

A Final Report
for Year One of the Task
**Methodology for Automating
Software Systems**

Task I of the Foundations for
Automating Software Systems.

Prepared for Tim Crumbley
Nasa Marshall Space Flight Center
Redstone Arsenal, Alabama

N2813

by

Dr. Warren Moseley
University of Alabama in Huntsville
Research Institute Room M34C
Huntsville Alabama, 35899

Report Number 814

I. General Introduction and Background for The Poor Man's Case Tool(PMCT).

Research Platform CASE Environment.

The University of Alabama in Huntsville is in the early stages of a five-year intensive research program to establish a experimental research platform for software engineering. There will be a major emphasis placed on CASE and the importance of CASE to the improvement of the practice of software engineering. This project is a first step in establishing this research program, and first in a part of this three year effort here at NASA.

Outline of major functions of our case tool

The operation of this system, Poor Man's CASE Tool, is based on the Apple Macintosh system, employing available software including: **Focal Point II, Hypercard, XRefText and Macproject**. These programs are functional in themselves, but through advanced linking are available for operation from within the tool under development.

The software industry is in need of maps, a plan where they want to go, how they want to get there and something to measure their progress as they journey. They need CASE tools. As hardware technology advances are reported on a daily basis, true software advances are much fewer and farther between. The technology required to dramatically increase the processing speed of a computer produces very visible and objective results, but software improvements are often subjective and very tenuous. Today the focus on software is no longer entirely aimed at getting the job done, but, due to the rising cost of maintaining and developing software, rather, to make the process of arriving at the solution more efficient. Since applicable software theory is limited to the confines of the hardware and operating systems available, and major breakthroughs are rarely imminent, the only solution to this "software crisis" is some form of software

production engineering. This methodology would allow software to be synthesized instead of "written" or even "built."

Computer Aided Software Engineering, CASE, is a tool well suited to this concept. Software development has already gone through enough phases to allow for reuse of design at the concept or even the code level. Such is the aim of the Department of Defense mandating that all new software systems be written in a standardized and certified Ada system. Thereby a new portability can be found in one of the largest software development arenas in the world. This mandate also implies some operation requirements on the hardware to be used. What has then been ordained is an ability to employ "technology transfer" across development lines. This allows for information and ideas to be reused, since, in the present economy, it is far cheaper to use something that has already been done, than it is to prepare a customized system. While the tendency used to be that a customer would require a system to do the job just as he did it on paper, or by hand, today's customers are more ready to accept something that works already and make some modification to the process to be performed, whether it involves simply using a new form or a new procedure. The task is not to get the old job done, but to produce better results more efficiently.

The state of the software world is still predominantly made up of custom built software systems, but as more modular languages, such as Pascal, C and Ada, and operating systems, such as UNIX and UNIX derivatives, come into play, generic function code segments no longer need to be rewritten. The programmer need only pool his resources with those of others and find a routine that already performs the required function. Fortunately, the availability of these routines and access to them is steadily increasing through the use of Local Area Networks, Bulletin Boards, software libraries and software warehouses. All of these facilities encourage sharing software and are being relied upon more and more to cut down

on development time. Although much is being done about this problem, it still remains. There are only so many concept changes than can be effected in the current software development and use arenas, since there is such a large distribution of effort and users in the field.

Standards, wisely chosen and stringently implemented, will help to set the pace, but the volatility of the computer industry itself does not lend itself to a lot of trust in committing to a specific system. Today, standards are more widespread, but changes are still rapid, and necessary. As standards approach the concept and implementation level, that is beyond specific coding routines and methodologies, real, functional, progressive systems can be implemented at many levels of service.

The term 'CASE TOOL' refers to a computer system tool which provides the capability to perform software system design, i.e.. Computer-Assisted-Software-Engineering. The approach represents the problem solving process at an extremely high level of abstraction. I feel this is important since often the software engineering process relies specifically on an outline of the complete problem domain. After all, if the engineer cannot see 'the big picture' how can he be expected to know exactly where the smaller entities and procedures should be integrated. The development of CASE tools has been fairly recently introduced into the software development community and has been met with a tremendous amount of enthusiasm. The functionality of a CASE tool is based upon the general structure of the software life-cycle and is built to allow a general specification for each phase of the software engineering process. This capability provides a flexibility that cannot be found with other software design tools. From the specification / requirements phase where data dictionaries / entity relationships are constructed, to the maintenance phase where the system design can be

restructured, the CASE environment provides a variety of tools to aid in the construction of system software.

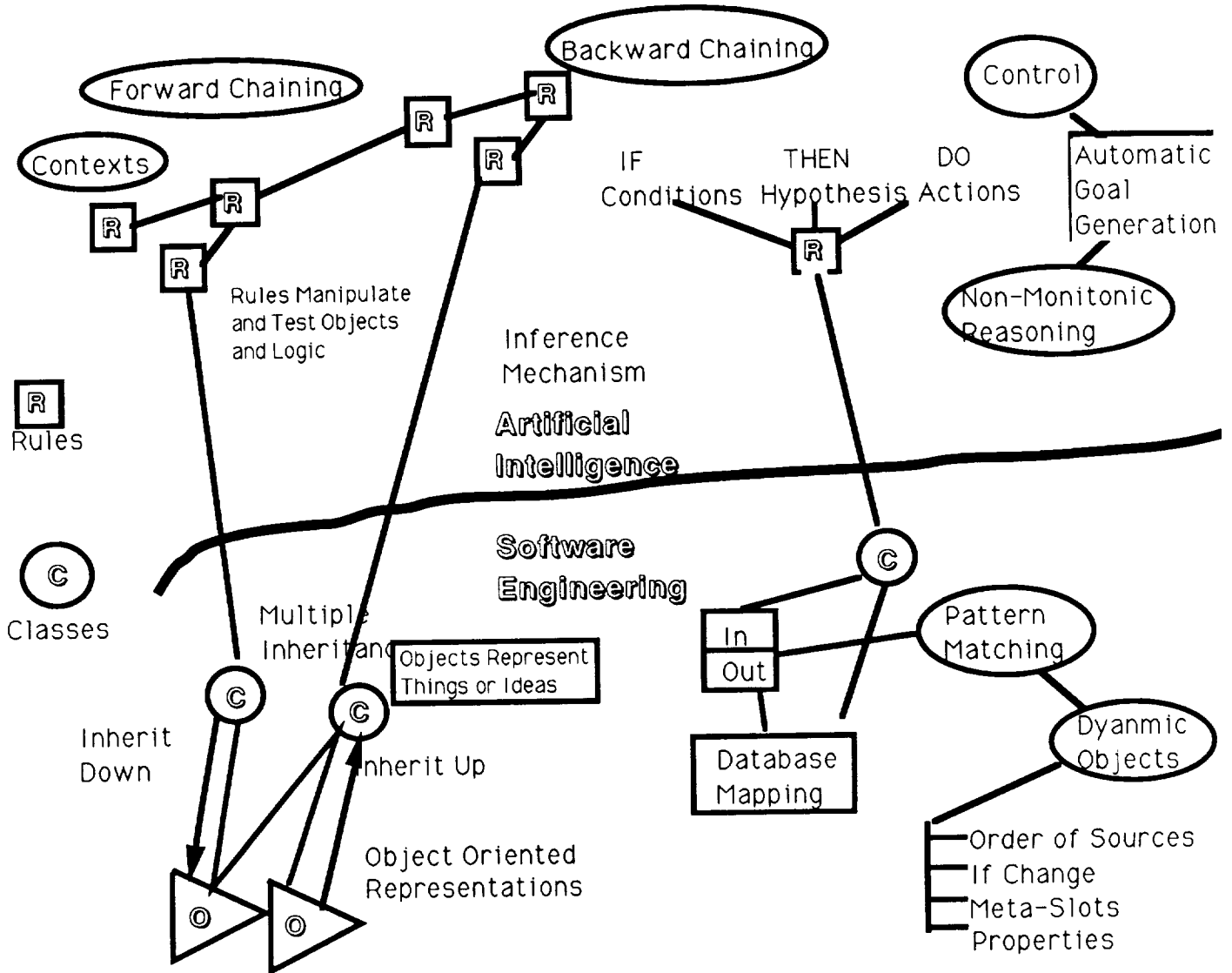
II. CASE Needs

The fact that CASE tools are in great demand is partially due to the change in software needs: programs should be efficient, easily maintained and modifiable, and are usually quite large. The size of programs especially calls for an efficient organizational tool for the software engineering process. Many times, the design process is documented on blackboards, and some even reside completely inside one software engineer's mind. This causes an extremely difficult problem for the implementation of large systems when many programmers are needed to complete the task. Also, the diverse methods used by some designers do not allow for easy access to program sections by other persons of the software development team. The use of CASE tools, however, provides a means in which the entire task (be it large or small) can be assigned to any number of designers on the team. This capability allows documentation to be constructed while the system is being constructed. Therefore, the integration of the entire system and changes to the design can both be performed easily. Not only documentation, but when one member of the team finds a need to access information from another team member, that information can be accessed immediately from the same environment. This has a psychological impact, in fact, when each member feels like 'part of the whole', the system design process will be more expediently performed.

Artificial Intelligence and Software Engineering

The following diagram illustrates the relationship of Artificial Intelligence and Software Engineering and how some of the components fit into an overall scheme of problem solving.

PMCT Overview of How AI and Software Engineering Fit

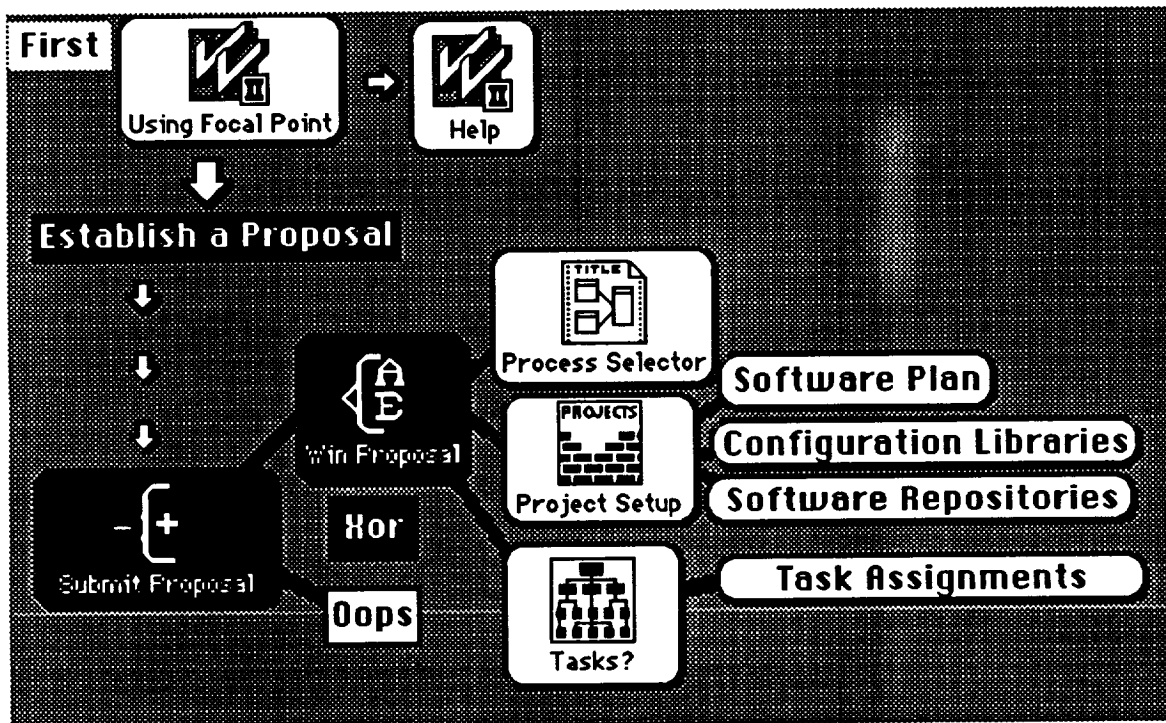


This diagram is included in this report to give an overall road map to the process involved in this section. It will be important in the future of an experimental platform to have the ability to merge the two technologies, Software Engineering and Artificial Intelligence. This is an important factor. In the first prototype of the PMCT the sections of AI are not included. It is important to understand that phase

one of the task Methodology for Automating Software Systems includes the establishment of the overall framework for the inclusion of all of the components.

Since the original prototype of MASS was done on the Macintosh, primarily in Hypercard, the ability to launch application gives NASA the ability to invoke Clips from any point in the process. Dan Rochowiak in his task, which is a part of the overall UAH task, shows a demonstration of how Clips can be used in the context of a Hypercard application. This concept will be explored in more detail in phase II and phase III.

The Sequence of Events section is the reference facility designed to start the design and then keep it on track once design is in process.



The original proposal is entered into this section and it is modified only as the customer deems necessary, beyond this, it is only used as a reference tool to keep the project in line with the customer demands. The Tool also includes work planning tools, including facilities to generate PERT charts and COCOMO cost

estimating models. These tools are industry standard job tracking references, allowing new users with project control experience to use the system without having to learn new methods of project tracking.

Another portion of the tool maintains the project database. Functions of this portion include specification generation and analysis. These segments are generated at the inception of the project and serve as benchmarks for the design process. The Data Dictionary support environment is maintained through the use of Hypercard and facilitates quick paging through data samples and formats. This section is directly accessible through the Tool's main menu. The Screen Painter serves as a user interface to the Data Dictionary. Using Focal Point II, it allows the user to create data entry and query forms to be used against the dictionary. Implementation support is a repository of functions to control and maintain system software design. The functionality of this would also include traceability of software modifications and other configuration management functions, such as binary library maintenance. This portion also includes the DOD standard specifications for software development, and a reusability subsystem, to prevent duplicate effort on the same project.

Tool Integration is the Focal Point.

One of the main objectives of the Poor Man's Case Tool was to provide access to a variety of tools for an integrated software development environment. The idea that a system should be visually intuitive is the focus of this project. From this point the user can get to any portion of the Case tool that is needed. One of the important parts of this tool is the Project Monitoring function. It does not appear on the main card or anywhere in the utilization of the tool. One of the ideas for this project was to try to derive from simple work pattern study techniques the sequence of events and underlying patterns that are an integral part of the software engineering process. The project monitoring function is totally transparent to the

engineer that is using the tool. This places no burden on the engineer in the data collection process. From studying the patterns, further research will be conducted to help derive the conceptual structures that are a part of the thought process that goes into the design of software products.

There are several major functions contained in our CASE tool, built for NASA, they include capabilities to perform the following tasks:

Planning and Monitoring

Statement of Work

Status Reports

Pert Charts

Cost Estimation

Project Data Base Construction Tools

Flexible Drawing Tool

Data Dictionary Construction

Implementation Interface

Compiler Support

Configuration Management Support

Software Quality Assurance Support

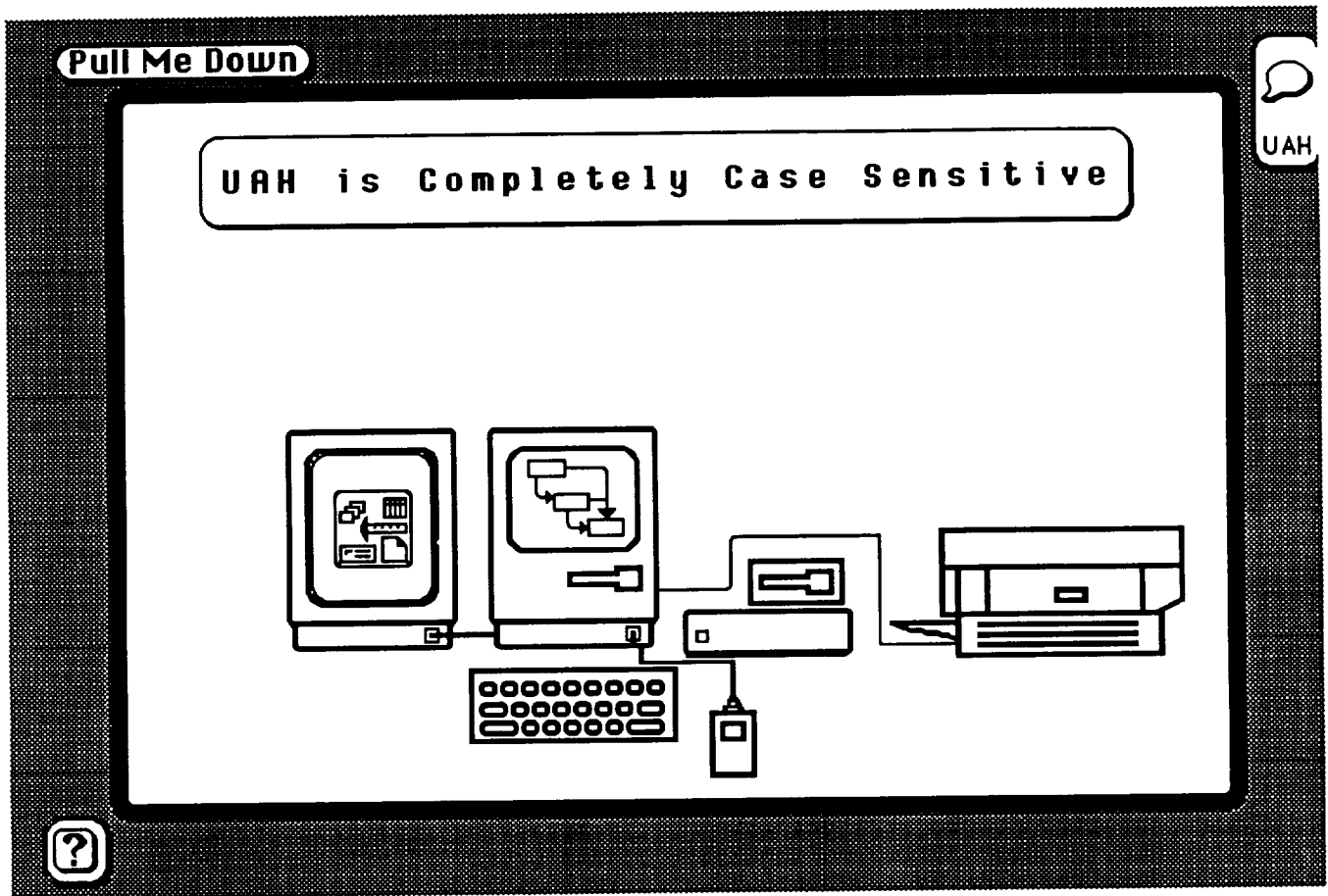
Independent Verification and Validation

Complete Traceability

These components are the major tools used by software engineers for a complete software design. They work together as illustrated. In order to visit any section of the tool just point the mouse at the appropriate item and click the mouse to select that item. The following screen is the main focal point of the entire Poor Man's Case Tool (PMCT).

Important Change since Mid-Year Report.

There have been many additions since the generation of the first Mid-Year Report. The first is the main screen and the approach for the software engineer, and the approach to the entire software process.



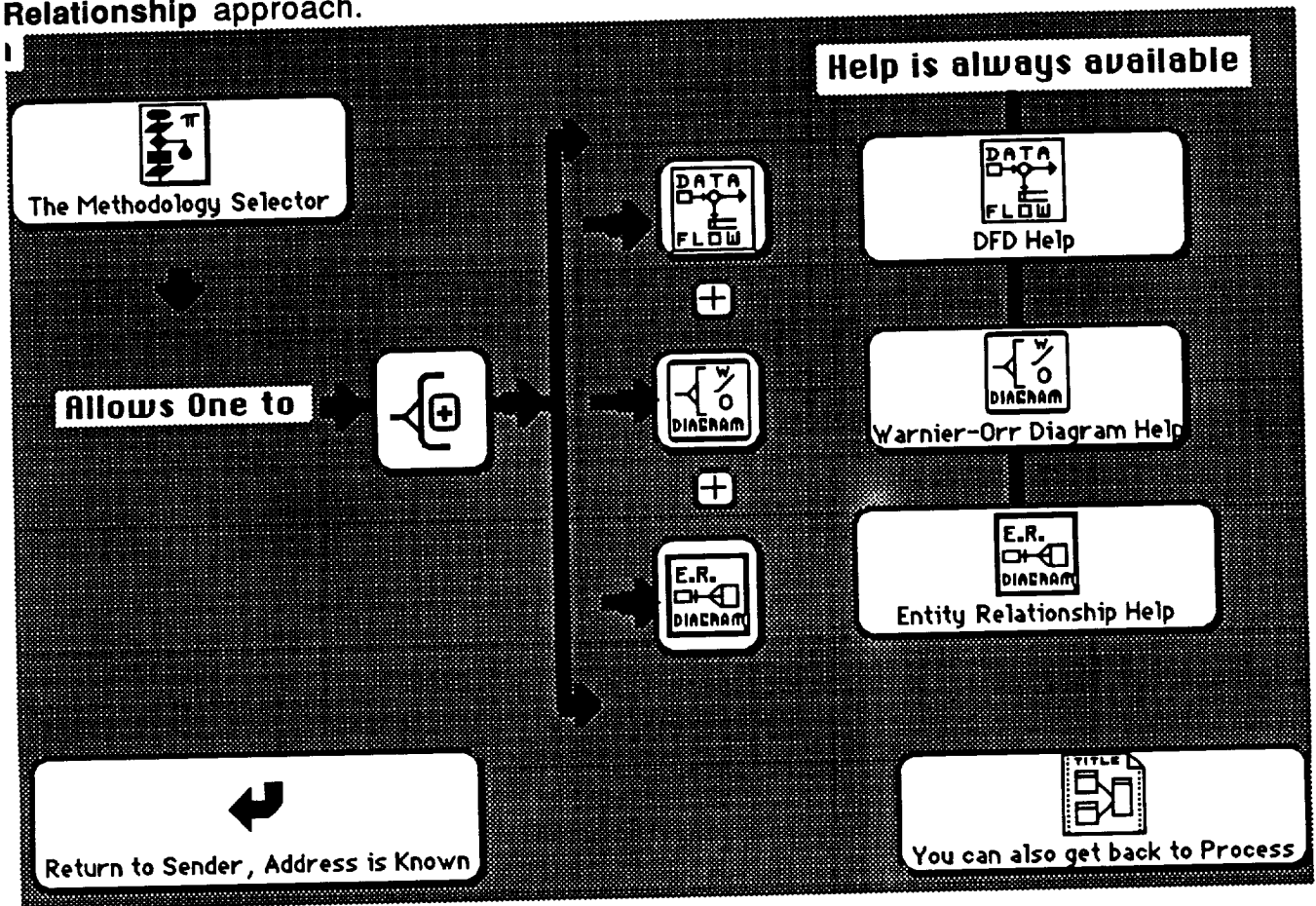
With the pull down menu in the above screen, the user has options not before available. This Menu looks like the following.

Pull Me Down

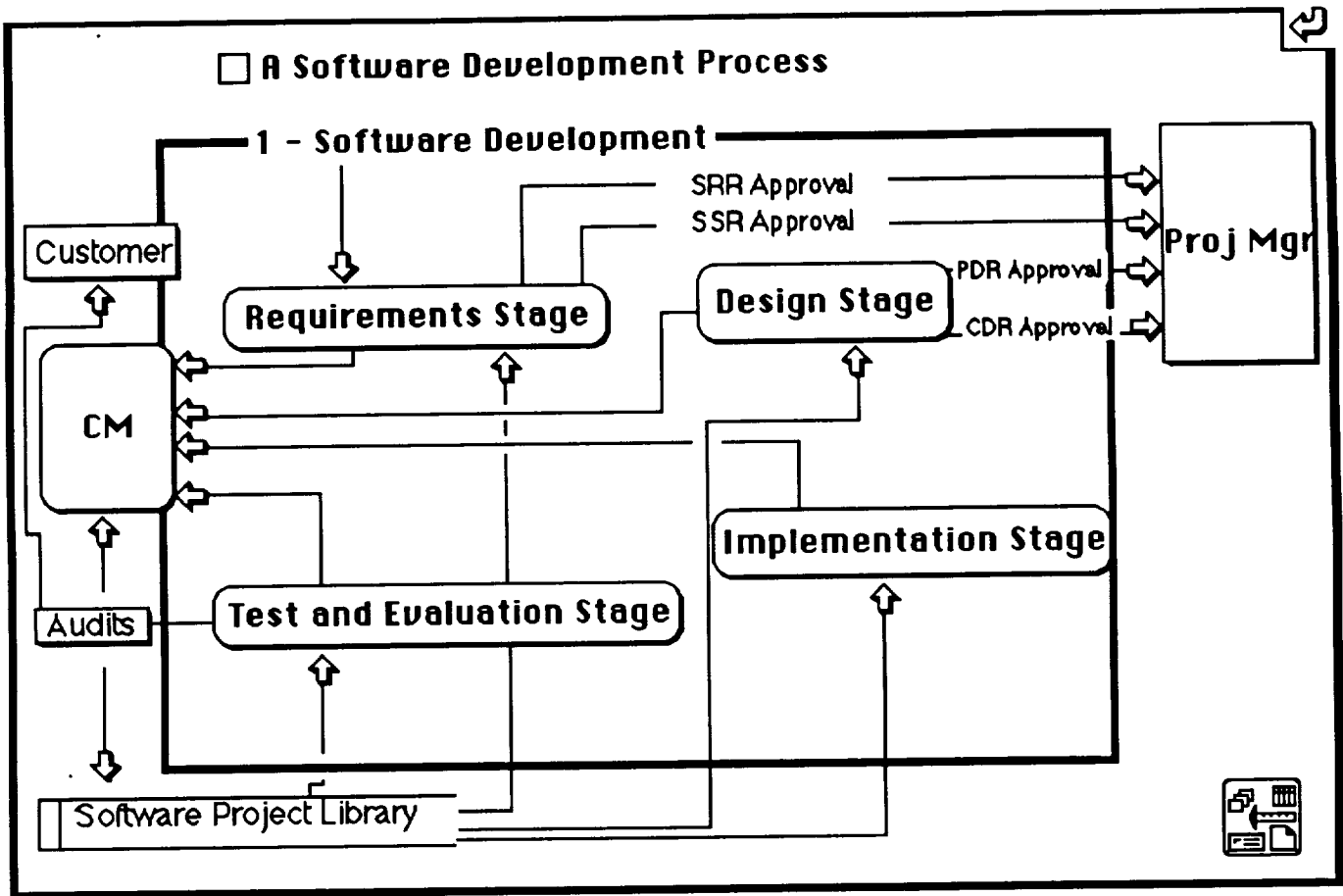
**Process
Methods**

**Plan
Repository
Help**


Assume the user chooses **Methods** as the selection the following screen would appear. A major change to the PMCT is that now the tool provides methodology specific help and drawing tool aid. The current methods supported at this time are **Data Flow**, **Data Structured(Warnier-Orr)**, and **Entity Relationship** approach.



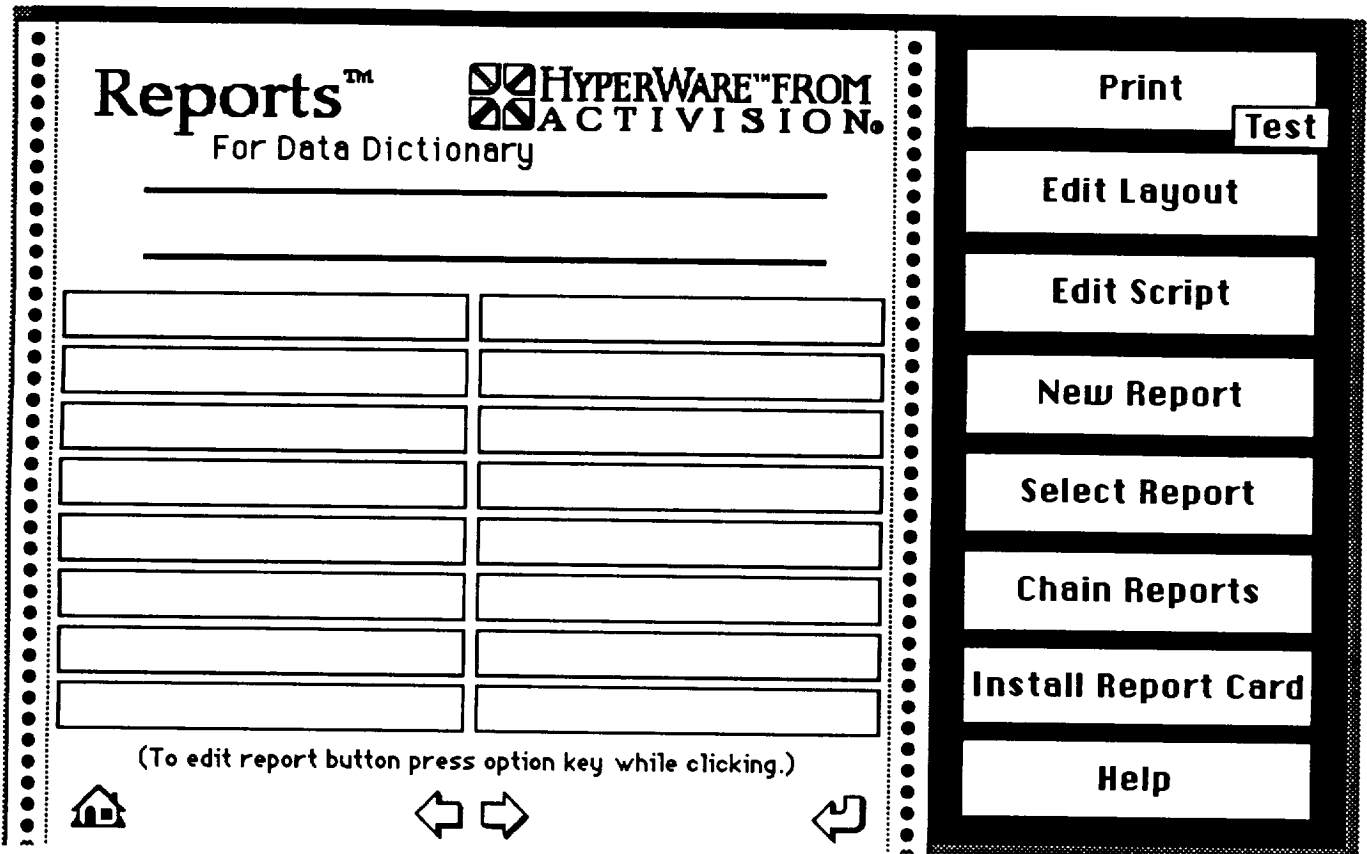
In addition there is also the addition of process. A CASE tool will not provide the desired result unless it is based on an adequate software process model. The Current models available are IEEE, 2167a, and the NASA Management Standard.



One of the salient features that has been added to the PMCT since the mid-year is the inclusion of a simple report generator. It was decided to use the package REPORTS from Activision for the prototype, since it works easily with the existing hypercard framework. The following is the new data dictionary setup in the PMCT. It is this structure that allows for expanded control of the data dictionary. When one is satisfied that the data dictionary is correct, or one wants to examine the data dictionary, the person can

Name of Field	Ballon Address		↩
Group Name	Weather Ballon		☞ ☜
Actual Description			↑
Acceptance Status			↓
Access Authority			↑
Definition Responsibility			↓
Validity and Edit Rules			
 Report Card <input type="button" value="New DD Entry"/>			

select a report card from the reporting section of the data dictionary stack. The following is the standard reporting card. For further features see the Reports Reference Manual.



Drawing Tool

Description

The drawing tool assists in the creation of Data Flow Diagrams. The tool provides buttons that generate fields to represent External and Storage nodes, creates a button to represent a Process node and generates a field to contain the Data Link. These objects allow the data flow diagram to be searched by button links and text searches. A set of Macpaint tools is included to add additional text and graphics to the diagrams. The objects can be modified or deleted at any time. It should be noted that the user can select the option of using the embedded drawing tool or they may chose to import into this drawing tool their favorite format.

Importance

The drawing tool provides a map into the design of the project. This map is a high level plan toward achieving the project goal, describing the flow of data and

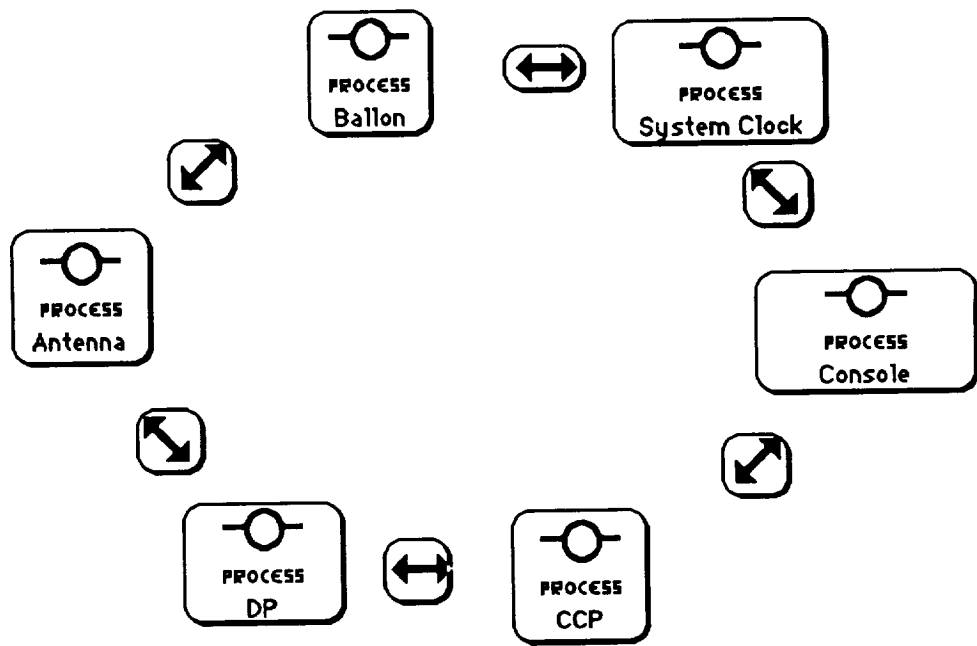
the relationship among components. The drawing tool is the map that guides the creation of the other maps within the CASE tool.

Sample Example

The following will be a sample of the types of screens that will be available from the drawing tool. The first is the main card. The entire drawing tool is completely flexible to support many types of applications. It will make available through selective buttons the ability to create and modify standard shapes, such as oval, square etc. The user can just point the mouse at the buttons and click to create that shape.

Graph Designer Main Diagram of DFD Diagrams

Help

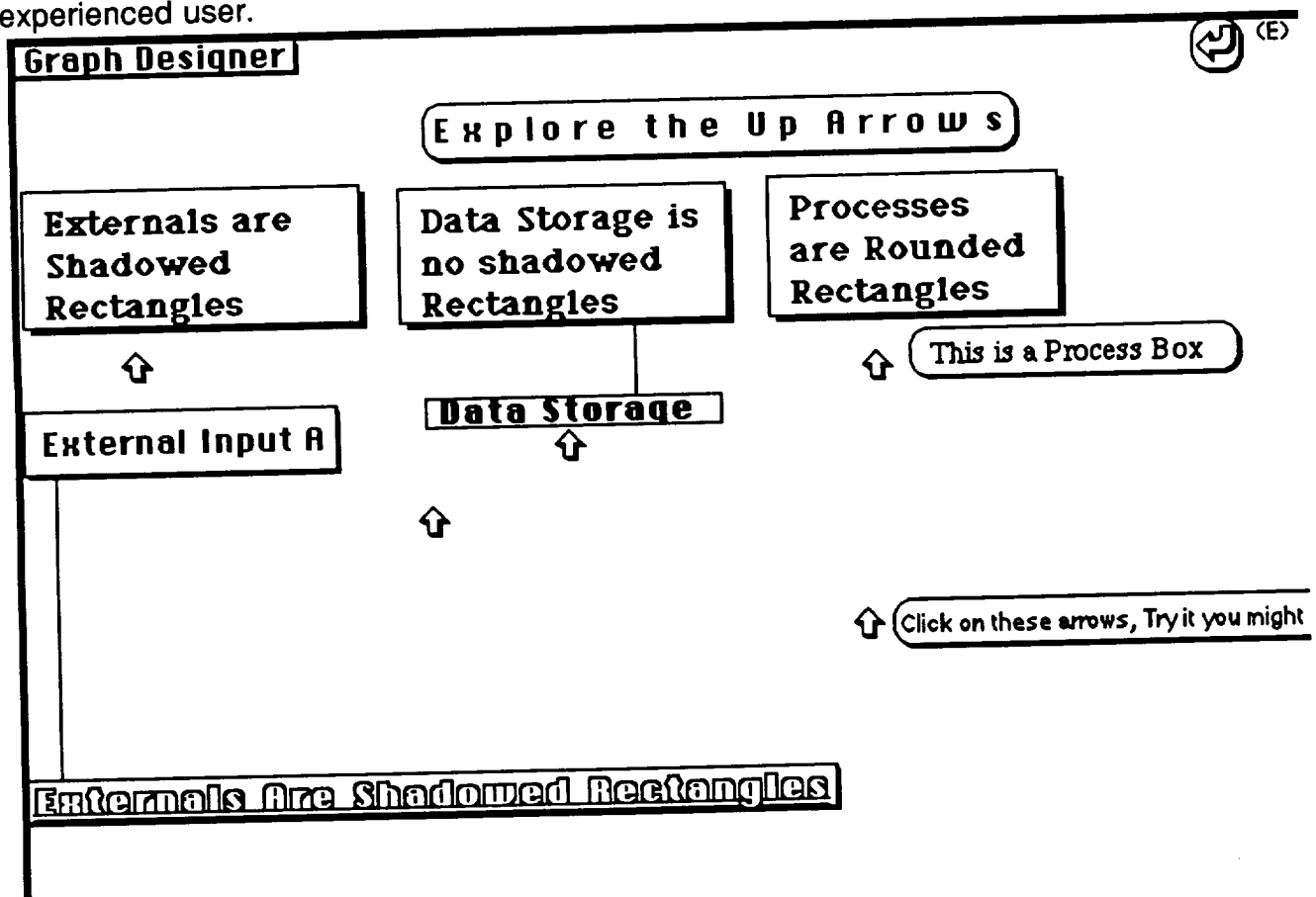


Notice the icon in the bottom of the screen, this is a sticky note. When the user clicks on this it will pop open a note window for the user to collect and record any thoughts on the subject at hand. The user can then easily tuck these away into the existing diagram, by closing the window.

The Report Generator is also active in this section of the PMCT. There are numerous reports that one can select from the PMCT.

Job Aid

All of the accessible functions in the Poor Man's Case Tool have their own job aids. The idea was to make the initial training time minimal, and to provide a level of context sensitive help facilities available to the novice user as well as the experienced user.



Data Dictionary Support Environment

Data Dictionary Description

In PMCT the data dictionary is directly linked to the drawing and vice versa. It is important and mandatory to establish this link between graph and dictionary. The data dictionary is a storehouse of information for data fields. The data dictionary is responsible for describing many aspects of the total system. The data dictionary includes many cross references between data elements and program modules. The cross references also include modules, the reports they generate, and the variables used in them. The information that the dictionary contains is broken down into three parts.

The sections of the PMCT

1. Start up procedures

- a. Who is doing the project
- b. Scope of Project
- c. Project Start
- d. Scheduled Completion
- e. Why is the project being done
- f. Who says what the project will encompass
- g. Who is going to pay for the project

2. Personnel

- a. Group Name
- b. The Database Administrator
- c. Individual member of the group
- d. The individuals job title
- e. The company that the individual works for
- f. The address of the company
- g. Phone numbers for contacting the individual

3. Variables and Data

- a. Project ID
- b. Group Name doing the project.
- c. Variable name
- d. Variable length and range
- e. Variable type
- f. Modules accessed
- g. Reports generated
- h. Module Creation and Update

Importance:

The importance of the data dictionary is to limit the confusion that occurs during major programming tasks. The standards are set up in the data dictionary that the project will use. The data dictionary allows everyone access to the standard format of the data and the standard usage of the data. In this respect the data dictionary directly supports software development and maintenance.

Sample Example

The following are sample examples from the data dictionary section of the PMCT(Poor Man's Case Tool). The first screen is the main screen to select the type of information to go into the dictionary.

Data Dictionary



Personnel information

Start-up



Data information

JOB AID



Drawing Tool Link



The next information is the personnel assigned to the project and the information pertinent to each of those personnel. Notice that the Job Aids are always accessible from any place.

Group Name
DBA
Name
Job Title
Company
Street
City

State **Work Phone**
Zip Code **Home Phone**

The report generator can generate reports from the data dictionary.

Reports™

HYPERWARE™ FROM
 ACTIVISION®

For Data Flow Diagrammer

Unpaid Invoices

(To edit report button press option key while clicking.)

This customizing process is quite simple and flexible in the framework of the hypercard environment.

Name of Field	<input type="text"/>	Sort by field Name
Alias	<input type="text"/>	Function
Actual Description	<input type="text"/>	<input type="text"/>
Acceptance Status	<input type="text"/>	<input type="text"/>
Definition Responsibility	<input type="text"/>	
Access Authority	<input type="text"/>	
Validity and Edit Rules	<input type="text"/>	
Consistency Checking	<input type="text"/>	
Reasonableness Checks	<input type="text"/>	
Usage Propagation	<input type="text"/>	
Validation Propagation	<input type="text"/>	
Bill Gates	<input type="text"/>	

New DD Entry

Job Aid

It is an important feature to remember that all screens contain a link to the job aids for that section of the project. It is also important to indicate that in the job aids themselves there is the ability for the end user to have on-line note-taking capability, so they can enhance the job aids.

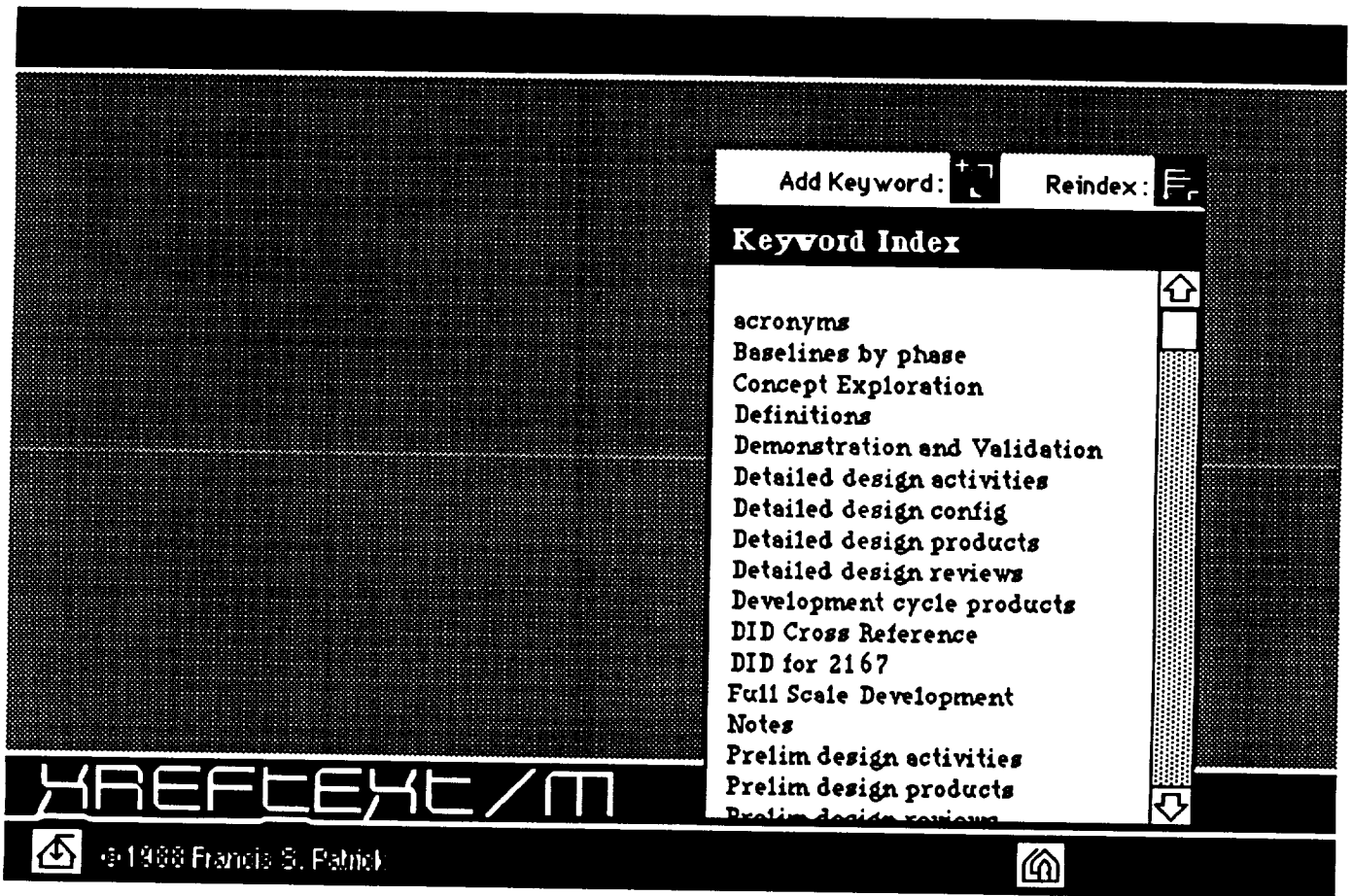
Screen Painter

Linked directly to the data dictionary and contained in the Reports section of the PMCT is the ability to select fields into a temporary schema and produce a painted screen or mock report. This utilizes a tool called Reports. Reports is a simple easy report generator available inside hypercard.

Implementation Support Standards

The standard supported by PMCT are NASA Management Standards for Space Station, DoD STD 2167 and associated companion document DoD STD 287.

This will be a system and format designed to maintain coordination of development of project components. A statement of this standard follows::



The above is a sample of the the main screen in the on line version of 2167. Currently the process is in place to move to 2167a and 287.

Importance

The importance of this aspect of the Tool following this standard is that it will allow for more structured and traceable code development. Since this standard will become the model for all new code development for military projects, government contracting corporations will be required to present their development work in this format. This standardization for such a large industry will lead to a more evenly distributed development process for all software development undertakings.

Sample Example

DID Cross Reference

acronyms
Definitions

Data requirements list and cross reference. When this standard is used in an acquisition which incorporates a DD Form 1423, Contract Data Requirements List (CDRL), the data requirements identified below shall be developed as specified by an approved Data Item Description (DD Form 1664) and delivered in accordance with the approved CDRL incorporated into the contract. When the provisions of the DOD FAR Supplement 27.410-6 are invoked and the DD Form 1423 is not used, the data specified below shall be delivered by the contractor in accordance with the contract or purchase order requirements. Deliverable data required by this standard is cited in the following subparagraphs.

Paragraph No.	Data Requirements Title	Applicable DID No.
5.1, 5.1.1.5,	System/Segment	>DI-CMAN-80008 <
5.8.1.2.3, 20.4.1,	Specification	
20.4.2, 20.4.5.2,		
30.3.1, 30.3.1.1,		
30.3.1.3, 40.6.2.2		
4.3, 4.4, 4.6, 4.7,	Software Development	DI-MCCR-80030
4.8, 5.1, 5.1.1.1,	Plan	
5.1.1.2, 5.1.1.7,		

The next section of the implementation part of PMCT is the Compiler Support interface. This would allow the users to select the compiler of their choice.

Sample Example

The following is a sample example of the compiler interface card.



Compilers

File Maker	File Maker Job Aid
Light Speed C	Light Speed C Job Aid
Light Speed Pascal	Light Speed Pascal Job Aid
Ada	Ada Job Aid

Job Aid

The following is one of the job aids that is available in the PMCT. It is the job aid for Lightspeed Pascal.

LIGHTSPEED PASCAL JOB AID

WELCOME TO THE LIGHTSPEED PASCAL JOB AID.

INTRODUCTION ***

THE PURPOSE OF THIS JOB AID WILL BE TO GIVE A GENERAL INTRODUCTION TO THE USE OF LIGHTSPEED PASCAL IN GENERATING AND RUNNING A SIMPLE PROGRAM THROUGH THE COMPILER.

THE OBJECT OF THIS JOB AID IS NOT TO BE AN INSTRUCTION MANUAL, BUT RATHER A GUIDE THAT MAY BE USED IN SYSTEM CONFIGURATION BUILDING.

Reusability Component Library Interface

Description and Importance.

Reusability is not a new concept in the world of software engineering. Software engineering has reached a "software crisis", (an overwhelming increase in the demand for software that is reliable, efficient, maintainable, understandable, delivered on time and at "reasonable" costs), that has brought reusability into the spotlight. This "software crisis" has made the reusability of software an issue that must be reconciled. Reuse is the use of previously acquired concepts and objects in a new situation.

Reusability is a measure of the ease with which one can use those previous concepts and objects in the new situation. "With both software production costs and the amount of new software produced escalating annually, the application of reusability to software development offers the potential for vast improvements in programmer productivity which will be a key to solving the "software crisis". If current trends continue, in the near future many companies that have not adopted software reuse as a standard will find themselves in a situation where they can no longer be competitive in the contracts arena. Boehm¹ has stated that, "the demand for new software is increasing faster than our ability to supply it, using traditional approaches."

With systems today being too large for a single individual to comprehend and increased pressure to keep development costs less than system complexity growth, and the cost of software being an exponential function of its size, there is the creation of a no win situation for software developers using the traditional approaches referred to by Boehm. This, and the serious shortage of qualified

1

programmers to meet these demands, will become a driving force behind reusability.

The Software Engineering Repository Retrieval System (SERRS) provides a mechanism for the cataloging and retrieval of various repository components. This system implements the design methodology of hypertext. SERRS is implemented in XREFTEXT running on top of HYPERCARD as defined previously.

SERRS was created with the idea of allowing the maximum amount of flexibility for traversal while maintaining a structured traversal method, and minimizing maintenance complexities. Flexibility was built into the basic structure to allow repositories to be added with minimal structure changes. The basic structure is a modified hierarchical structure stored as a key word index stack. Each element on the stack is known as a card. From the top of the structure, forward paths available lead to more detailed levels of the repository, one level at a time. The backward paths of the structure are less hierarchical in that all levels have paths that lead back to the previous level, but there are also paths that return to the top levels. As the user becomes more familiar with the stack, a selection can be made from the key word index that allows the immediate selection of the desired card instead of traversing SERRS from the top. An asterisk, (*), following a word indicates that this is a key word for a card containing information specific to the key word. Selection of this word will provide a forward or backward path to another level.

SERRS HIERARCHY

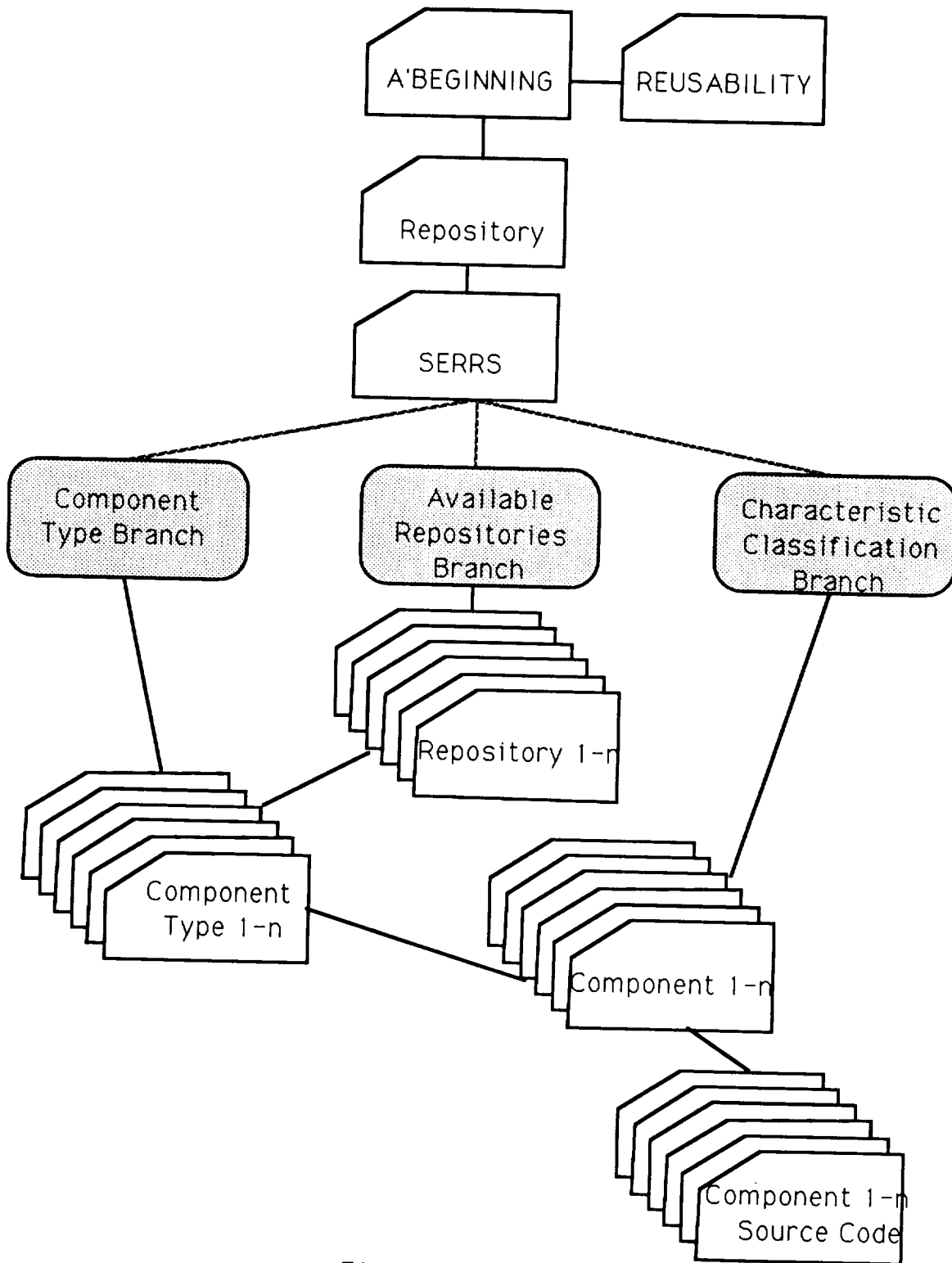


Figure 3

The first four cards, as shown in Figure 3, contain both introductory information concerning software reusability and an informal table of contents. The title page and credits for SERRS is contained on the A'BEGINNING card. The A'BEGINNING card is the highest level of the hierarchy. From the A'BEGINNING card the user can exit SERRS or select a forward path to the Reusability card for an explanation of the principles of reusability, or the Repository card to traverse SERRS. The Reusability card is only a definition card and exists as a side card from A'BEGINNING. Because this card is not an intricate part of SERRS, the user must use the direction keys provided on the top line of the screen to return to A'BEGINNING. The Repository card details the various repositories and provides a brief description of the structure of SERRS, with a backward path to the A'BEGINNING card and a forward path to the SERRS card. The Repository card is the only means to return to the A'BEGINNING card. The SERRS card provides a detailed listing of the high level break down of SERRS, including access to the general copyright and disclaimer information. The SERRS card can be read as an informal table of contents. The hierarchy breaks down into three branches from the SERRS card that will be discussed separately. These branches are: The Component Type, the Available Repositories, and the Classification Characteristics, (refer to Figure 3).

The Component Type branch allows the selection of the following component type cards: Artificial Intelligence (AI), Benchmarks, Common APSE Interface Set (CAIS), Communications, Reusable Software Components, Database Management, Documentation, Graphics, Project Management, Ada Software Development Tools (ASDT), and Other Tools, see Figure 4. Each component type card provides a listing

of all components specific to a selected component type. From each component type card a backward path is provided to the SERRS card, the Repository card, and to a specific repository card. A forward path to all component cards applicable to the component type is accessible from the component type cards (refer to Figure 3). The traversal of the component cards is detailed in the final paragraphs.

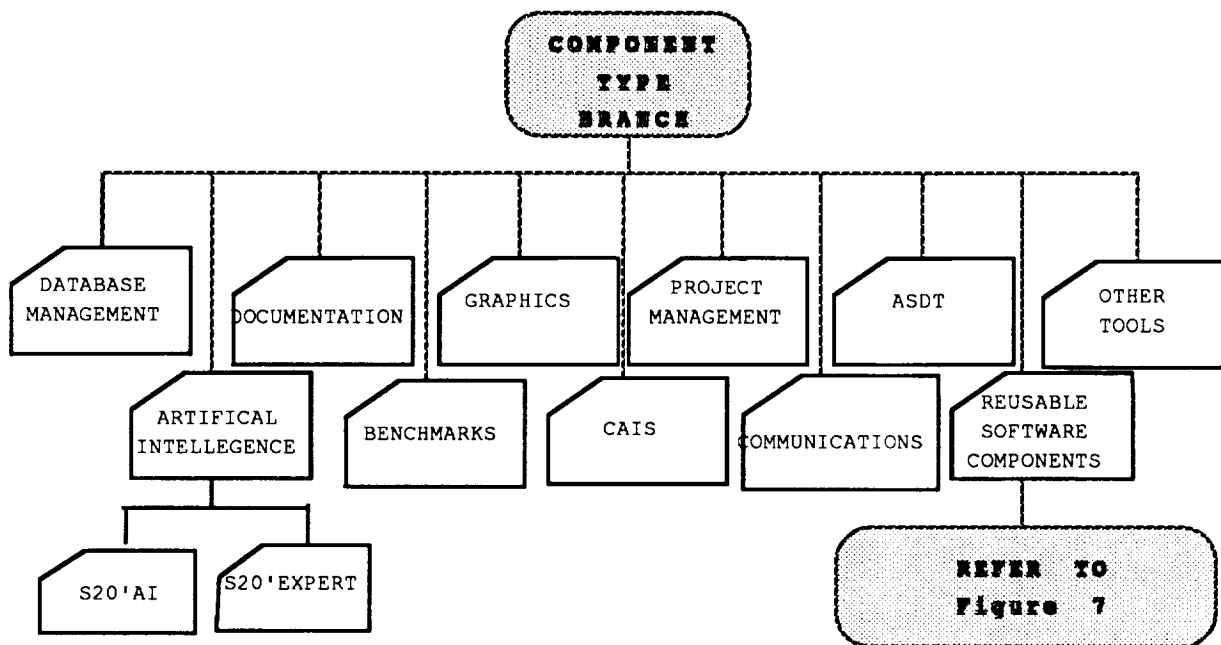


Figure 4

The Available Repositories branch provides access to the various software Repository cards that are available within SERRS, (refer to Figure 5). The repository high-level qualifiers are shown in parentheses corresponding to the appropriate repository. The repositories include: BMA Math (BMA), BOOCH (BOO), CAMP (CMP), GRACE (EVB), QTC Math (QTC), and Simtel20 (S20). From a specific repository card, there is a backward path to the SERRS card. The selection of the

Component Type cards provide the forward path within the specific Repository cards. These Component Type cards are the same as the Component Type cards selected in the Component Type branch, (refer to Figure 3). Only the Component Type cards specific to a particular Repository card are accessible from that Repository card. From the selected Component Type cards, a backward path can be selected to the SERRS card, the Repository card, and to a specific repository card. A forward path is provided to all Component cards under the component type, see Figure 3. Again, the traversal of the Component cards is explained in the final paragraphs.

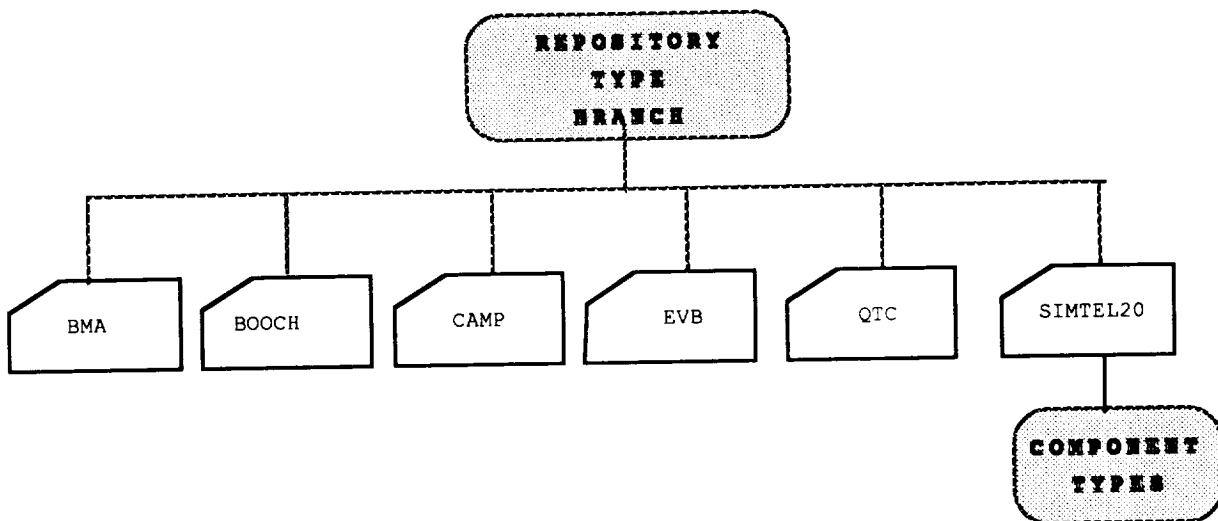


Figure 5

The Classification Characteristics branch accesses eight classification characteristics levels of the Ada language. The eight Classification Characteristics cards that can be selected are: the Generic Packages card, the Definition Packages card, the Object-Oriented Packages card, the Tasks card, the Functions card, the Procedures card, and the Programs card, (refer to Figure 6). A backward path can be selected to the SERRS card. The forward paths are the selection of the various

Component Cards meeting the selected characteristic, see Figure 3. Again, the traversal of the component cards is explained in the final paragraphs.

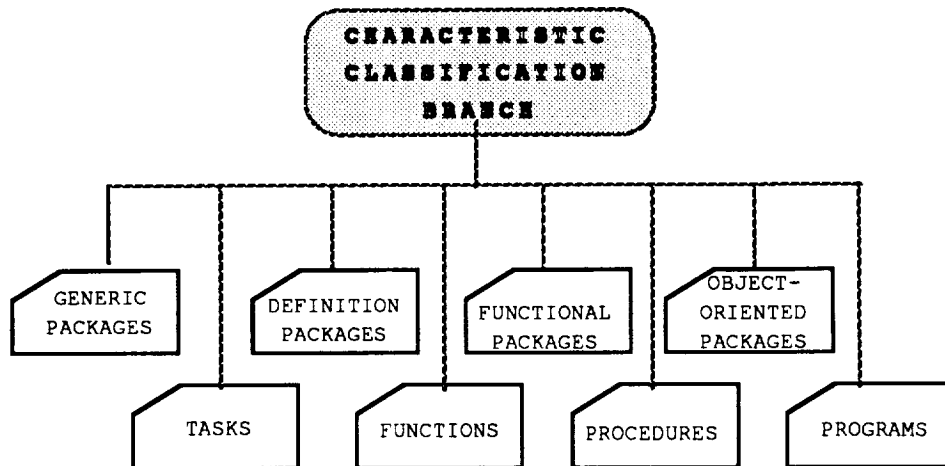


Figure 6

Regardless of the path chosen, all paths terminate at an abstraction of a particular component in the repository, excluding the documentation component type. With the exception of the Reusable Components card, selection of a Component card provides the user with a prologue explaining the operations performed by the selected component and a list of all associated files residing in the corresponding repository. The basic path structure provided for the component cards is a backward path to a specific repository card, a backward path to the Repository card, a backward path to a specific classification characteristics card, and/or a backward path to a specific Component Type card as applicable. Due to storage constraints, only selected Component cards have forward paths to the applicable Component Source Code card, which makes up the bottom level of the hierarchy,

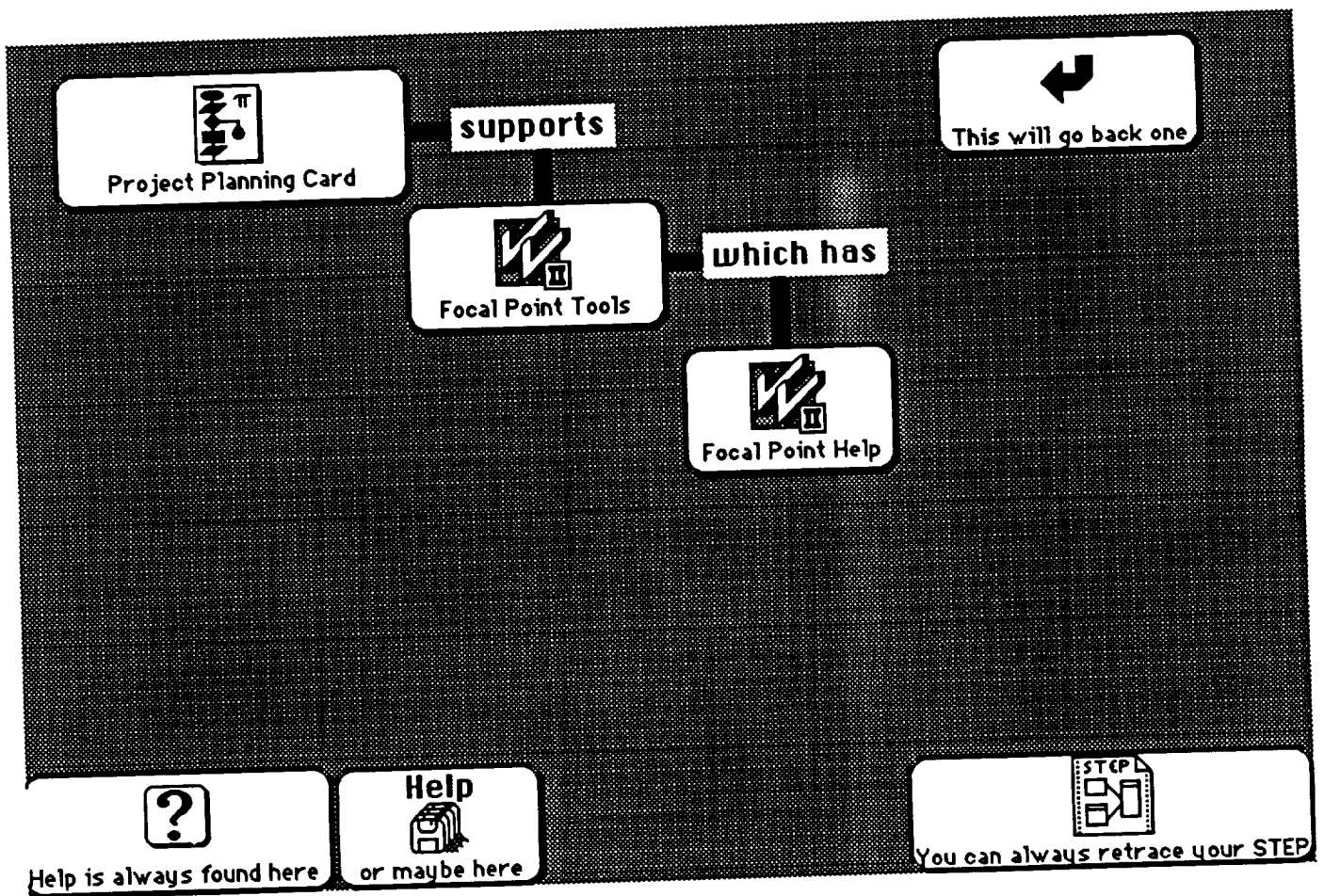
(refer to Figure 3). From the prologue provided on the selected Component Card, the Component Source Code card can be selected from the associated files list and viewed. These files are identifiable by an "A" following the repository qualifier (i.e. S20A'BIT). Only a backward path to the Component card is provided from the Component Source Code card.

In the case of the Reusable Components card, a further hierarchical break down is provided for math components and structure components. This further break down is provided because of the interest in the components falling under these categories. All other components listed on the Reusable Components card are selected and traversed in the manner described above, so the traversal method will not be re-iterated. From the Reusable Components Card, math components and structure Components can be traversed by the Math card, the Structures card, a Repository Math card, or a Repository Structures card, see Figure 7. The backward paths for the Math card and the Structures card returns to the Reusable Components card. The forward paths from the Math card are to the Math Component Type cards. The forward paths from the Structures card are to the Structures Component Type cards. The backward paths from the Math Component Type cards are to the Reusable Components card, and the Math card. The forward path is to the Math Component cards. The backward paths from the Structures Component Type cards are to the the Reusable Components card, and the Structures card. The forward path is to the Structures Component cards. The backward path from the Repository Math card and the Repository Structures card is to the Reusable Components card. The forward path from the Repository Math card is to the Math Component cards. The forward path from the Repository Structures card is to the Structures Component cards. Once either the Math Component cards or the Structures Component cards have been selected, traversal of these cards occurs exactly as traversal of the Component cards in the

previous paragraph occurred. Refer to Figure 7 for a detailed description of the forward paths within the Reusable Components card.

Planning Tools

The same planning tools are available with the addition of time management. Added also is the feature of time management with the addition of Focal Point II. Refer to the Focal Point II Manual and Job Aid for additional information on the features of this tool.



Description.

The addition of Focal Point II to the planning scenario is to allow for a more flexible monitorable planning process. Before discussion of the planning process it is important to understand the importance of monitoring of the planning process in the overall Software Engineering Scenario.

A major focus of activity will be on the importance of applying technology to the improvement of the state of practice in software engineering. The goal of this program is to develop an experimental software engineering platform that can serve as a research vehicle to carry software research into the 90's and beyond. This project is a first step in establishing this research program. There are two thrusts to this program, one is **Automation**, and the other is **Quality**.

Quality in the software process is quite tenuous. This paper suggest some criteria built in the supporting environment that rely on tried and proven methods for the increase of quality demonstrated in other engineering disciplines. This study will help insure quality in the software engineering process. Watts Humphry¹ in his book "Managing the Software Process" describes five levels of software maturity. These five levels are:

1. Initial - Until process is under statistical quality control, orderly progress in process improvement is not possible. While there are many degrees of statistical quality control, the first step is to achieve rudimentary predictability of schedules and cost.
- 2.. Repeatable - A stable process with repeatable levels of quality control, by initiating rigorous project management of commitments, cost, schedules, and changes.
3. Defined - The organization has defined the process as a basis for consistent implementation and better understanding. At this point advanced technology can be usefully introduced.
4. Managed - The organization has initiated comprehensive process measurement and analysis. This is when the most significant quality improvements begin.
5. Optimizing - The organization now has a foundation for continuing improvement and optimization of the process.

¹Humphrey, Watts, Managing The Software Development Process, Addison Wesley, 1989.

While there are other factors involved in the maturing of the software process, the primary objective is to achieve a controlled and measured process for the foundation. In the developing of the Software Assessment Procedures, the Software Engineering Institute tried to establish guidelines that would help software developers discover the level currently achieved, and to prescribe a formula for moving from one level to another.

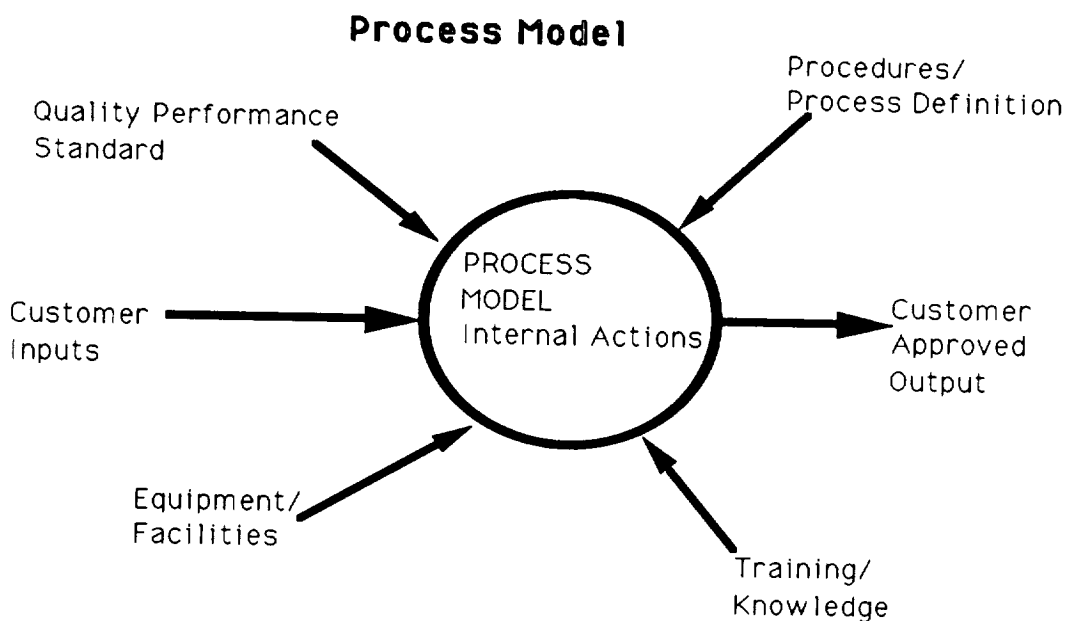
This aspect of the process explored in this paper is that of requirements traceability, and the importance of requirements traceability as an integral part of the software process. Using a more traditional approach the idea of traceability belongs in the area of documentation. It was up to the contractor to keep a traceability mapping from requirements to test item, and to demonstrate conformance to the customer. Instead of contributing to the overall accomplishment of the process this tedious task puts an increased burden on the development process. The delivery of the traceability part of the product is usually delayed until the latter steps in the process. The clerical burden placed on the project was often a deterrent to the progress of the product.

There is an increased demand for the inclusion of traceability at all levels of the software development process. Computer Assisted Software Engineering research has placed too much emphasis on trying to answer the questions concerning software life-cycle support, and not on trying to define what the software process is all about. It is of utmost importance to carefully distinguish between the idea of process and life-cycle. A process will be thought of as an ongoing activity, where a life-cycle will have specific beginning and ending tasks. The concept of traceability in a product belongs to the life-cycle aspects of the project, but the idea of traceability is a process that must transcend the individual process. It must be an integral part of the software development environment, and it must become an integral component of the idea of quality in the software engineering process.

If you would ask a fellow worker, student, or professor the question, "Are you for quality?", they would overwhelmingly respond Yes. If everyone is for quality, and quality is an integral part of the software process, then why is the production of quality software so hard? The problem of quality in the production of software stems from adoption of the philosophy of appraisal as a means of producing good quality software, instead of prevention. "Quality is conformance to requirements".¹ Phil Crosby symbolizes the process of quality by the establishment of four quality absolutes. These are

1. **A Definition of Quality** - Conformance to requirements
2. **A System for Quality** - Prevention instead of inspection and appraisal
3. **Performance Standards for Quality** - Zero Defects
4. **Measurement to the Performance Standards** - The Price of Nonconformance.

Crosby describes a process by the following diagram.



¹Crosby, Phillip, *Quality Improvement Through Defect Prevention: The Individual's Role*. Phil Crosby Associates, Inc., 1985.

This is a simplistic diagram of what is usually a more complex phenomenon. It is important to understand that the software process is the foundation for quality, and the process is the target for knowledge capture in a knowledge-based CASE Environment.

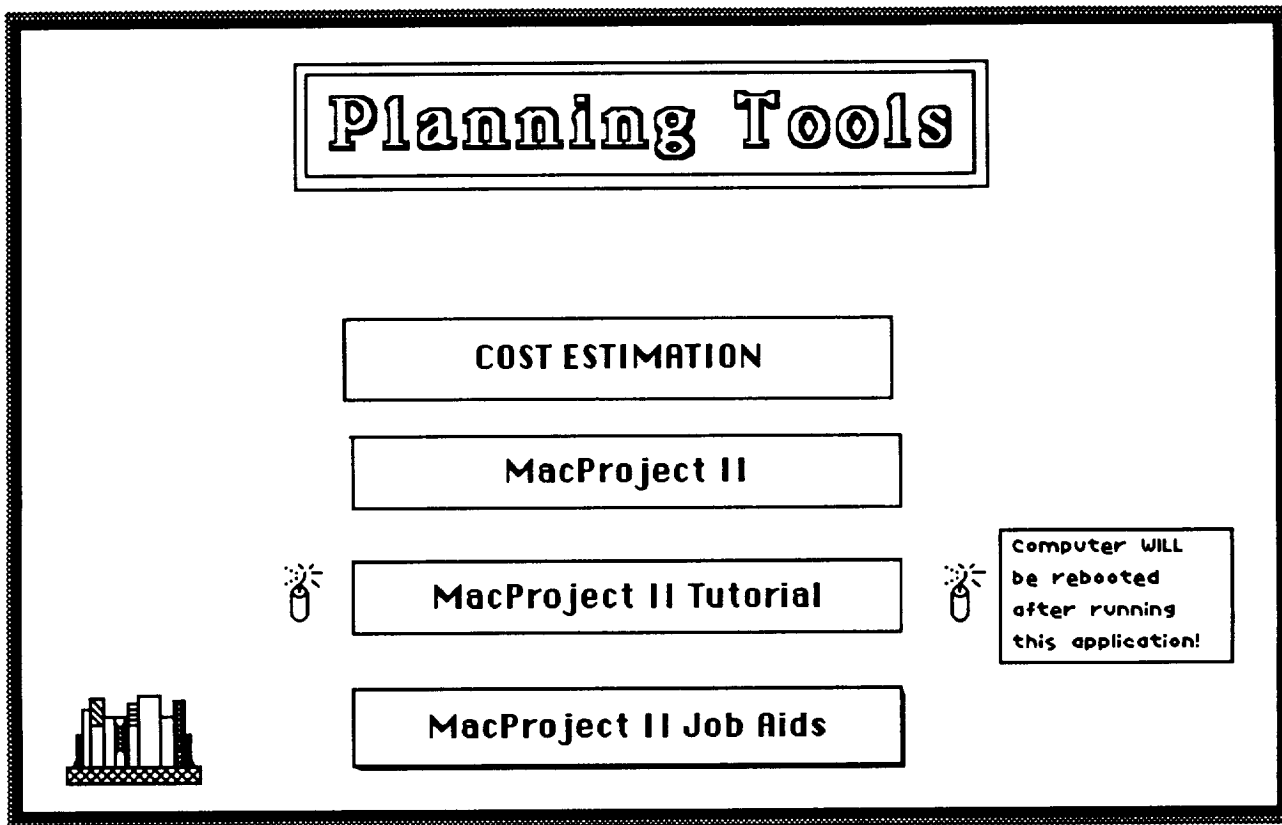
The term CASE refers to a computer system tool which provides the capability to assist in the software development process, hence, Computer-Assisted-Software-Engineering. The proliferation of CASE tools has fairly recently met with a tremendous amount of enthusiasm from the software community. There are lots of good CASE tools that are present in the marketplace today. The question arises "why another CASE Tool?" The idea behind the **Poor Man's Case Tool (PMCT)** currently under development at the University of Alabama in Huntsville is to establish a research tool for exploring new ideas about Software Engineering. The CASE vendors while providing a critical service to the software industry, have not provided a low cost approach to an experimental platform for the study of the software process. The idea of an experimental platform to explore new techniques, methods, policies, and environments gives the software engineering community the ability to try new approaches that would not be feasible within the constraints of the software process. The fact that CASE tools are in great demand is partially due to the change in software needs. Programs should be efficient, easily maintained and modifiable, however this is not always the case. Large Embedded Software Systems such as the software for NASA's Manned Space Station and the software support for the Strategic Defense Initiative call for the support of an efficient organizational tool designed to support the software engineering process. Many times, the process and the product are poorly documented and this leads to problems in the conformance/non-conformance determination. The use of knowledge-based CASE tools allows documentation to be generated as an active part of the system construction and not be a burden to the overall purpose. Therefore, the integration of, change of, re-evaluation of, and

implementation of the entire system should be performed with the minimum amount of effort. A fundamental objective of the research program at University of Alabama in Huntsville is the introduction of quality as a measurable component of the software support environment, and the inclusion of metrics to support level of improvements in the Software Assessments.

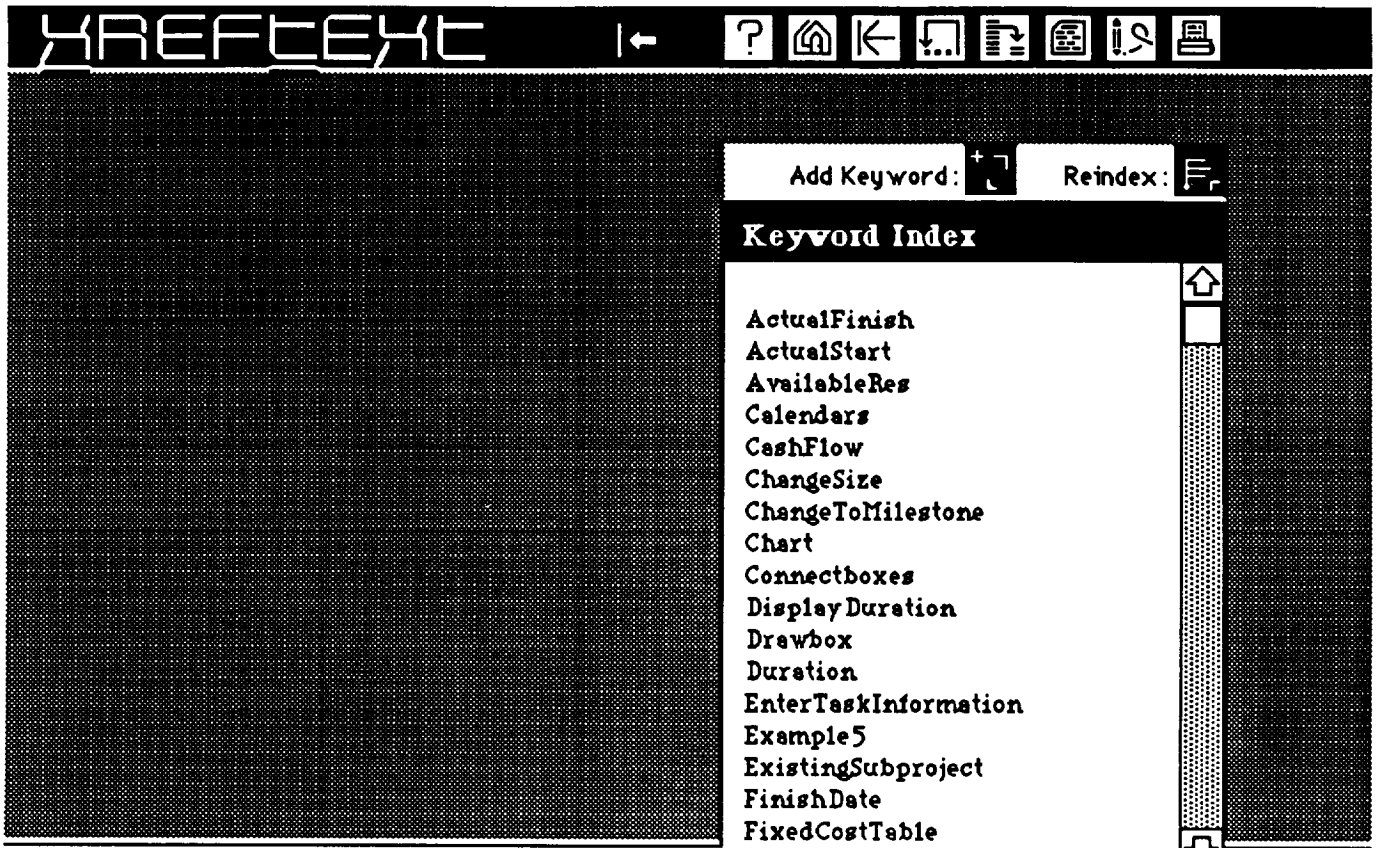
In the first version the pert chart was the significant portion of the planning process. This is implemented with MacProject II. It features schedule charts, calendars, resource tables, fixed cost tables, cash flow charts, and task time lines. The schedule charts are made of task bosses joined together with lines to show the sequence of events. The critical path is marked with a bold line, so the analyst can easily see which task has to be done next for progress to continue smoothly. If a task is a major point in the project it can be marked as a milestone. As the project progresses tasks that are completed can be marked finished. You can list up to eight resources per task, and because different resources will have different work weeks, work week calendars are provided. In each project or subproject there are eight calendars available, each of these calendars can be assigned to a particular resource. MacProject II uses the calendars to calculate the expenses per week, and that figure is entered automatically in the cash flow table. The fixed cost table is used for one time expenses or income such as tools, equipment, and loans. The fixed cost table is added to the cash flow table, so that the planner will know at any one time how much money is available. The task time line is a graph showing the progress of the project. It shows the percent completed of each task. This gives a user a good idea where he stands and which tasks need more attention. Any changes made in the number of resources or cost will automatically be reflected in the appropriate tables and graphs.

Sample Example

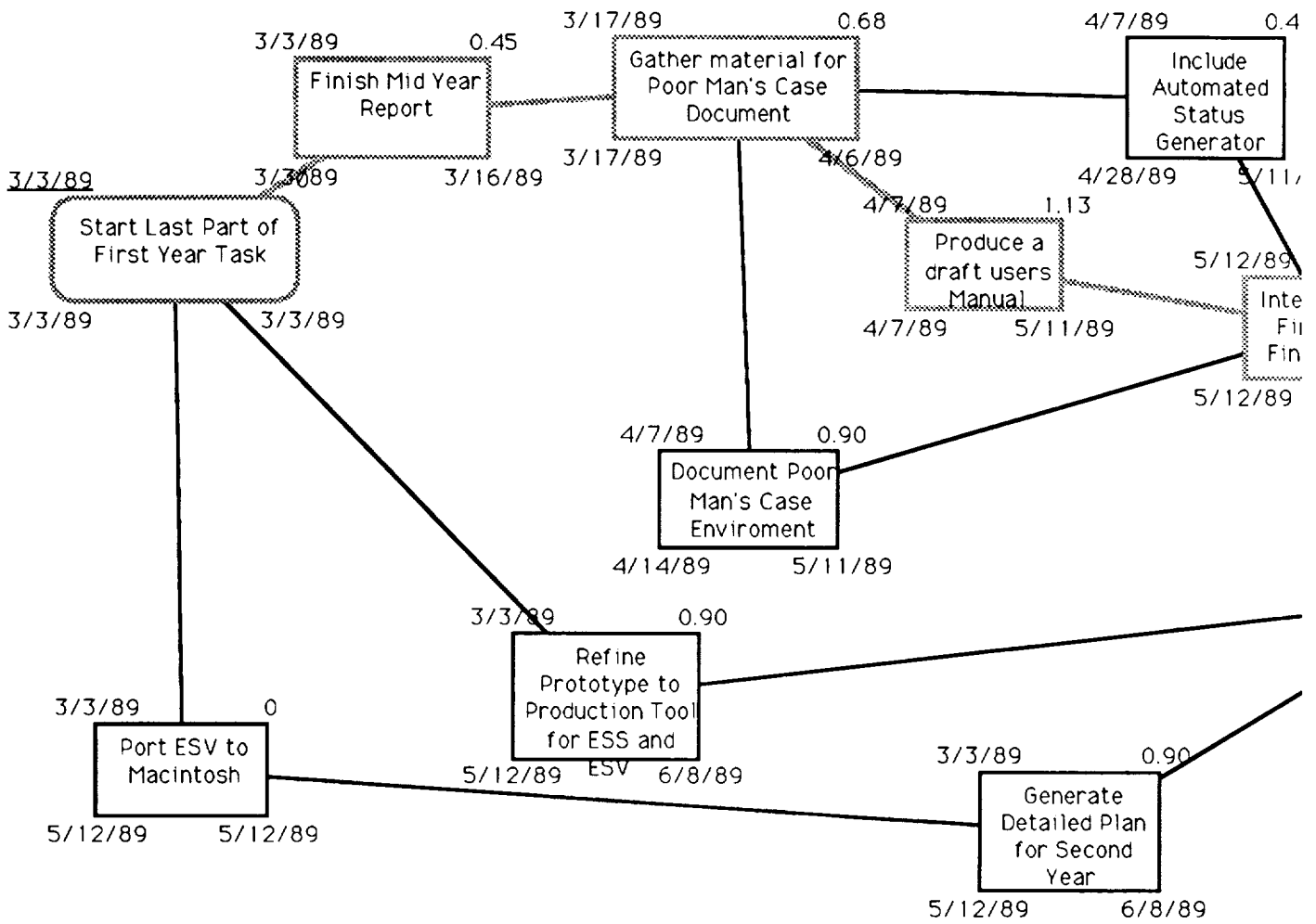
The following represent the types of planning tools provided in the Poor Man's Case Tool. The user when entering the planning will be given the option to select from the job aids, the use of COCOMO, Macproject or a tutorial on Macproject.



All sections contain self-contained job aids. The following is a sample of the card that is the key to the job aids on the pert chart. The user may select any option or sequence of options for explanation on how to prepare a pert chart.



The original agreement on the development of this tool was that we would use no more than existing software that was on the Macintosh in Room A238 in Building 4487 at NASA at the Marshall Space Flight Center in Huntsville, Alabama. This is an example of the use of Macproject. We did not develop this just to integrate this into the overall framework of the design of the PMCT. The following chart is the standard pert chart. The numbers around the boxes are earliest start, duration, latest finish, and latest start, starting at the top left and reading clockwise.



In order to monitor the planned process, the addition of the Project Manager from Focal Point II was added in addition to Pert and COCOMO.

File Edit Go Stacks Launch Projects

Project Record

Project STEP Prototype Management Project # 100 Proposal
 P.O.# Project

Client # 100 Name TI DSEG
 Contact #1 Marshall Bynum Telephone 575-3536
 Contact #2 Telephone

Category	Estimated	Actual
Labor	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Materials	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Expenses	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
TOTAL	\$	\$

Start Date 6/12/89 Invoiced to Date \$
 Due Date 8/15/89 Payments to Date \$
 Date Completed Next Follow-Up •←-8/2/89 Net Outstanding \$ []

Status []

In this frame you see the project overview. It gives project name, project #, Client, Contact, Cost, Next follow-up time and status information. This frame will break down into other frames.

Notice the icon on the right column of the diagram that resembles a tree structure. This is the task breakdown icon. It can be used to model a traditional work breakdown structure, but it adds many more features. After the specification and the structure of the projects are constructed the project manager, or the software engineer can do task assignment. This does not necessarily have to be done by management. This tool can be viewed as just a productivity improvement tool by the participating individuals on the task.

File Edit Go Stacks Launch Tasks

Task Manager

+ Project Task Supervisor **Warren Moseley**

Task Summary

Task Description

.....

.....

+ Task Responsibility

Date Agreed To Date/Time Sent

Original Due Date Revised Due Date

Original Budget \$ Revised Budget \$

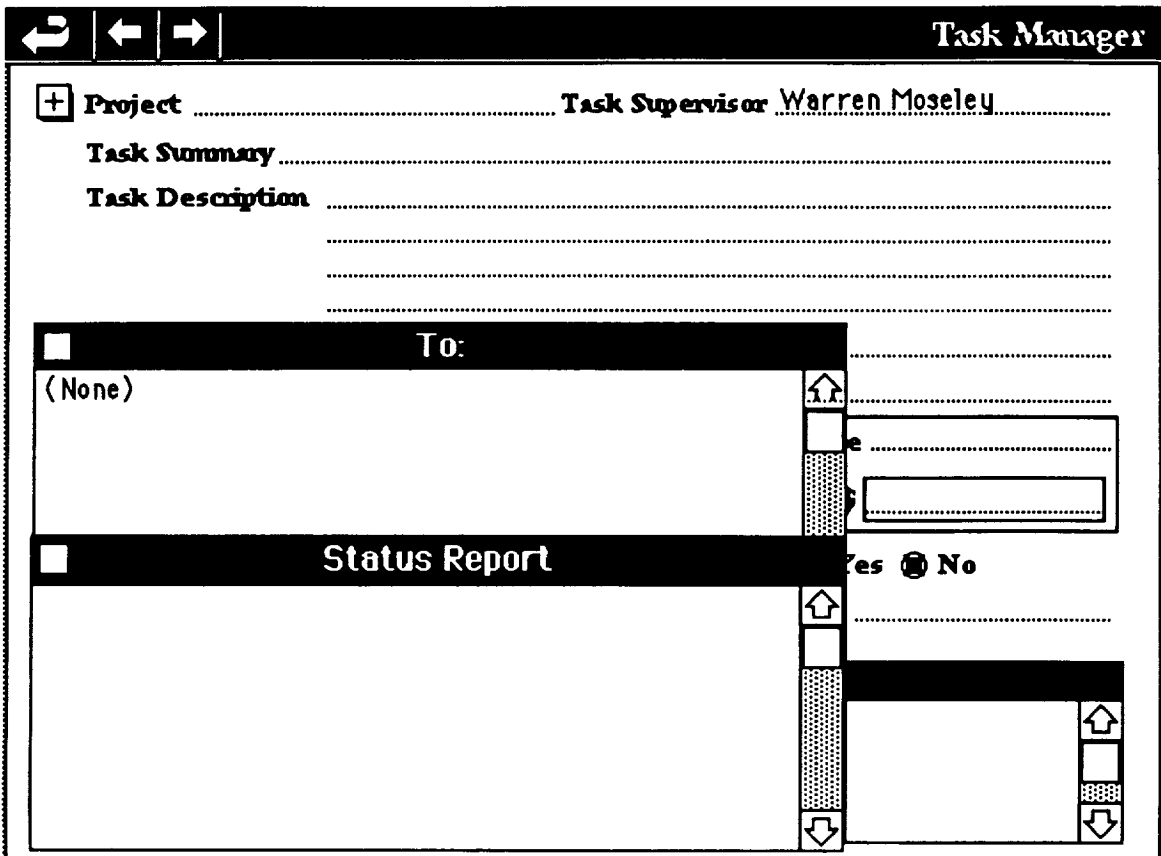
+ Last Report Date Completed? Yes No

Completion Date

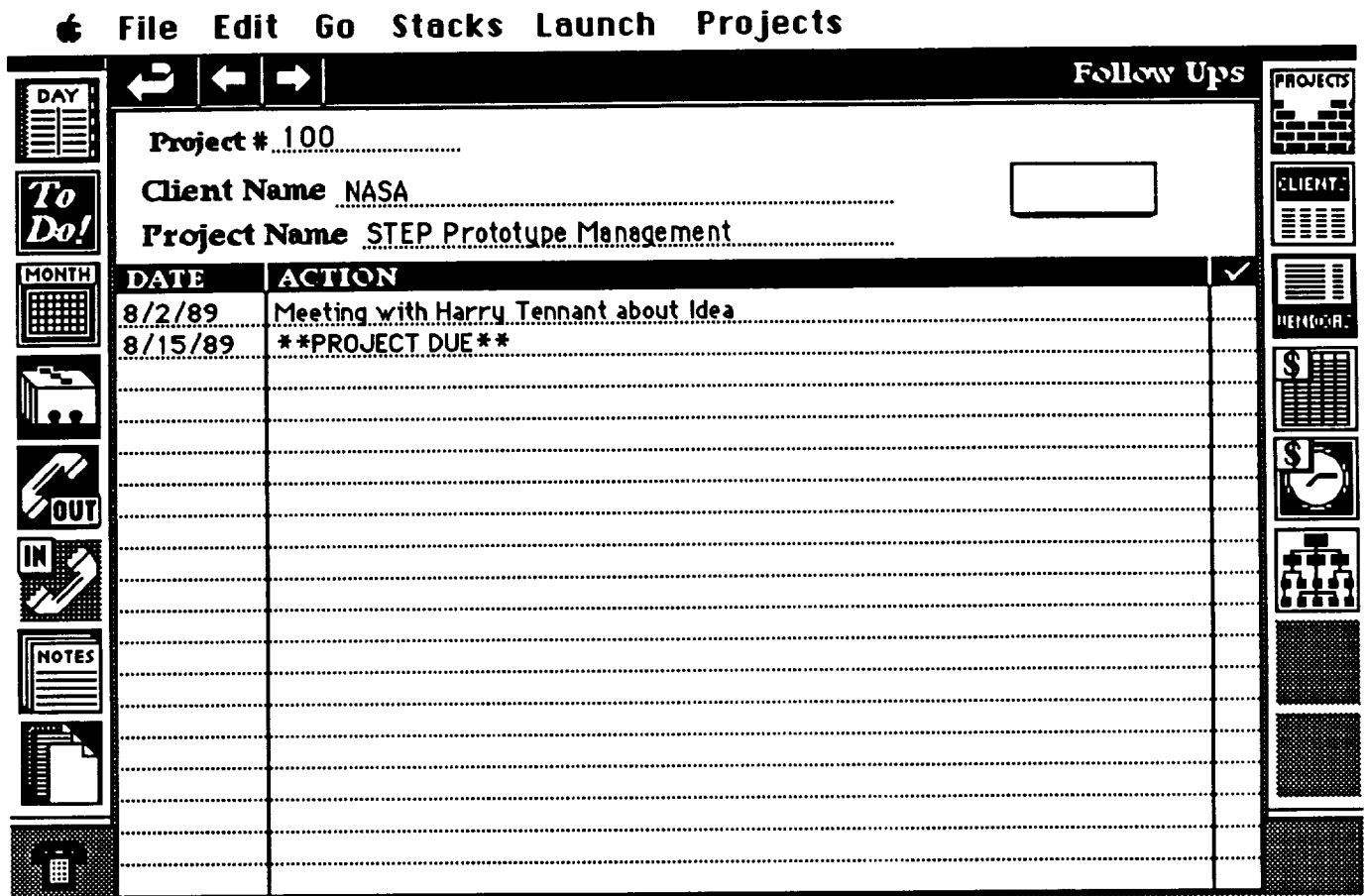
Delegated Sub-Tasks

Due	Who	Task

In the task manager the above frame serves as a main menu. Observe the section labelled Task Responsibility, and Last Report Date. Beside that section is a radio box labelled completed. This is designed so that when a user finishes a task it can be checked off, and when the report section of this program is run, it will generate a status report of the completed task. Frame below shows those two sections exploded. In the "To" box the person describes the reporting responsibility of the person or persons assigned. The bottom is the actual project status, and this is the weekly status reporting slot.



In the project summary frame above there was a next follow up. The next frame demonstrates the project follow up record.



The many reports available from focal point are in addition to the many types of reports that can be derived from Macproject. Just as all of the other parts of the PMCT, this too has a job aid and a tutorial concerning its usage.

COCOMO Model for Cost Estimating

Description

Parameters of the COCOMO model.

AEXP - Application experience - rates the experience of the project team on similar applications. A 'very low' rating is given for less than 4 month experience and 'very high' for more than 12 years.

ASCAP - Analyst capability - rates the analyst team in terms of analysis ability, efficiency and the ability to co-operate.

BASIC MODEL

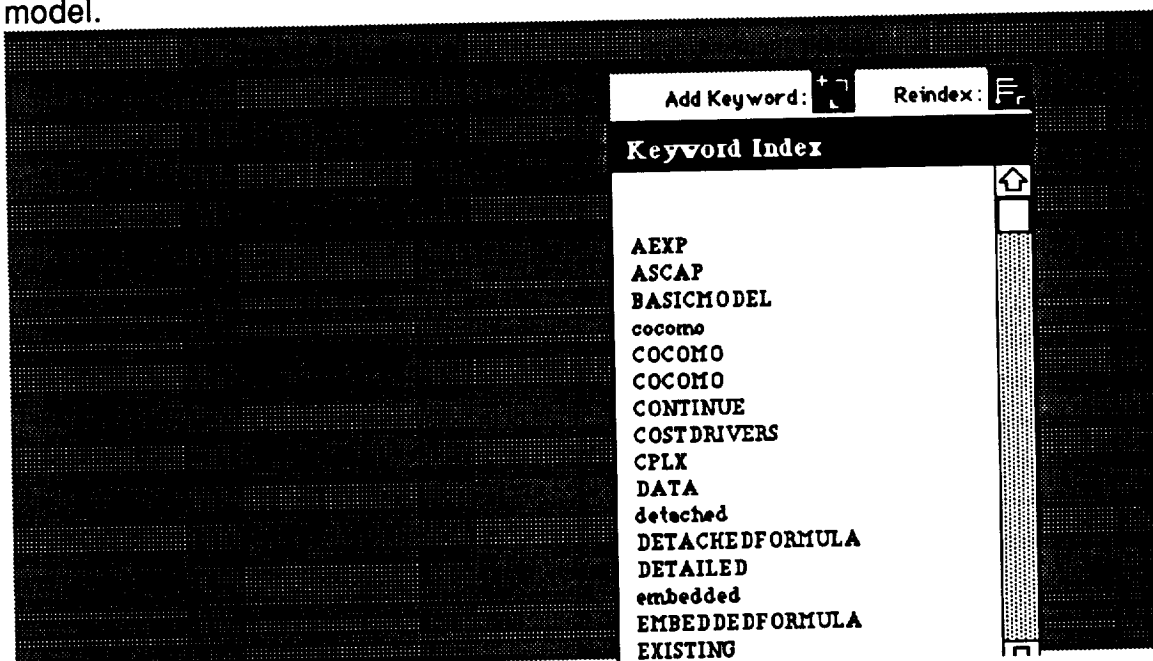
The basic model estimates the cost of a project in a quick and rough fashion. It is used mostly for small to medium software projects, it uses three modes of software development organic, semi-detached and embedded COCOMO.

CONSTRUCTIVE COST MODEL

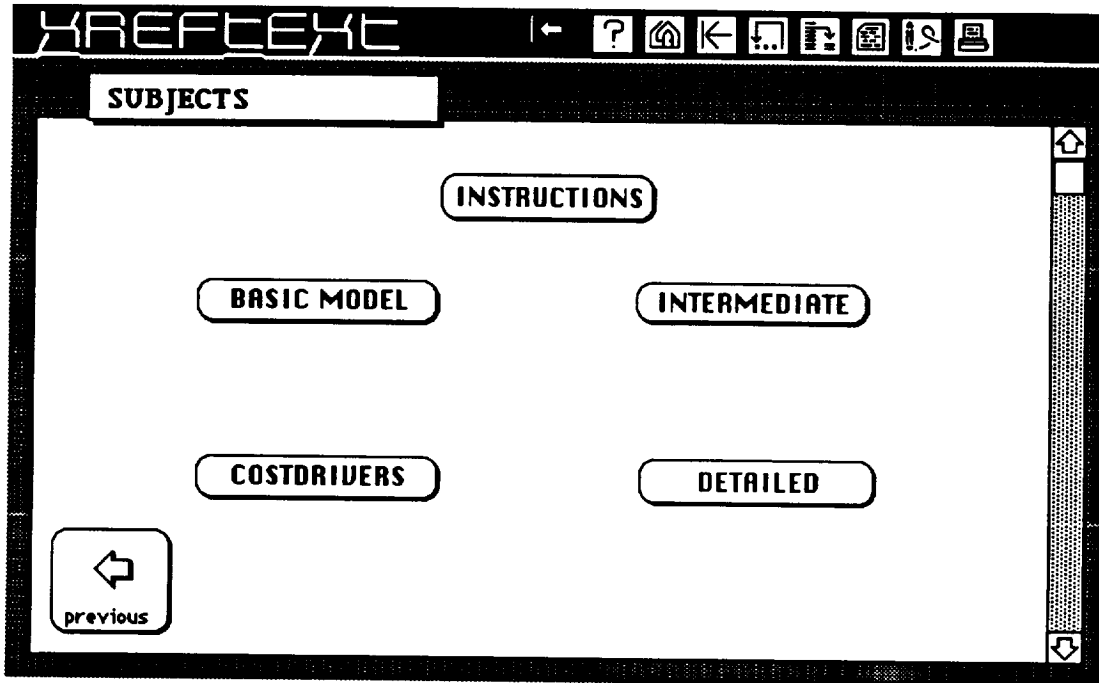
Boehm's Constructive Cost Model (COCOMO) will be used for the software cost estimation. COCOMO is a hierarchy of software cost-estimation models which includes basic, intermediate and detailed models. Because the project environment will be different for each project, cost drivers are used to make estimation more accurate.

Sample Example

The following is a sample of the job aid on the COCOMO cost estimating model.



The user can choose the level of difficulty of the system.



They will be able to receive help and calculate any level of the COCOMO cost estimating model. This was the model chosen to integrate into the prototype, but in later editions the user will be able to select from other popular cost estimating models, or to include their own into the structure.

XREFTEXT [Navigation icons]

DETACHEDFORMULA

MANPOWER COST
 $K(m) = 3.0 (S(k)**1.12)$
 K(m) is manpower cost in man.months.
 S(k) is the size of the project expressed in thousands of delivered source statements.

DEVELOPMENT TIME
 $T(d) = 2.5 (K(m)**0.35)$
 T(d) is the development time expressed in months.
 K(m) is the manpower cost derived above.

BACK

The following is a sample of the Organic Formula.

XREFTEXT [Navigation icons]

ORGANICFORMULA

MANPOWER COST
 $K(m) = 2.45 (S(k)**1.05)$
 K(m) is manpower cost in man.months.
 S(k) is the size of the project expressed in thousands of delivered source statements.

DEVELOPMENT TIME
 $T(d) = 2.5 (K(m)**0.38)$
 T(d) is the development time expressed in months.
 K(m) is the manpower cost derived above.

BACK

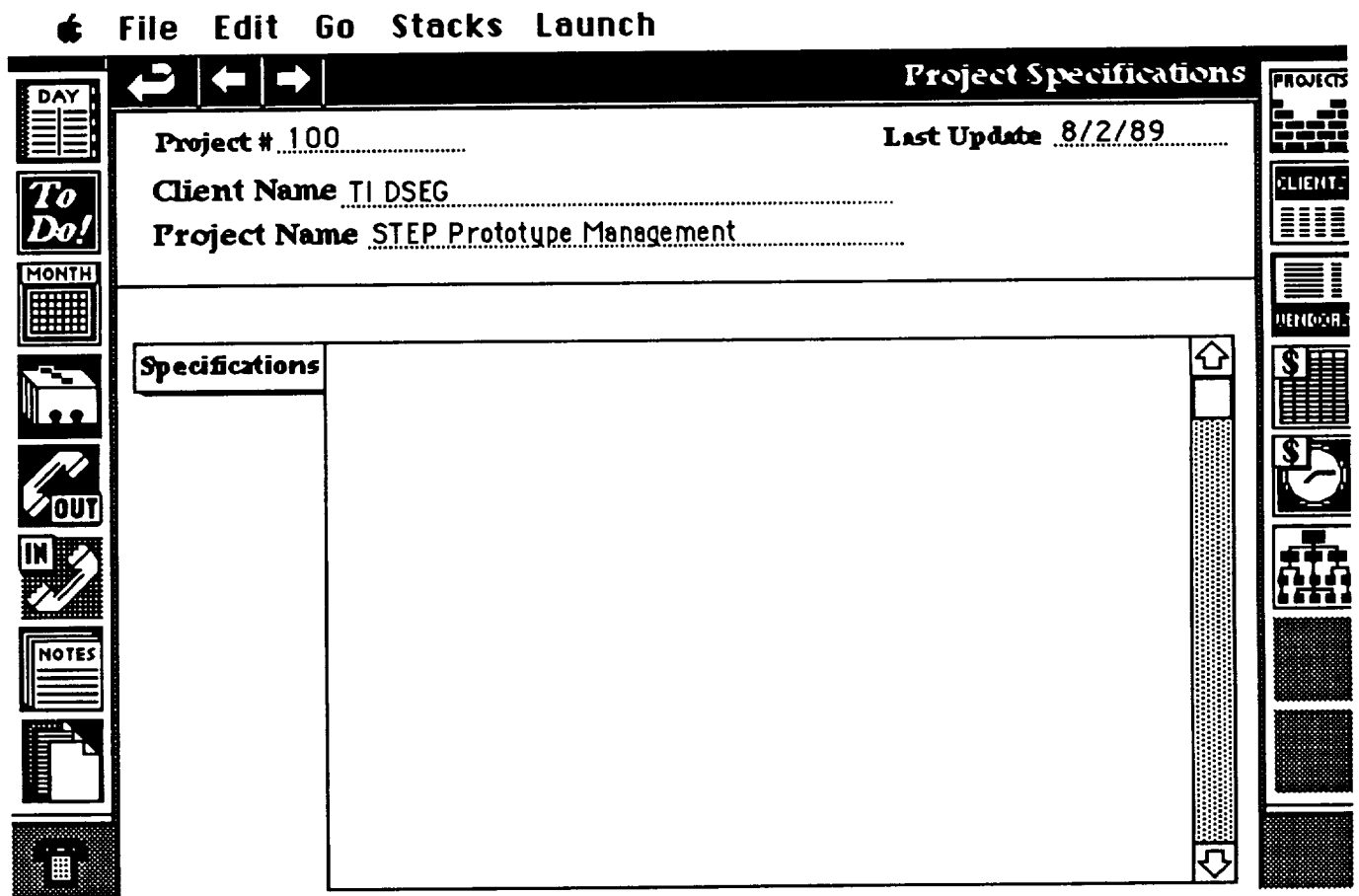
Command-Click goes. Option-Click creates. Delete Card: [X icon]

Project Data Base

Specification Generation and Analysis

Specification Generator Tool

In the following frame the user can see the section which can contain the project specifications. In the earlier version there is a specification data base, in order to knit the pieces together the user must come to this frame in the planner and then enter the Specification Tool, this allows for the monitoring of the individual pieces.

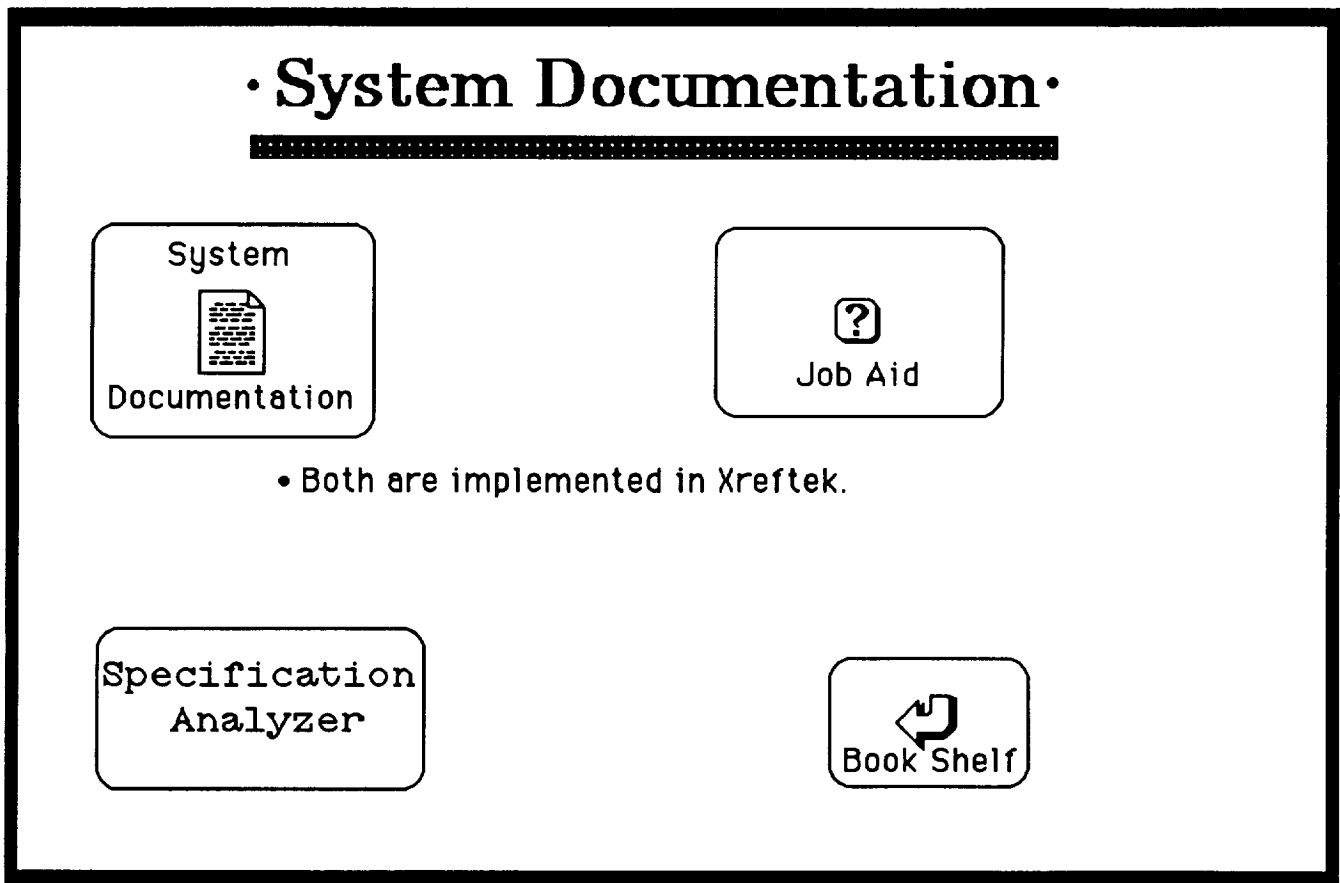


Description

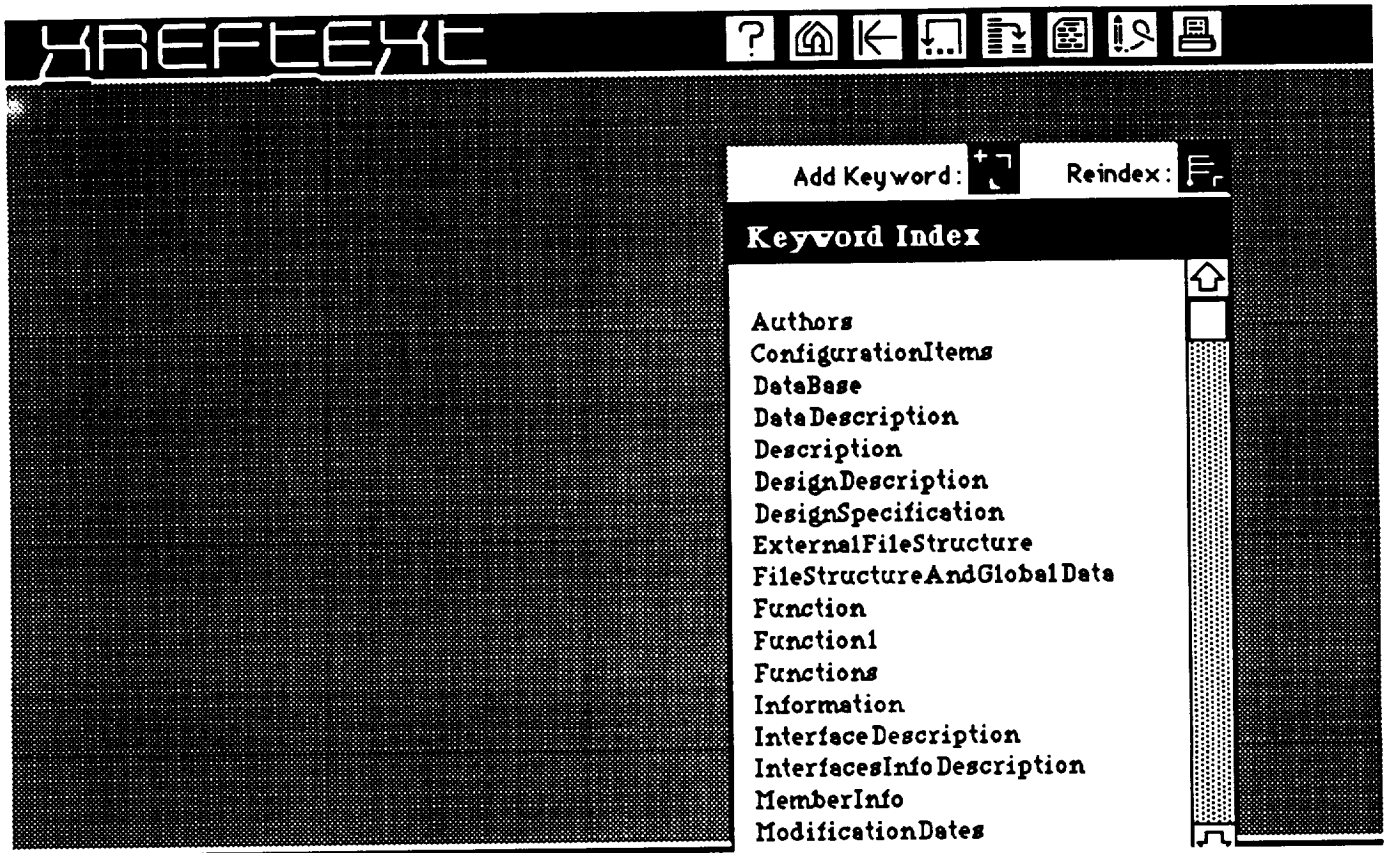
The project data base integrated with the data dictionary and the drawing tool serves the focal point of the PMCT. It is the place where the user can store all of the project documentation, link to all of the system drawings, and in turn to the data dictionary.

Sample Example

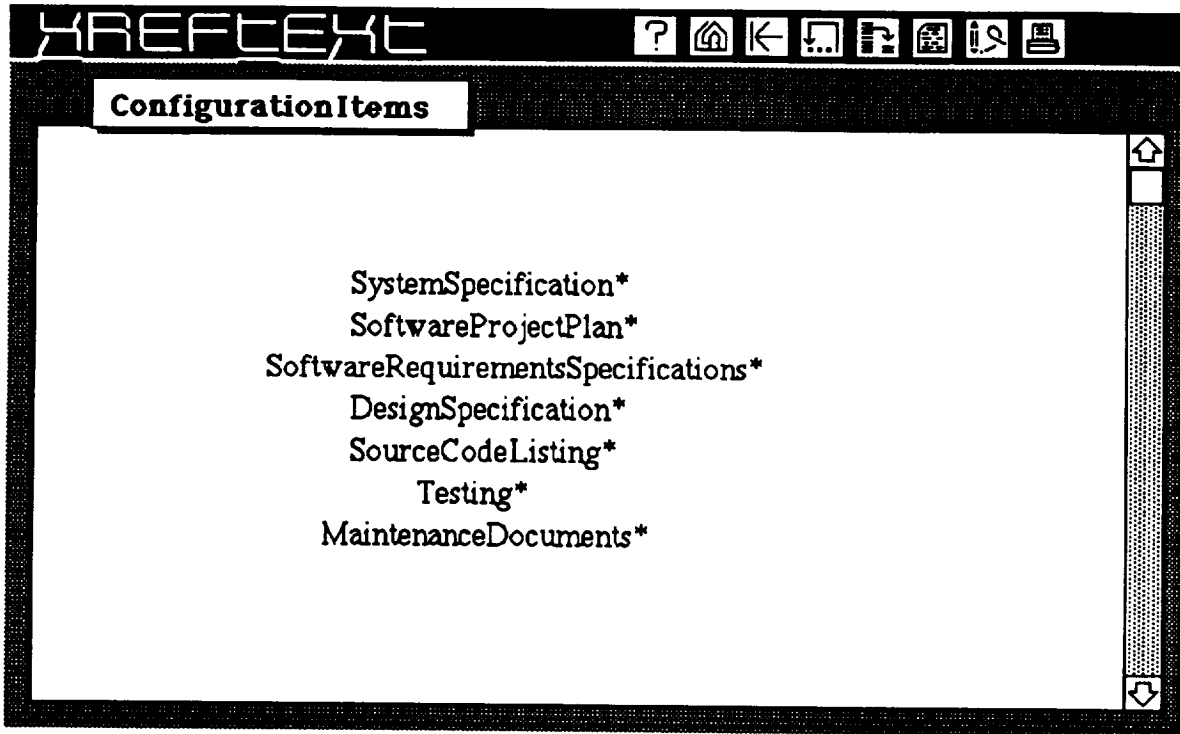
The following is the main menu for the project data base. It is important to be able to enter the documentation into the data base but be able to analyze the existing specifications.



When the user selects project documentation the user will be allowed to view or enter any part of the document in any part of the system that is relevant. It is this hypertext feature that makes the storage of specifications elegant.

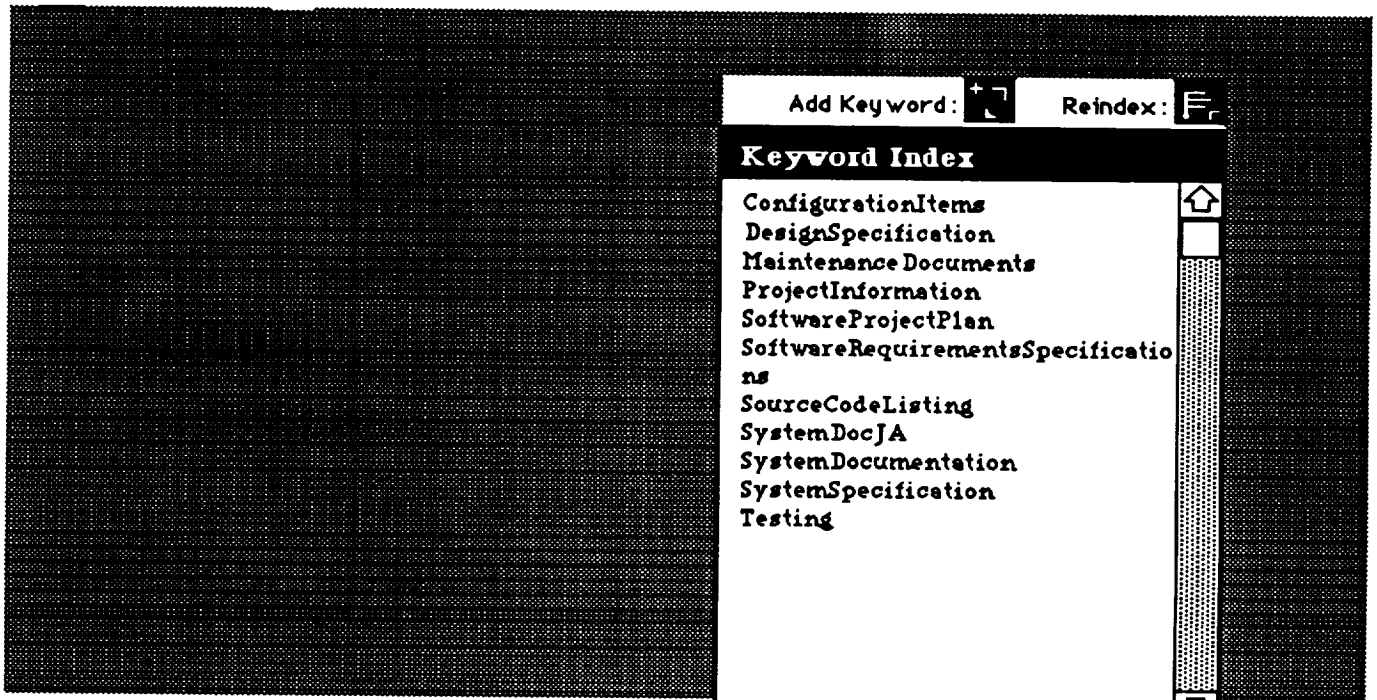


A sample of the project data base is the configuration items card. Any field that has an asterisk will be active and can be selected. If the user wants to go to System Specification then they must just click on that item.



Job Aid

There is also a self contained job aid on the use of the project data base.



Time and Motion Studies on The Software Engineering Process.

One of the salient features of the PMCT is that it is build on a hypermedia based environment, and in the background there exists a monitor to determine the activities for each utilization of the tool. One can monitor each frame, the time entered, the time exited, the sections of the screen that were active in each of the user interfaces to the system. The concept of measurement in the quality process demands that we have time and motion data to help modify the process so that the user can produce quality software. This leads to the section of this research on Requirements.

Building Requirements that are Conceptually Traceable

Requirements Traceability is a method of demonstrating the conformance or non-conformance of the process selected. Usually this is demonstrated by some mechanism that will show the links between the final product documentation and the requirements document. This is usually done by the means of a requirements traceability matrix. In DoD-Std -2167a, the Department of Defense has mandated that traceability of requirements is a critical and required component when delivering mission critical defense systems. Requirements traceability is a method to ensure that not only is a software system correct, but that it is also complete. It gives a path from requirements to code that the developer can trace in either direction. These traceability links are essential in the verification of the component in question, but are also a valuable tool in the assessment of a software change.

In light of the discussion of quality and Crosby's four absolutes, the problem of the monitoring and mapping of conformance/non-conformance from the process to the product is an important issue. In the use of static documents this is quite difficult. Here we introduce into the experimental platform the concept of active knowledge bases as software requirements. In the utilization of the PMCT a person does not build documentation. This is a by-product of the knowledge capture process. Instead of being a document for traceability the specification for a system is an active component

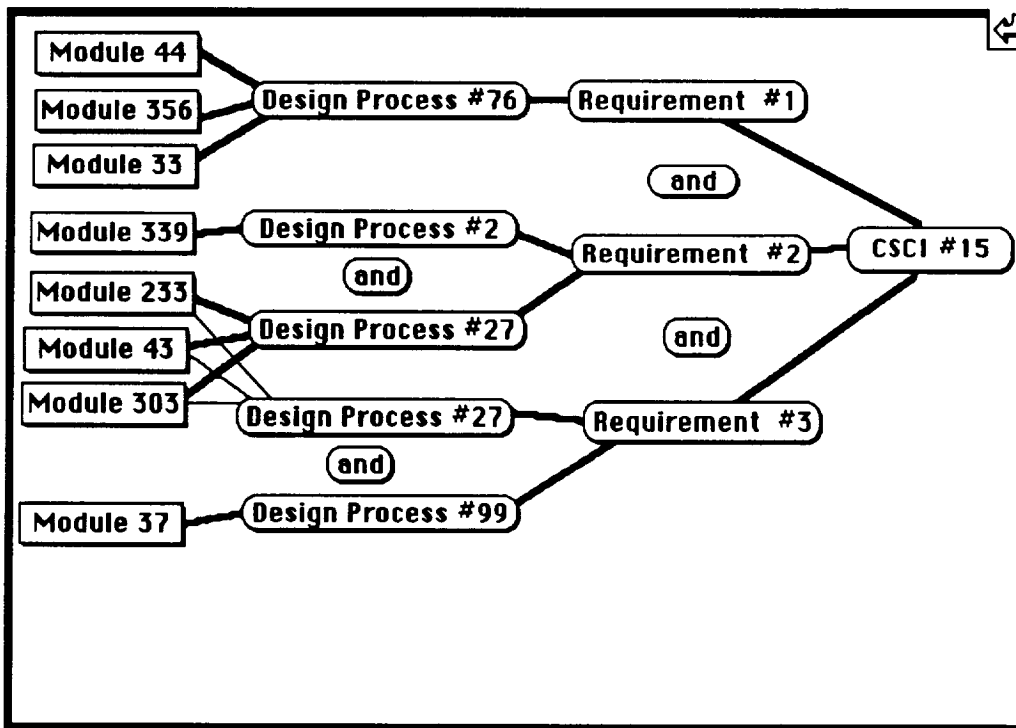
(a knowledge base) and this knowledge base is **executable** and the traceability aspects of the problem are addressed in the explanation functions of the expert system tool selected. There are typically two types of questions that an acceptable expert systems shell should be able to answer. These are

1. Why did you arrive at this conclusion?
2. How did you derive such an answer?

These also are two important issues(questions) in the concept of traceability:

1. Why is this subcomponent necessary to conform to the requirement?
2. How does this subcomponent trace to its requirement?

The following is a design of a Computer Systems Configuration Item(CSCI) in a real-time embedded system, created with the PMCT.



It contains three major requirements which decompose into 4 designs which decompose into 8 modules. Design processes are captured, put under configuration

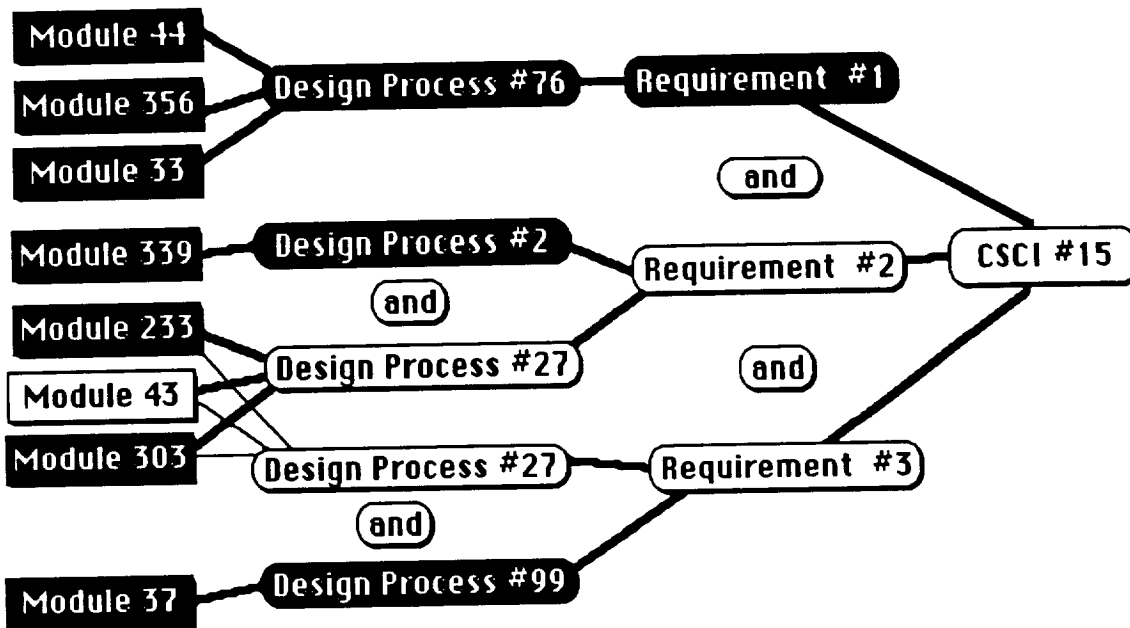
control as each CSCI is defined, placed in the reusable repository, and put into the active knowledge base which reflects the product to be delivered. Notice also that design process #27 is a part of requirement #2 and requirement #3 also. Reusable components (designs and code) are becoming an important issue in the design of complex systems.

As each CSCI is deemed necessary, it is entered into the knowledge base. This would provide the expert systems shell with a goal or hypothesis. As each requirement was added to the functionality of CSCI 15 an entry would be made into the the knowledge base. A first level rule would be as follows:

If Requirement1 and Requirement2 and Requirement 3 then CSCI15.

In the building of knowledge based systems there are two basic chaining mechanisms that are important in the building of production rule knowledge based systems. These are backward chaining and forward chaining. Backward chaining starts with a goal and tries to determine if all of the subgoals and premises of the goal are true. It starts with the then part of the clause and tries to get a true response from all of the if parts of the production rules. In our example CSCI 15 would be the goal and the three requirements necessary to satisfy that goal would be the subgoals. These subgoals are necessary to prove that goal true. In determining the traceability of components of a system, one would use backward chaining The three requirements are now subgoals that must be true for the final CSCI to be deemed true(met). As each requirement is addressed by a certain sequence in the design processes, the knowledge base is expanded. The concept of explanation in knowledge-based systems is one of the unique features of such systems. Consider a different version of the above diagram. The areas in black indicate completion. If we question the knowledge-based system as to the status of CSCI 15 it would reply that

CSCI 15 is not met. If we asked why it would conclude that Requirement #1 has been met, but that design process 27 is still not satisfactory to complete the requirement.



Further examination would conclude that design process #2 of requirement #2 and design process #99 of requirement #3 are complete, however module #43 of design process #27 seems to be a problem. If asked how did the system determine that conclusion, the backward chaining mechanism would point out that module 43 is a part of design process #27 which is a part of both requirements #2 and requirements #3. We could at this time focus attention on the module in question. Supposition, Module #43 cannot be met.

We would like to be able at this point to assess the impact of this problem on the entire design. Data-driven knowledge-based systems reason from premise to conclusion, instead of from hypothesis to premise as in backward chaining. This would be important to be able to do in assessing the impact of the design change on the entire system. We could query the system to determine what requirements are impacted by module 43. It is certain that module 43 impacts design process #27 in requirement #2 and requirement #3. If this was a complex system, there could be

hundreds of cases that could possibly contain this scenario. It is very difficult to manipulate that kind of information even if it is contained in a requirements traceability matrix. By capturing the requirements, design and implementation in a knowledge base, the specification becomes an active component in the system and not just a part of the documentation.

Impasse-Driven Software Engineering

In the insertion of quality into the software process it is not just enough to examine each of the premise action portions of the knowledge base. It is possible and quite feasible that the premise could be traceable, but not correctly defined. This is one of the more prevalent problems in the software development process. In the knowledge capture process for a system that uses a basic absolute of quality that "quality is conformance to requirements", the implementation of a wrong requirement presents a problem with the entire approach to quality. There must be means to analyze the requirements to determine the conceptual integrity of the individual components. The first part of the knowledge-based approach to requirements traceability depends on the capture of the knowledge into production rule format, however this is the least important facet of the measurement process. This section of the traceability will focus on the construction of software requirements and will propose a radically new model of building and evaluating requirements. This approach is described as Impasse-Driven Software Engineering.

In previous studies Guindon and Curtis 88¹ suggests that a software methodology built on a cognitive model of learning has significant potential for making

¹Guindon, Raymonde and Bill Curtis. 'Control of Cognitive Processes During Software Design: What Tools are Needed', Proceeding of Human Factors in Computing Systems CHI '88, Washington, D.C.: May 1988.

the software development process simpler, and in turn allows the software developer to work on more significantly complex domains with the same amount of cognitive toil.

Software Engineering task descriptions should include in the list student and teacher. The knowledge capture process involved in the creation of a software specification contains a learning/teaching dualism that interprets instructions, thereby incorporating new information from the environment into the conceptual structure of the learner. It is this incremental inclusion of information into the problem structure, and the translation of this information into the solution structuring that drives the software engineering process.

During the early phases of the design process, when requirements and specifications are vague, initial design control strategies are based upon a cognitive model of both knowledge sources and process sources. Guindon and Curtis have examined software designers during this initial phase and have found that the design process is partially controlled by recognition of partial solutions at different levels of abstraction. This is considered an opportunistic design process and contains only a minimum number of breakdowns which include:

- Difficulty to merge partial solutions into a complete solution

- Difficulty in remembering to return to postponed subproblems

The difficulties with the opportunistic design process can be further analyzed by examining the learning process and design process as a knowledge communication method. '**Getting stuck**' while problem-solving is called an Impasse. VanLehn¹ explains that the impasse occurs when the step that the student (or designer) should execute next cannot be performed, and is not clearly understood. Inductive learning

¹VanLehn, Kurt. 'Toward a Theory of Impasse-Driven Learning', Learning Issues for Intelligent Tutoring systems, Springer-Verlag New York: 1988.

occurs at these impasses. To enhance software engineering the support software environment must have a vehicle which can produce a sequence of actions that will force the student(designer) to this impasse.

Learning by Asking Questions

Requirements definition in the Software Engineering Process can be viewed as the ability to ask proper questions to the proper person at the proper time, and then to translate the question-answer sequences into the basis for the knowledge that must be communicated to the persons that will implement the software system. Question-answer sequences provide the means of evaluating current thought patterns and mapping these patterns into the general concepts to be represented within the problem domain. These sequences offer contributions to indicate the knowledge structure needed, although some of these contributions may be relevant, while others will not be relevant to the unique domain. The goal of the requirements-producing process is to develop a knowledge-based computational model of the specified problem domain by watching, listening, and questioning the end-user and by incrementally testing the validity of the knowledge gained in each of these sessions. In the past, much of this process has been ambiguous, dull, and boring, because of the amount of redundancy in the communication process. This redundancy occurs because of conceptual misunderstandings between designer and user. The misunderstandings occur because there is an impasse between problem domain and problem solution, and the designer needs help in the identification of the impasse and help in repairing this impasse.

Conceptual Mapping of Software Specifications

To actually create the conceptual structure that will be a software specification is one of the salient features of the PMCT. The conceptual aspect of meaning is illustrated by a classification system designed by Peter Mark Roget in 1852. Best known for his Roget's Thesaurus, Roget produced a list of 1,042 concepts and

identified the relationships among them as a hierarchy. He then mapped ordinary natural language (English) words to these concepts. Roget called these concepts **"related areas of meaning"**.

There exists one aspect of Roget's taxonomy, however, which demands immediate attention, and that is the fact that his classification system is domain independent. It not only contains information concerning tangible concepts like PHYSICS and MATTER, but also intangible concepts like AFFECTIONS and SENSATION. It, therefore, appears that Roget's Conceptual taxonomy has a wide variety of application, especially in the application of Artificial Intelligence to the Software Engineering Process.

Roget's taxonomy is explicitly referenced through our natural language, the words we use everyday. Each one of us have different vocabularies and we use different words to communicate an idea (or concept) than someone else might who was referencing the same idea (or concept). The Word Mapping Process takes the basic assumption that to communicate an idea, we choose words from our vocabulary according to the conceptual relationships among those words. This implies that there exists a "principle of commonality" among the words chosen to convey an idea, ie. they must have a conceptual relationship to that idea being communicated.

Word Mapping and the Concept Induction Process[©]™

Ausborn ¹ introduced a process called the Concept Induction Process[©]™. This process of mapping of words to concepts can be achieved by the use of the '**related areas of meaning**' developed by Peter Mark Roget in 1852.²(Roget's hierarchy can

¹Ausborn, Carolyn. 'Mapping Words to Concepts', Proceedings of Expert Systems: Solutions in Manufacturing, Michigan: 1989. Society of Automotive Engineers

² Chapman, Robert L.,ed. 'Roget's International Thesaurus', Harper & Row Pub., New York: 1977.

be found in Roget's International Thesaurus). The Concept Induction Process^{©™} is a domain independent process to map words into specific concepts into classes of concepts, and to allow the user to monitor, manipulate, and generate these conceptual classes. After producing a conceptual representation, the user can determine the point of contact in which a specification changes. It is this type of monitoring mechanism that will not only lead to traceable requirements, but traceable correct requirements. The traceability factor must include software requirements traceability, but also the mapping of idea to concept to product. This in the past has been a very difficult subject to research, but with the introduction of a CASE environment that supports time and motion studies, demands active traceable knowledge components, allows process to define tool needs as opposed to tools dictating what can be accomplished, progress can be made in understanding just what is the software process.