt Dave Fautheree                              *
 *         HISTORY:                                                 *
 *                                                                  *
 ********************************************************************/

int     chk_save_file()
{
    int      i = 0;
    FILE     *fptr;
    int ret_val = FALSE;

    wprintw(sts_win, "\nChecking SAVE File...");
    wrefresh(sts_win);
    if (access("kbsee.save", 4) EQ 0)
    {
        ret_val = TRUE;
        fptr = fopen("sys$login:kbsee.save", "r");
        fgets(curr_proj.project_name, SIZE_NAME, fptr);
        fgets(curr_proj.work_file, SIZE_WORK_FILE, fptr);
        fclose(fptr);

        while(curr_proj.project_name[i] NEQ ' ')
            i++;
        curr_proj.project_name[i] = '\0';
```

```
        i = 0;
        while(curr_proj.work_file[i] NEQ ' ')
            i++;
        curr_proj.work_file[i] = '\0';
    }
    else
    {
        wprintw(sts_win, "\nNo Current Project...");
        wrefresh(sts_win);
        wsetattr(mesg_win, _BLINK | _REVERSE);
        wprintw(mesg_win, "\nEstablish a Project and Profile IMMEDIATELY");
        wclrattr(mesg_win, _BLINK | _REVERSE);
    }
    wrefresh(mesg_win);
    wrefresh(def_win);
    return(ret_val);
}
```

```
/*****************************************************************
*                                                               *
*     DATE: 2/20/86                                             *
*     VERSION: 1.0                                             *
*                                                               *
*     NAME: exec_start_project                                 *
*     MODULE NUMBER: 1.8.0.6                                    *
*        DESCRIPTION: starts a new project                     *
*        PASSED VARIABLES:                                     *
*        RETURNS:                                              *
*        GLOBAL VARIABLES USED: curr_proj, mesg_win, sts-win, def_win *
*        GLOBAL VARIABLES CHANGED: curr_proj                   *
*        FILES READ:                                          *
*        FILES WRITTEN: kbsee.projects                        *
*        HARDWARE INPUT:                                      *
*        HARDWARE OUTPUT:                                     *
*     MODULES CALLED:                                         *
*                                                               *
*     CALLING MODULES: chk_main_1                             *
*                                                               *
*        AUTHOR: Capt Dave Fautheree                          *
*        HISTORY:                                             *
*                                                               *
*****************************************************************/

exec_start_project()
{
    int     i;
    FILE    *fptr;

    wprintw(mesg_win, "\nEnter Project Name: ");
    wscanw(mesg_win, "%s", curr_proj.project_name);
    wprintw(mesg_win, "Enter Project Location: ");
    wscanw(mesg_win, "%s", curr_proj.location);
    wprintw(mesg_win, "Enter Name of Work File: ");
    wscanw(mesg_win, "%s", curr_proj.work_file);

    chdir(curr_proj.location);

    wmove(def_win, 0, 12);
    wclrtoeol(def_win);
    wmove(def_win, 1, 12);
    wclrtoeol(def_win);
    wmove(def_win, 2, 12);
    wclrtoeol(def_win);
```

```
    wsetattr(def_win, _REVERSE);
    mvwaddstr(def_win, 0, 12, curr_proj.project_name);
    mvwaddstr(def_win, 1, 12, curr_proj.work_file);
    mvwaddstr(def_win, 2, 12, curr_proj.location);
    wclrattr(def_win, _REVERSE);
    wrefresh(def_win);

    for (i = strlen(curr_proj.project_name); i < SIZE_NAME; i++)
        curr_proj.project_name[i] = ' ';
    curr_proj.project_name[SIZE_NAME - 1] = '\0';

    for (i = strlen(curr_proj.work_file); i < SIZE_WORK_FILE; i++)
        curr_proj.work_file[i] = ' ';
    curr_proj.work_file[SIZE_WORK_FILE - 1] = '\0';

    for (i = strlen(curr_proj.location); i < SIZE_LOC; i++)
        curr_proj.location[i] = ' ';
    curr_proj.location[SIZE_LOC - 1] = '\0';

    fptr = fopen("sys$login:kbsee.projects", "a");
    fputs(curr_proj.project_name, fptr);
    fputs(curr_proj.work_file, fptr);
    fputs(curr_proj.location, fptr);
    fclose(fptr);

    wprintw(sts_win, "\nProject Added Successfully...");
    wrefresh(sts_win);
}
```

```
/**********************************************************************
*                                                                    *
*     DATE: 2/20/86                                                  *
*     VERSION: 1.0                                                   *
*                                                                    *
*     NAME: get_project                                             *
*     MODULE NUMBER: 1.3                                             *
*        DESCRIPTION: loads project from project database           *
*        PASSED VARIABLES:                                          *
*        RETURNS:                                                   *
*        GLOBAL VARIABLES USED: curr_proj, mesg_win, sts-win        *
*        GLOBAL VARIABLES CHANGED: curr_proj                        *
*        FILES READ: kbsee.save                                     *
*        FILES WRITTEN:                                             *
*        HARDWARE INPUT:                                            *
*        HARDWARE OUTPUT:                                           *
*     MODULES CALLED:                                              *
*                                                                    *
*     CALLING MODULES: main                                        *
*                                                                    *
*        AUTHOR: Capt Dave Fautheree                               *
*        HISTORY:                                                   *
*                                                                    *
*                                                                    *
**********************************************************************/

get_project()
{
    FILE      *fptr;
    int       i = 0;
    int       found = 0;
    char      proj[SIZE_NAME];
    char      wfile[SIZE_WORK_FILE];

    wprintw(sts_win, "\nLoading Current Project...");
    wrefresh(sts_win);

    fptr = fopen("sys$login:kbsee.projects", "r");

    while ( (feof(fptr) EQ 0) AND (found NEQ 1))
    {
        fgets(proj, SIZE_NAME, fptr);
        fgets(wfile, SIZE_WORK_FILE, fptr);
        fgets(curr_proj.location, SIZE_LOC, fptr);

        while(proj[i] NEQ ' ')
```

```
        i++;
    proj[i] = '\0';
    i = 0;
    while(wfile[i] NEQ ' ')
        i++;
    wfile[i] = '\0';
    i = 0;
    while(curr_proj.location[i] NEQ ' ')
        i++;
    curr_proj.location[i] = '\0';

    wprintw(mesg_win,"\nRead = %s",proj);
    wprintw(mesg_win,"|");
    wprintw(mesg_win,"\nRead = %s",wfile);
    wprintw(mesg_win,"|");
    wrefresh(mesg_win);

    if ( (strcmp(curr_proj.project_name, proj) EQ 0) &&
         (strcmp(curr_proj.work_file, wfile) EQ 0))
    {
        found = 1;
    }
}

if (found EQ 0)
{
    wprintw(sts_win,"\nProject Error - No Match");
}
else
{
    chdir(curr_proj.location);

    wmove(def_win, 0, 12);
    wclrtoeol(def_win);
    wmove(def_win, 1, 12);
    wclrtoeol(def_win);
    wmove(def_win, 2, 12);
    wclrtoeol(def_win);

    wsetattr(def_win, _REVERSE);
    mvwaddstr(def_win, 0, 12, curr_proj.project_name);
    mvwaddstr(def_win, 1, 12, curr_proj.work_file);
    mvwaddstr(def_win, 2, 12, curr_proj.location);
    wclrattr(def_win, _REVERSE);
    wrefresh(def_win);
}
```

```
    wrefresh(sts_win);
    fclose(fptr);
}
```

```
/**********************************************************************
*                                                                    *
*      DATE: 2/20/86                                                  *
*      VERSION: 1.0                                                   *
*                                                                    *
*      NAME: save_project                                            *
*      MODULE NUMBER: 1.9                                             *
*         DESCRIPTION: saves current project into save file          *
*         PASSED VARIABLES:                                           *
*         RETURNS:                                                    *
*         GLOBAL VARIABLES USED: curr_proj                           *
*         GLOBAL VARIABLES CHANGED:                                   *
*         FILES READ: kbsee.save                                     *
*         FILES WRITTEN:                                              *
*         HARDWARE INPUT:                                             *
*         HARDWARE OUTPUT:                                            *
*      MODULES CALLED:                                               *
*                                                                    *
*      CALLING MODULES: main                                         *
*                                                                    *
*         AUTHOR: Capt Dave Fautheree                               *
*         HISTORY:                                                   *
*                                                                    *
**********************************************************************/

save_project()
{
    int     i;
    FILE    *fptr;

    delete("sys$login:kbsee.save");

    fptr = fopen("sys$login:kbsee.save", "w");

    for (i = strlen(curr_proj.project_name); i < SIZE_NAME; i++)
        curr_proj.project_name[i] = ' ';
    curr_proj.project_name[SIZE_NAME - 1] = '\0';

    for (i = strlen(curr_proj.work_file); i < SIZE_WORK_FILE; i++)
        curr_proj.work_file[i] = ' ';
    curr_proj.work_file[SIZE_WORK_FILE - 1] = '\0';

    fputs(curr_proj.project_name, fptr);
    fputs(curr_proj.work_file, fptr);
```

```
    fclose(fptr);
}
```

Source Code Listing

for the

Knowledge Based

Software Module Analyzer

(KBSMA)

```
;**************************************************************************
;*          DATE:  11/9/85                                               *
;*          VERSION:  1.1                                                *
;*                                                                       *
;*          TITLE:  OPS-5 Data Structures and Production Rules for       *
;*                  a Prototype Software Engineering Analysis Tool       *
;*          FILENAME:  KBSMA.OPS                                         *
;*          COORDINATOR:  Capt Dave Fautheree, GCS-86M                   *
;*          PROJECT:  MS Thesis                                          *
;*          OPERATING SYSTEM:  VAX/VMS 4.2                               *
;*          LANGUAGE:  OPS-5                                             *
;*          USE: RUN KBSMA                                               *
;*          CONTENTS:                                                    *
;*                      Module                                           *
;*                      Variable                                         *
;*                      Cohesion-Answers                                 *
;*                      Coupling::Data:1                                 *
;*                      Coupling::Data:2                                 *
;*                      Coupling::Data:3                                 *
;*                      Coupling::Stamp:1                                *
;*                      Coupling::Stamp:2                                *
;*                      Coupling::Stamp:3                                *
;*                      Coupling::Control:1                              *
;*                      Coupling::Control:2                              *
;*                      Coupling::Control:3                              *
;*                      Coupling::Common:1                               *
;*                      Coupling::Common:2                               *
;*                      Coupling::Common:3                               *
;*                      Cohesion::One-Function                           *
;*                      Cohesion::Activities-Related                     *
;*                      Cohesion::Sequence-Important                     *
;*                      Cohesion::Same-Category                          *
;*                      Cohesion::Functional                             *
;*                      Cohesion::Sequential                             *
;*                      Cohesion::Communcational                         *
;*                      Cohesion::Procedural                             *
;*                      Cohesion::Temporal                               *
;*                      Cohesion::Logical                                *
;*                      Cohesion::Coincidental                           *
;*                                                                       *
;*          FUNCTION:  Define data structures and production rules       *
;*                     for a Knowledge Based Software Module Analyzer,   *
;*                     a Software Engineering tool for determining       *
;*                     module coupling and cohesion.                     *
;*                                                                       *
;**************************************************************************
```

```
;
;       Define Module data structure
;

(literalize Module                      ; The following data items are defined from
                                        ; the AFIT/ENG Software Development Guidelines
             module-name
             project
             module-number
             description
             ;  passed-variables
                  passed-variable-1
                  passed-variable-2
                  passed-variable-3
             returns
             ;  globals-used
                  global-used-1
                  global-used-2
                  global-used-3
             ;  globals-changed
                  global-changed-1
                  global-changed-2
                  global-changed-3
             files-read
             files-written
             calling-modules
             modules-called
             version
             date
             author
             filename

             coupling-type
             cohesion-type
             reccomendation)

;
;       Define Variable data structure
;

(literalize Variable
             variable-name
             type                       ; SIMPLE or RECORD
             control)                   ; YES or NO - control variable from
                                        ;              another module

;
;       Define Cohesion Answers data structure
```

```
;

(literalize Cohesion-Answers
            module-name
            one-function               ;   YES or NO
            activities-related         ;   DATA CONTROL or NEITHER
            sequence-important         ;   YES or NO
            same-category)             ;   YES or NO


;
;   Define KBSMA Startup for OPS5
;

(startup
    (watch 0)
    (disable halt)
    (strategy lex)
    (@ kbsma_instances.dat)
    (run))
```

```
;********************************************************************
;*      DATE:   5/24/85                                           *
;*      VERSION: 1.0                                              *
;*                                                                *
;*      NAME:  PrintModule                                        *
;*      DESCRIPTION: Prints deduced results for each module       *
;*                                                                *
;*      ALGORITHM:                                                *
;*                IF the module state is complete                 *
;*                THEN print the module information               *
;*                                                                *
;*      AUTHOR: Capt Dave Fautheree                               *
;********************************************************************

(p PrintModule
    {(Module ^module-name <n1>
             ^coupling-type <n2>
             ^cohesion-type <n3>
             ^reccomendation <n4>
             ^coupling-type <> nil
             ^cohesion-type <> nil) <module>}
-->
    (write (crlf)Module <n1> (crlf))
    (write    Coupling: <n2> (crlf))
    (write    Cohesion: <n3> (crlf))
    (write    Reccomendation: <n4> (crlf) (crlf))
    (remove <module>))
```

```
;*********************************************************************
;*        DATE:  5/24/85                                            *
;*        VERSION: 1.0                                              *
;*                                                                  *
;*        NAME:  Coupling::Common                                   *
;*        DESCRIPTION: Production Rules for common coupling         *
;*                                                                  *
;*        ALGORITHM:                                                *
;*                   IF the module uses a global and                *
;*                      coupling has not been determined            *
;*                   THEN set module coupling to common and         *
;*                      give a reccomendation and                   *
;*                                                                  *
;*        AUTHOR: Capt Dave Fautheree                               *
;*                                                                  *
;*********************************************************************

(p Coupling::Common:1
    {(Module ^coupling-type nil
             ^global-used-1 <globall>
             ^global-used-1 <> nil) <module>}
-->
    (modify <module> ^coupling-type Common
                     ^reccomendation |pass the required data item|))


(p Coupling::Common:2
    {(Module ^coupling-type nil
             ^global-used-2 <global2>
             ^global-used-2 <> nil) <module>}
-->
    (modify <module> ^coupling-type Common
                     ^reccomendation |pass the required data item|))


(p Coupling::Common:3
    {(Module ^coupling-type nil
             ^global-used-3 <global3>
             ^global-used-3 <> nil) <module>}
-->
    (modify <module> ^coupling-type Common
                     ^reccomendation |pass the required data item|))
```

```
;*********************************************************************
;*      VERSION: 1.0                                                *
;*                                                                  *
;*      NAME:  Coupling::Control                                    *
;*      DESCRIPTION: Production Rules for control coupling          *
;*                                                                  *
;*      ALGORITHM:                                                  *
;*              IF a module has a parameter and                     *
;*                 coupling has not been determined                 *
;*              AND the parameter is defined and                    *
;*                 it is a control variable                         *
;*              THEN set module coupling to control and             *
;*                 give a reccomendation                            *
;*                                                                  *
;*      AUTHOR: Capt Dave Fautheree                                 *
;*                                                                  *
;*********************************************************************

(p Coupling::Control:1
    {(Module ^coupling-type nil
             ^passed-variable-1 <param>
             ^passed-variable-1 <> nil
             ^global-used-1 nil
             ^global-used-2 nil
             ^global-used-3 nil) <module>}
    {(Variable ^variable-name <param>
             ^control Yes) <control>}
-->
    (modify <module> ^coupling-type Control
                     ^reccomendation |not use imported control information|))


(p Coupling::Control:2
    {(Module ^coupling-type nil
             ^passed-variable-2 <param>
             ^passed-variable-2 <> nil
             ^global-used-1 nil
             ^global-used-2 nil
             ^global-used-3 nil) <module>}
    {(Variable ^variable-name <param>
             ^control Yes) <control>}
-->
    (modify <module> ^coupling-type Control
                     ^reccomendation |not use imported control information|))


(p Coupling::Control:3
    {(Module ^coupling-type nil
```

```
            ^passed-variable-3 <param>
            ^passed-variable-3 <> nil
            ^global-used-1 nil
            ^global-used-2 nil
            ^global-used-3 nil) <module>}
   {(Variable ^variable-name <param>
            ^control Yes) <control>}
-->

   (modify <module> ^coupling-type Control
                    ^reccomendation |not use imported control information|))
```

```
;*****************************************************************
;*      DATE:   5/24/85                                          *
;*      VERSION: 1.0                                             *
;*                                                               *
;*      NAME:  Coupling::Stamp                                   *
;*      DESCRIPTION: Production Rule for determining stamp coupling *
;*                                                               *
;*      ALGORITHM:                                               *
;*                  IF a module has a parameter and              *
;*                     coupling has not been determined          *
;*                  AND the parameter is defined and             *
;*                     its type is Record and                    *
;*                     it is not a control variable              *
;*                  THEN set module coupling to stamp and        *
;*                     give a reccomendation                     *
;*                                                               *
;*      AUTHOR: Capt Dave Fautheree                              *
;*                                                               *
;*****************************************************************

(p Coupling::Stamp:1
     {(Module ^coupling-type nil
              ^passed-variable-1 <param>
              ^passed-variable-1 <> nil
              ^global-used-1 nil
              ^global-used-2 nil
              ^global-used-3 nil) <module>}
     {(Variable ^variable-name <param>
              ^type Record
              ^control No) <record>}
 -->
     (modify <module> ^coupling-type Stamp
                  ^reccomendation |only pass in specific data items|))


(p Coupling::Stamp:2
     {(Module ^coupling-type nil
              ^passed-variable-2 <param>
              ^passed-variable-2 <> nil
              ^global-used-1 nil
              ^global-used-2 nil
              ^global-used-3 nil) <module>}
     {(Variable ^variable-name <param>
              ^type Record
              ^control No) <record>}
 -->
     (modify <module> ^coupling-type Stamp
                  ^reccomendation |only pass in specific data items|))
```

```
(p Coupling::Stamp:3
      {(Module ^coupling-type nil
               ^passed-variable-3 <param>
               ^passed-variable-3 <> nil
               ^global-used-1 nil
               ^global-used-2 nil
               ^global-used-3 nil) <module>}
      {(Variable ^variable-name <param>
                 ^type Record
                 ^control No) <record>}
  -->

      (modify <module> ^coupling-type Stamp
                       ^reccomendation |only pass in specific data items|))
```

```
;********************************************************************
;*      DATE:  5/24/85                                             *
;*      VERSION: 1.0                                               *
;*                                                                 *
;*      NAME:  Coupling::Data                                      *
;*      DESCRIPTION: Production Rule for determining data coupling *
;*                                                                 *
;*      ALGORITHM:                                                 *
;*                  IF a module has a parameter and               *
;*                     coupling has not been determined           *
;*                  AND the parameter is defined and              *
;*                     its type is Simple and                     *
;*                     it is not a control variable               *
;*                  THEN set module coupling to data and          *
;*                       give a reccomendation                    *
;*                                                                 *
;*      AUTHOR: Capt Dave Fautheree                                *
;*                                                                 *
;********************************************************************

(p Coupling::Data:1
     {(Module ^coupling-type nil
              ^passed-variable-1 <param>
              ^passed-variable-1 <> nil
              ^global-used-1 nil
              ^global-used-2 nil
              ^global-used-3 nil) <module>}
     {(Variable ^variable-name <param>
                ^type Simple
                ^control No) <data>}
  -->
     (modify <module> ^coupling-type Data
                      ^reccomendation |keep up the good work|))


(p Coupling::Data:2
     {(Module ^coupling-type nil
              ^passed-variable-2 <param>
              ^passed-variable-2 <> nil
              ^global-used-1 nil
              ^global-used-2 nil
              ^global-used-3 nil) <module>}
     {(Variable ^variable-name <param>
                ^type Simple
                ^control No) <data>}
  -->
     (modify <module> ^coupling-type Data
                      ^reccomendation |keep up the good work|))
```

```
(p Coupling::Data:3
    {(Module ^coupling-type nil
             ^passed-variable-3 <param>
             ^passed-variable-3 <> nil
             ^global-used-1 nil
             ^global-used-2 nil
             ^global-used-3 nil) <module>}
    {(Variable ^variable-name <param>
               ^type Simple
               ^control No) <data>}
  -->
    (modify <module> ^coupling-type Data
                     ^reccomendation |keep up the good work|))
```

```
;*****************************************************************
;*        DATE:   5/24/85                                        *
;*        VERSION: 1.0                                           *
;*                                                               *
;*        NAME:   Cohesion::Questions                            *
;*        DESCRIPTION: Production Rules for asking Cohesion questions *
;*                                                               *
;*        ALGORITHM:                                             *
;*          ·          IF cohesion has not been determined       *
;*                     AND there is insufficient data to deduce it *
;*                     THEN ask the appropriate question          *
;*                                                               *
;*        AUTHOR: Capt Dave Fautheree                            *
;*                                                               *
;*****************************************************************

(p Cohesion::One-Function
   {(Module ^cohesion-type nil
            ^module-name <name>) <module>}
-->
   (write (crlf)Is module <name> doing only one function? |(Yes or No)| )
   (make Cohesion-Answers ^module-name <name>
                          ^activities-related nil
                          ^one-function (accept)))


(p Cohesion::Relate-Activities
   {(Module ^cohesion-type nil
            ^module-name <name>) <module>}
   {(Cohesion-Answers ^module-name <name>
                      ^activities-related nil
                      ^one-function No) <cohesion>}
-->
   (write (crlf)In module <name> what relates the activities?)
   (write |(Data Control Neither)| )
   (modify <cohesion> ^activities-related (accept)))


(p Cohesion::Sequence-Important
   {(Module ^cohesion-type nil
            ^module-name <name>) <module>}
   {(Cohesion-Answers ^module-name <name>
                      ^activities-related << Data Control >>) <cohesion>}
-->
   (write (crlf)In module <name> is the sequence important?)
   (write |(Yes No)| )
   (modify <cohesion> ^sequence-important (accept)))
```

```
(p Cohesion::Same-Category
    {(Module ^cohesion-type nil
             ^module-name <name>) <module>}
    {(Cohesion-Answers ^module-name <name>
                       ^activities-related Neither) <cohesion>}
  -->
    (write (crlf)In module <name> are the activities in the)
    (write same general category?)
    (write |(Yes No)| )
    (modify <cohesion> ^same-category (accept)))
```

```
;********************************************************************
;*        DATE:  5/24/85                                          *
;*        VERSION: 1.0                                            *
;*                                                               *
;*        NAME:   Cohesion::XXXXXXXX                             *
;*        DESCRIPTION: Production Rules for determining Cohesion  *
;*                                                               *
;*        ALGORITHM:                                             *
;*                    IF cohesion has not been determined         *
;*                    AND there is sufficient data to deduce it   *
;*                    THEN deduce it and remove Answer structure  *
;*                                                               *
;*        AUTHOR: Capt Dave Fautheree                            *
;*                                                               *
;********************************************************************

(p Cohesion::Functional
   {(Module ^cohesion-type nil
            ^module-name <name>) <module>}
   {(Cohesion-Answers ^module-name <name>
                      ^one-function Yes) <cohesion>}
  -->
   (modify <module> ^cohesion-type Functional)
   (remove <cohesion>))


(p Cohesion::Sequential
   {(Module ^cohesion-type nil
            ^module-name <name>) <module>}
   {(Cohesion-Answers ^module-name <name>
                      ^one-function No
                      ^activities-related Data
                      ^sequence-important Yes) <cohesion>}
  -->
   (modify <module> ^cohesion-type Sequential)
   (remove <cohesion>))


(p Cohesion::Communicational
   {(Module ^cohesion-type nil
            ^module-name <name>) <module>}
   {(Cohesion-Answers ^module-name <name>
                      ^one-function No
                      ^activities-related Data
                      ^sequence-important No) <cohesion>}
  -->
   (modify <module> ^cohesion-type Communicational)
   (remove <cohesion>))
```

```
(p Cohesion::Procedural
    {(Module ^cohesion-type nil
             ^module-name <name>) <module>}
    {(Cohesion-Answers ^module-name <name>
                       ^one-function No
                       ^activities-related Control
                       ^sequence-important Yes) <cohesion>}
 -->
    (modify <module> ^cohesion-type Procedural)
    (remove <cohesion>))


(p Cohesion::Temporal
    {(Module ^cohesion-type nil
             ^module-name <name>) <module>}
    {(Cohesion-Answers ^module-name <name>
                       ^one-function No
                       ^activities-related Control
                       ^sequence-important No) <cohesion>}
 -->
    (modify <module> ^cohesion-type Temporal)
    (remove <cohesion>))


(p Cohesion::Logical
    {(Module ^cohesion-type nil
             ^module-name <name>) <module>}
    {(Cohesion-Answers ^module-name <name>
                       ^one-function No
                       ^activities-related Neither
                       ^same-category Yes) <cohesion>}
 -->
    (modify <module> ^cohesion-type Logical)
    (remove <cohesion>))


(p Cohesion::Coincidental
    {(Module ^cohesion-type nil
             ^module-name <name>) <module>}
    {(Cohesion-Answers ^module-name <name>
                       ^one-function No
                       ^activities-related Neither
                       ^same-category No) <cohesion>}
 -->
    (modify <module> ^cohesion-type Coincidental)
    (remove <cohesion>))
```

Appendix E

User's Manual

for the

Knowledge Based

Software Engineering Environment

(KBSEE)

Description

The Knowledge Based Software Engineering Environment, KBSEE, is an executive system encompassing a variety of software development tools. It consists of four major subsystems: the command interpreter, the display manager, the project manager, and the tool set. The command interpreter reads user keyboard commands and executes user menu selections. The display manager maintains the multiple window display. The project manager maintains a data base of the user's projects, files, and locations. It also stores the user's prefered edit, compile, and link commands. The tool set consists of a variety of software tools for developing, documenting, and maintaining software systems.

System Requirements

Operating System - VAX/VMS version 4.2 or later.

Compiler - VAX C version 2.0 or later (required only for maintenance and addition of future enhancements).

Terminal - VT 100 series, VT 200 series, or compatable.

System Operation

Logical Name Definition

Each KBSEE must have the logical name KBSEE$SYSTEM defined as the location of the KBSEE executable program and

.COM tool execution files. For example, if the KBSEE is located in DUA1:[KBSEE], then the logical name would be defined by the DCL command:

    DEFINE KBSEE$SYSTEM DUA1:[KBSEE]

in the user's LOGIN.COM file or by the DCL command:

    DEFINE/SYSTEM KBSEE$SYSTEM DUA1:[KBSEE]

in the system wide startup file SYS$MANAGER:SYSTARTUP.COM.

System Execution

Once the logicl name KBSEE$SYSTEM has been defined, the KBSEE can be activated in several ways, depending upon the user and the system manager. Unless one of the steps described below is accomplished, the user must enter the DCL command:

    RUN KBSEE$SYSTEM:KBSEE

to activate the environment. The preferred method is to have the system manager place the DCL command:

    $KBSEE == "RUN KBSEE$SYSTEM:KBSEE"

in the system wide login file defined by the system logical name SYS$SYLOGIN. If this cannot be done, then the command should be placed in the user's LOGIN.COM file.

Once the symbol KBSEE has been defined in either the system wide login file or the user's login file, the environment can be executed by entering the command:

    KBSEE

at the VMS DCL $ prompt.

Once the environment has been activated, the copyright notice appears for approximately 3 seconds, followed by the main menu. The system tries to load the last project selected by the user. If the file KBSEE.SAVE does not exist, then the user must start or select a new project and user profile as soon as possible. This is accomplished by selecting the START PROJECT or SELECT PROJECT menu items and the USER PROFILE menu item. The environment will interact with the user in the messages window for the necessary information.

Keyboard Commands

The current menu selection is always highlighted. Pressing the select key on the VT200 editing keypad or 7 on the VT100 numeric keypad causes the current selection to be executed. The current selection is changed by pressing the arrow keys or numeric keypad keys 1, 2, 3, and 5, which simulate the left arrow, down arrow, right arrow, and up arrow keys, respectively (see attached figure). The remove key on the VT200 editing keypad or the PF3 key on the VT100 numeric keypad removes the current menu. If the current menu is the main menu, then the system exits immediately. The menus have a wrap around capability; i.e., if the user is at the bottom of a menu and presses the down arrow, the

current menu item becomes the top item on the menu. The same is true for changing the menu column of the main menu. Pressing the right arrow when the current item is in the rightmost menu column causes the topmost menu item in the leftmost column to become current.

Menu Selections

Build Program - edits, compiles, and links the current work file.

Edit - edits the current work file.

Compile - compiles the current work file.

Link - links the object code produced by compiling current work file.

Run - runs the executable image produced by the link command.

Display Errors - displays the error messages returned by any of the previous selections.

Start Project - starts a new project and makes it current.

Select Project - makes a different project in the project database current.

List Projects - lists all projects in the user's project database.

Display Project - displays all work files and locations associated with the current project.

Work File - creates a new work file or selects an existing one from the project database and makes it current.

User Profile - creates a profile of the edit, compile, and link commands for the current work file.

Introduction - displays useful information about the KBSEE.

Debugger - executes the VAX/VMS symbolic debugger.

Analyzer - analyzes the modules using the KBSMA.

Librarian - executes the VAX/VMS librarian utility.

Printer - prints a file on one of the system printers.

Text Formatter - executes a text formatting utility, MASS11 or Runoff.

Spawn to CLI - temporarily exits the KBSEE and invokes a subprocess at the Command Language Interpreter level. Entering the DCL command LOGOUT returns the user back to the KBSEE.

Exit - exits the system.

Maintenance

Building the System

The source code for the KBSEE is contained in three files. The master file is named KBSEE.C, which calls the other two files through C preprocessor commands:

    #include kbsee_proj

#include kbsee_exec

The file kbsee_proj.c contains the code for the project
manager. The file kbsee_exec contains the code for the
command interpreter and display management functions.

To modify the system, edit the appropriate source file.
Then, compile the modified system by entering the following
DCL command:

CC KBSEE

Ignore the warning about the conflicting definition of
LIB$SPAWN.

Prior to the link command, check to ensure that the two
logical names LNK$LIBRARY and LNK$LIBRARY_1 are defined as
shown below:

define/nolog lnk$library sys$library:vaxcrtl

define/nolog lnk$library_1 sys$library:vaxccurses

These logical names are essential for the proper operation
of the link command when linking KBSEE. It defines the
location of all external Run Time Library functions called
by KBSEE. Since almost all routines used by the KBSEE
modules are in one of these Run Time Libraries, failure to
specify these logical names will cause severe errors in the
link process.

Once the modified system compiles without fatal errors
and the two lnk$ logicals are defined, link the KBSEE with
the following command:

E - 7

      LINK KBSEE, OPTIONS_FILE/OPT

where options_file is a file named options_file.opt with the following line:

      sys$share:vaxcrtl.exe/share

This options file speeds the link time and execution time by making all the references to the VAX C Run Time Library shareable.

For further information about compiling and linking, see the Programming in VAX C and the VAX/VMS Linker Reference Manual.

The file generated by the link command is the executable image for the KBSEE, KBSEE.EXE. This is the file used by the DCL RUN command when the user enters KBSEE or RUN KBSEE$SYSTEM:KBSEE into the DCL CLI.

Adding a New Tool

To add a new tool to the KBSEE, edit the file KBSEE_EXEC.C and add an item to one of the menus (main_menu_1, main_menu_2, main_menu_3, main_menu_4). Then edit KBSEE.C and modify the main_data structure initialization for the appropriate menu by adding 1 to the max_y value. Add an appropriate string comparison to the chk_main_1, 2, 3 or 4, routines (depending on which menu the new item is in) and add a function call to a new exec routine that actually calls a subprocess command file and

executes the tool. If the tool writes to the screen, then the command file must contain a clear screen command before and after the tool is actually invoked.   See the exec_analyzer.com and exec_edit.com for examples of how to accomplish this. If the tool does not write to the screen, then use the exec_compile.com as an example.

Appendix F

User's Guide

for the

Knowledge Based

Software Module Analyzer

(KBSMA)

User's Guide - KBSMA

Description

The Knowledge Based Software Module Analyzer, KBSMA, is a knowledge based system which analyzes software modules using the software engineering principles of coupling and cohesion.

System Requirements

Operating System - VAX/VMS version 4.2 or later.

Compiler - VAX OPS-5 version 1.0 or later (required only for maintenance and addition of future enhancements).

Terminal - VT 100 series, VT 200 series, or compatable.

System Operation

Knowledge Base

The KBSMA consists of two files, the knowledge base, which contains the rules and the data structure definitions, and the instances, containing data about the modules to be analyzed. The rules are written in OPS-5 productions. The data structures are defined by OPS-5 LITERALIZE statements. For details on the syntax of OPS-5 productions and literalizations, see the OPS-5 User's Manual.

The knowledge base for the KBSMA is contained in the file KBMSA.OPS. This file does not have to be recompiled unless more productions or data structures definitions are added.

The instances are contained in the file KBSMA_INSTANCES.DAT, which contains all the data about the specific modules to be analyzed. This file must be edited manually to add the necessary information. The current version of the file is an excellent example of the required syntax and data items. This file is loaded into working memory whenever the KBSMA is activated. This separation of the knowledge base and instances provides a great deal of flexibility and ease of use, since the knowledge base does not change for each module, only the instances.

System Execution

The KBSMA can be activated in two different ways, manually and by selection of the Analyze option in the KBSEE. To activate the system manually, enter

RUN KBSEE$SYSTEM:KBSMA

The KBSMA contains startup information which loads the instances into working memory automatically. The STARTUP function is located in the file KBSMA.OPS. The KBSMA prompts the user for cohesion information, but automatically deduces the coupling type from the instances.

Maintenance

Building the System

The source code for the KBSMA is contained in the file

KBSMA.OPS. To build a new executable image, enter the following command:

    OPS KBSMA

The OPS-5 compiler produces an executable image file directly, so no link operation is necessary.

Adding a New Rule

To add a new rule, design the rule in an Engish like manner, then translate it into the OPS-5 language. For details on the OPS-5 syntax, see the OPS-5 User's Manual. The new rule needs to be added to the source file KBSMA.OPS. If the new rule uses a data structure that has not yet been defined, then add another LITERALIZE to the source file which defines the new structure.

Edit the instances file KBSMA_INSTANCES.DAT and add new instances using the newly defined and currently existing data structures. Then, recompile the knowledge base with the new rules and data structure definitions as described above. Test the new rule by executing the newly build executable image, KBSMA.EXE.

Bibliography

AFIT/ENG. *AFIT/ENG Development Documentation Guidelines and Standards, Draft #2.* Department of Electrical and Computer Engineering, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base, Ohio, 1984.

Aho, Alfred V. et al. *The Design and Analysis of Computer Algorithms.* Reading MA: Addison-Wesley Publishing Company, 1974.

AJPO, Ada Joint Program Office. *Requirements for a Programming Environment for the Common High Order Language,* preliminary Stoneman, Department of Defense, Washington, DC, 1980.

AJPO, Ada Joing Program Office. *Ada Programming Language Reference Manual,* ANSI/MIL-STD 1850A-1983, Department of Defense, Washington, DC, 1983.

Babb, Robert G. II et al. "Workshop on Models and languages for Software Specification and Design," *Computer, 18:* 103-108 (March 1985).

Barstow, David R. and Howard E. Shrobe. "From Interactive to Intelligent Programming Environments", *Interactive Programming Environments,* McGraw-Hill Book Company, New York, New York, 1984.

Boehm, B. W. "Software Engineering", *IEEE Transactions on Computers,* Vol C-25, 12:1226-1241 (December, 1976).

Chandrasekaran, B. "Generic Tasks in Expert System Design and Their Role in Explanation of Problem Solving," *Proceedings of the NAS/ONR Workshop on AI and Distributed Problem Solving,* May 1985.

-----. "Towards a Taxonomy of Problem Solving Types," *The AI Magazine:* 9-17 (Winter/Spring 1983).

Charniak, Eugene et al. *Artificial Intelligence Programming.* Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers, 1980.

Cohen, Paul R. and Edward A. Feigenbaum. *The Handbook of Artificial Intelligence.* Los Altos, California: William Kaufmann, 1982.

DEC. OPS5 User's Manual, AA-BH00A-TE, Digital Equipment Corporation, Maynard, Massachusetts, 1984.

-----. Programming in VAX C, AA-L270B-TE, Digital Equipment Corporation, Maynard, Massachusetts, 1985.

DeMarco, Tom. Structured Analysis and System Specification. New York: Yourdon Press, 1979.

Deutsch, Michael S. "Validating Functional Requirements using a Human Knowledge Base," Draft for Submission to IEEE Transactions on Software Engineering: August 1985.

Forgy, Charles, et al. "Initial Assessment of Architectures for Production Systems," Department of Computer Science, Carnegie Mellon University, DAPRA Order No 3597: 116-120 (1984).

Freeman, P. Tutorial on Software Design Techniques, IEEE Computer Society, (1976).

Gould, John and Clayton Lewis. "Designing for Usability: Key Principles and What Designers Think," Communications of the ACM, 28 (3): 300-311 (March 1985).

Hadfield, 2Lt Steven M. and Gary B. Lamont. "The Software Development Workbench: An Integrated Software Development Environment," Proceedings of the Digital Equipment Computer Users Society: 171-177 (1983).

Hadfield, 2Lt Steven M. An Interactive and Automated Software Development Environment. MS Thesis AFIT/GCS/EE/82D-17. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1982.

Hansen, Wilfred J. "User Engineering Principles for Interactive System", Fall Joint Computer Conference Proceedings, (39): 523-532 (1971).

Harmon, Paul and David King. Expert Systems: Artificial Intelligence in Business, John Wiley and Sons, New York, 1985.

Hayes-Roth, Frederick et al. Building Expert Systems. Reading, Mass: Addison-Wesley Publishing Co, 1983.

Helms, Harry L. Computer Language Reference Guide, Second Edition, Howard W Sams and Co, Indianapolis, Indiana, 1984.

Horowitz, Ellis and Sartaj Sahni. _Fundamentals of Data Structures in Pascal_. Rockville MD: Computer Science Press, 1984.

Houghton, Raymond C., Jr. "Software Development Tools: A Profile," _Computer, 16_ (5): 63-70 (May 1983).

IEEE. "IEEE Standard Glossary of Software Engineering Terminology," _IEEE Std 728-1983_.

Kernighan, Brian W. and Dennis M. Ritchie. _The C Programming Language_, Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

Kinnucan, Paul. "Software Tools Speed Expert System Development," _High Technology, 5_ (3): 16-21 (March 1985).

Kowalsky, Robert. "AI and Software Engineering," _Datamation_: 92-102 (November 1, 1984).

Linden, Eugene. "IntelliCorp: The Selling of Artificial Intelligence," _High Technology, 5_ (3): 22-25 (March 1985).

MacLennan, Bruce. _Principles of Programming Languages_, Holt, Reinhart, and Winston, New York, 1983.

Manuel, Tom. "Cautiously Optimistic Tome Set for 5th Generation," _Electronics Week, 57_ (34): 57-63 (December 3, 1984).

Manuel, Tom and Michael Rand. "Has AI's Time Come At Last?", _Electronics Week, 58_ (5): 51-62 (February 4, 1985).

Martin, James. _System Design from Provably Correct Constructs_. Englewood Cliffs, New Jersey: Prentice-Hall, 1985.

Mihaloew, Reed A. _SYSFL, A Systems Flowcharting Routine Using Interactive Graphics_. Aeronautical Systems Division Computer Center, Air Force Systems Command, Wright-Patterson AFB OH, undated.

Myers, Glenford J. _Reliable Software Through Composite Design_. New York: Von Nostrand Reinhold Company, 1975.

Myers, Ware. "The Need for Software Engineering," Computer, 11 (2): 12-25 (February 1978).

Nilsson, Nils J. Principles of Artificial Intelligence. Palo Alto, California: Tioga Publishing Co, 1980.

Partach, H. and R. Steinbruggen. "Program Transformation Systems," Computing Surveys, 15 (3): 199-236, 1983.

Peters, Lawrence J. Software Design: Methods and Techniques. New York: Yourdon Press, 1981.

Ramanathan, Jayashree. "Softer Ware," News in Engineering, Ohio State University 56 (2): 15 (March 1984).

Rich, Elaine. Artificial Intelligence. New York: McGraw-Hill Book Company, 1983.

Rychener, M. D. "Expert Systems for Engineering Design: Problems, Components, Techniques, and Prototypes," Carnegie-Mellon University, Report DRC-05-02-83 (26 March 1984).

Sheil, B. A. "Power Tools for Programmers," Datamation, Technical Publishing Co, 1983.

Swartout, William R. "XPLAIN: A System for Creating and Explaining Expert Consulting Programs," Artificial Intelligence Journal, 21 (3): 285-325 (September 1983).

Teitelbaum, Tim and Thomas Reps. "The Cornell Program Synthesizer: A Syntax-Directed Programming Environment," Communications of the ACM, 24 (9): 563-573 (September 1981).

Teitelman, Warren. "A Display Oriented Programmer's Assistant", CSL 77-3, XEROX PARC, Palo Alto, California, 1977.

UM 170133010. Interim AUTOIDEF System User's Reference Manual. Materials Laboratory, Air Force Wright Aeronautical Laboratories, Air Force Systems Command, Wright-Patterson AFB OH, 1982.

Waters, Richard C. "The Programmer's Apprentice: Knowledge Based Program Editing," IEEE Transactions on Software Engineering, SE-8 (1): (January 1982).

Wess, Bernard P., Jr. "Artificial Intelligence Techniques Speed Software Development," <u>Mini-Micro Systems</u>: 127-136 (September 1984).

Wirth, Niklaus. <u>Algorithms + Data Structures = Programs</u>. Englewood Cliffs NJ: Prentice-Hall, Inc., 1976.

Woffinden, D.S., Instructor of Electrical and Computer Engineering. Lecture materials in EENG 793, Advanced Software Engineering. School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1985.
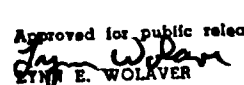
Vita

VITA-1

VITA

Captain David W. Fautheree was born on 8 July 1958 in Stuttgart, West Germany. He graduated from a U.S. Department of Defense Overseas Dependants high school in Naples, Italy in June 1976. He then attended Lousiana Tech University from which he received the degree of Bachelor of Science in Mathematics in May 1980. Upon graduation, he received a commission in the United States Air Force through the Air Force Reserve Officer Training Corps as a Distinguished Military Graduate. After completing the Computer Systems Development Officer course at Keesler AFB, Mississippi, his first assignment was to the Air Force Contract Management Division, Kirtland AFB, New Mexico, where he was the Chief, Computer Systems Planning Branch in the Computer Resources Management and Communications Office. During this time, Capt Fautheree designed, implemented, and managed a nationwide computer network. He entered the School of Engineering, Air Force Institute of Technology, in May 1984.

Permanent address: 14103 Oakstead

San Antonio, TX 78231

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION	1b. RESTRICTIVE MARKINGS
Unclassified	AI72407

2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT
	Approved for Public Release;
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	distribution unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
AFIT/GCS/ENG/86M-2	

6a. NAME OF PERFORMING ORGANIZATION	6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
School of Engineering	AFIT/ENG	

6c. ADDRESS (City, State and ZIP Code)	7b. ADDRESS (City, State and ZIP Code)
Air Force Institute of Technology Wright-Patterson AFB OH 45433	

8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
AFIT	ENG	

8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS.			
Air Force Institute of Technology Wright-Patterson AFB OH 45433	PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.

11. TITLE (Include Security Classification)
See Box 19

12. PERSONAL AUTHOR(S)
Fautheree, David W., B.S., Captain, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
MS Thesis	FROM ____ TO ____	1986 March 21	179

16. SUPPLEMENTARY NOTATION

17.	COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	Software Engineering Software Design
			Artificial Intelligence Software Life Cycle
			Expert Systems

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

TITLE: An Analysis Tool in a Knowledge Based Software Engineering Environment

THESIS CHAIRMAN: Gary B. Lamont, Ph.D.

Approved for public release IAW AFR 190-1
LYNN E. WOLAVER   9 May 86
Dean for Research and Professional Development
Air Force Institute of Technology (ASO)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
Gary B. Lamont, Ph.D.		AFIT/ENG

DD FORM 1473, 83 APR   EDITION OF 1 JAN 73 IS OBSOLETE.   UNCLASSIFIED

ABSTRACT: This thesis investigation presents the conceptual level
             development of a Knowledge Based software engineering
             environment.   A variety of existing tools are integrated
             into the environment as well as newly developed
             knowledge based tools, such as the software module analysis
             tool designed and implemented for this project.

             System development follows the software engineering
             lifecycle of requirements analysis, design, implementation,
             and operation as well as exploratory programming/rapid
             prototyping techniques.

END

11-86

DTIC