SOFTWARE ENGINEERING TECHNIQUES: DEVELOPMENT, USAGE, AND FUTURE

An Honors Thesis (ID 499)

by

Robin L. Ridgeway

Thesis Director

William J. Brown

Ball State University

Muncie, Indiana

November, 1982

Expected Date of Graduation: Fall, 1982
TABLE OF CONTENTS

LIST OF TABLES ........................................ iii
ACKNOWLEDGEMENTS ....................................... iv
I. THE PROBLEM ........................................... 1
II. THE SOFTWARE LIFE CYCLE ................................ 10
III. TECHNIQUES FOR SOFTWARE CREATION ..................... 18
IV. MANAGEMENT ISSUES .................................... 40
V. SURVEY OF COMPANIES .................................. 57
VI. SUMMARY AND CONCLUSIONS ............................... 75
BIBLIOGRAPHY .............................................. 94
APPENDIX A ............................................... 97
APPENDIX B ............................................... 99
APPENDIX C ............................................... 110
APPENDIX D ............................................... 119
APPENDIX E ............................................... 123
LIST OF TABLES

TABLE 1
LOW-DEVELOPMENT COMPANY ANALYSIS . . . . . . . . . . . . . . . . . . . . 59

TABLE 2
MEDIUM-DEVELOPMENT COMPANY ANALYSIS. . . . . . . . . . . . . . . . . 61

TABLE 3
HIGH-DEVELOPMENT COMPANY ANALYSIS. . . . . . . . . . . . . . . . . . 64
ACKNOWLEDGEMENTS

The first acknowledgement is to Dr. William F. Brown, without whose knowledge, guidance, and personal contacts this paper would have been impossible. Dr. Brown's dedication to his students is much appreciated.

Secondly, I would like to express my appreciation to the Indiana companies who participated in this study. Without their valuable information this study would not have the value that I hope it has.

Thirdly, I would like to thank the Ball State University Department of Mathematical Sciences for the use of their word processor. Also thank you to Dr. Clinton Fuelling, Dr. Norman Lee, Ms. Judy Bonneau, and Mrs. Janice Cool for their patience and help.

Lastly, thank you to Jeff Shelton for his help with the graphics in this report, and to my parents and friends for their support and patience.
I. THE PROBLEM

A. INTRODUCTION

Since the advent of the generation of computers utilizing transistor and later silicon chip circuitry, it has become increasingly apparent that it is not the fault of equipment that computer potentiality has not been reached. The date for the physical realization of decision support systems and complex "artificial intelligence" systems has been pushed into the future. The computer industry has reached a crisis point, and hardware technology is not to blame. Software is and will be the major source of difficult problems if new approaches and techniques for creating software systems are not developed and implemented.

"The software side of the computer industry has exhibited particularly poor performance. Software customers have typically come to expect the software they receive to be poorly documented (if at all), cost more than they originally expected, be poorly designed, and delivered later than originally promised."[BERS80]

Much of the software developed seemed to be a solution in search of a problem, and was so badly written that it was hard to maintain. Many of these problems can be traced to management and development techniques used in software production. Software has typically been produced as an art form, with products being changed at the whim of the creator.

In order to solve these industry-wide problems, a more disciplined approach to software creation was needed. Through the work of many data processing professionals, techniques were
developed to provide a software discipline. Software using these techniques was said to be "engineered." Software engineering techniques bring software production more into the realm of science. By applying formal techniques, software creation and operation can be properly controlled, documented, and implemented.

B. RESEARCH OBJECTIVES

The purpose of this paper is twofold. First, the techniques and methodologies of the software engineering discipline will be studied. The development and detail of the techniques will be presented.

Second, a study of selected companies in Indiana was conducted. Documentation from this study is presented and conclusions drawn. The objectives of the study are the following.

1. Determine if selected companies are aware of the modern software engineering techniques, and if they are, how many and which ones are in use.

2. Study the details of the choice of techniques, and find correlation between company characteristics and technique choice.

3. Study the details of the processes and procedures followed in software creation.

4. Determine how selected companies view the educational programs of universities for preparing potential employees in software development.

C. DEFINITION OF SOFTWARE SYSTEMS

Software systems can be defined in many different ways. First, a general definition of a software system is a system made up of modules containing either a natural language and/or an
artificial language (i.e. a programming language) which are organized to cause computer hardware to perform certain functions so as to yield required results.

Software systems can be defined according to type. There are operating systems, information (DP applications) systems, embedded systems, data management systems, and network systems. The applications systems are the primary focus of this paper.

Software systems are made up of programs organized to accomplish a specific function. A program is the most global unit of execution in a system. Programs are made up of one or many modules. A module is a unit of program text which must be linked together at some point to other modules before execution.[FREE75] A more complete module definition will be given at a later point.

The remainder of this paper will use these definitions when explaining the development and use of software engineering techniques that are used to help create and modify software systems.

D. HISTORY

In the late 1940s and early 1950s, software errors were viewed to be the result of inexperience. It was thought that through a normal maturation process, software developers would soon have errors under control, and growing software sophistication would justify the faith put into computer technology. Yet, as the 1950s became the 1960s, this maturation process did not take place.[WASS80a]

In the early 1960s, software developers began to attempt increasingly complex systems in order "to make innovative use of
computers and information technology."[WASS80a] Plans were made for airline and hotel reservation systems, on-line medical records systems, data base management systems and manufacturing process control systems. A significant percentage of these projects failed completely. Those completed were far over-budget, late, expensive with respect to the resources required, and/or poorly suited to their uses.

There were several different reasons for the failures.

1. The intellectual complexity of the application itself was so great that either the analyst(s) could not understand the problem completely or could not convey the problem to implementors.

2. The developers did not orient the system to the end users, who then rejected the system as being difficult to learn or use.

3. The analyst(s) didn't take the organizational environment into account in specifying the system thereby leading to an unusable system.

4. Analysts and developers were overly optimistic about the development time, development cost and/or operational cost of the system.

5. There were no intermediate steps during the project at which project management and customer could review the progress and determine system status.

6. The application demanded the use of high technology, which was often not generally available.

7. There was little or no effective communication between the user community and the development organization.[WASS80a]

In addition to this wide variety of problems, each piece of software developed was a "custom design," following no consistent pattern and using little information from previous development projects. Project management could not provide accurate estimates of development times or costs. The entire development
process had gone far out of control. "The major cause of difficulty was the lack of a systematic approach to information system design and development."[WASS80a]

In 1968 and 1969, the NATO science committee recognized this lack and held conferences on "Software Engineering." These conferences were based on the premise that an "engineeringlike form of discipline could be applied to building software systems."[WASS80a] The results of the conferences included a greater awareness of the software problem and new approaches to the building of software systems.

Although the need for a more disciplined approach for controlling software creation became increasingly apparent, software engineering remained primarily in research stages through the late 1960s and early 1970s. The research done had little effect on everyday software development procedures, but did generate key concepts which would provide modern software development with important tools. These concepts included "top-down design, stepwise refinement, modularity, and structured programming. The relationship between programming languages, problem solving, and high-quality software was recognized at this time."[WASS80a] Most attention was devoted to the development and use of individual techniques.

As the decade of the 70s progressed, only a few attempts were made to combine the techniques developed for specific software development problems into methodologies. The main focus of software engineering concepts was the production of code and good programming style.
Software engineering moved away from this focus when the "life cycle" concept was applied to software. The concept was borrowed from the manufacturing idea of the product life cycle. The application of a software "life cycle" shifted focus from code production to every phase of software creation—from recognition of the need for a system by the users to maintenance and modification of such a system. Since analysis was included as part of the cycle, the techniques developed soon included management practices as well as software development practices. The complete life cycle provided a framework to which engineering principles could be applied. These principles delineated the phases of software development. At the present time, much work is being done on development and implementation of tools and methodologies for every phase of software creation.\[WASS80a\] A complete breakdown of the software life cycle phases will be given in this paper.

E. DEFINITION AND GOALS

"Although useful definitions of the term remain elusive, software engineering clearly implies at least the disciplined and skillful use of suitable software development tools and methods, as well as a sound understanding of certain basic principles."\[ROSS80a\] The tools and methods must be applied to all phases of software creation, as the definition in a paper by Barry W. Boehm states. Boehm's definition of software engineering is "the practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them."\[BOEH79\] Boehm makes three main points about this
definition. First, "design" must be given a broad interpretation so that it includes the requirements definition phase of software creation. Second, the definition covers the entire software life cycle, including any redesign or modification normally termed "software maintenance." Finally the point must be made that at the present, the "scientific knowledge" the industry has gained concerning software is very limited, and this is where work should continue.[BOEH79]

Software engineering is emerging as a discipline for managing system development. It incorporates ideas from engineering, management, and computer science.[ZELK79] As such, its goals are oriented not only to those developing software, but also for the users of the systems. The four fundamental goals of the processes and principles of software engineering are "modifiability, efficiency, reliability, and understandability."[ROSS80a]

F. IMPORTANCE OF SOFTWARE ENGINEERING

Why have the concepts of software engineering become so important? The two encompassing reasons are software complexity and costs.

Society's continued automation of the processes that control the quality of life has put more people in contact with and depending on computer systems. The software needed to control the automated functions has become increasingly complex. Not only are the applications themselves more difficult, but also more provisions must be made for system and data security. Without a systematic approach to the creation of software, the complexity of the problem would cause numerous errors. With the current
trend in industry being the use of the computer by those having no specific training, errors in software can cause a major hazard in computer use. Those who do not work with software directly tend to trust implicitly in the results they see when running packaged programs. "Errors in medical software have caused people to lose their lives. And software errors cause a constant stream of social dislocations due to false arrests, incorrect bank balances or credit records, lost travel reservations, or long-delayed payments to needy families or small businesses."[BOEH80]

Society has come to view the computer as "super foul-up devices" instead of the fantastic machines that could save time and money.[BERS80] Software engineering provides a disciplined approach to software creation. By using software engineering techniques, the complexity of software can be broken down and handled efficiently. As the breakdown occurs, less complex modules can be programmed with less errors.

Complex software system cost is becoming prohibitive. The growth of costs for software development is far out-distancing the growth of the economy. In the past ten years, the cost per line of code developed has doubled. The increase in software costs are especially noticeable as hardware costs decrease. Creating software has more "people" costs than the building of hardware, and people costs are not coming down. Not only are software development personnel getting higher salaries, but also there is a substantial cost associated with high turnover in most data processing departments. The constant changing of system specifications can be a high percentage of total development cost. Also, most data processing departments have inherent low
productivity due to competition for limited computer resources. [SETH82]

As high as the direct costs of software development are, the indirect costs are even higher. In overall system development, software is generally on the "critical path." Slippages in the software schedule directly cause slippages in the system delivery schedule. Compensation for the time lost in development usually comes from testing time, with users switching to a new system before it is adequately tested. Correcting errors after implementation that would normally been found in testing can add millions to large scale products. [BOEH80]

Using software engineering techniques can help to control costs by making personnel time more productive and cutting development time. In the following section, the software life cycle will be defined as a basis for software engineering.
II. THE SOFTWARE LIFE CYCLE

The first tool necessary for a disciplined approach to software creation is a software life cycle. The life cycle will provide carefully delineated phases to which techniques can be applied, either directly to a phase or as a methodology covering all phases. While there are many different concepts of the software life cycle in the industry today, the one presented below will provide the basis for further discussion of software engineering techniques. The situation and progression presented are the ideal, and should be interpreted as an example for which information to be presented later can be applied.

A. PHASE DIFFERENTIATION

The software life cycle is divided into five primary phases. Each phase can be subdivided into subphases that accomplish a portion of the total job to be finished in that phase.1 The life cycle is not, at most times, a sequentially executed procedure. Much iteration takes place between phase and subphases, especially through the early stages.

Each phase has inputs, processes, and outputs to be fed into the next phase. It is left to the implementor to decide what form the output will take, but the output should provide a checkpoint to see if each phase has been properly and totally completed, and also to provide feedback on project development to management.

1Refer to Appendix A for a diagram of the software life cycle.
The phases of the software life cycle identified in the following paragraphs are not intended to be the only way the life cycle can be presented, for there is currently a multitude of example life cycles in publication and use. The phases presented highlight some principle features necessary to software engineering techniques.

B. PHASE ACTIVITIES

The first phase of the software life cycle is the Systems Analysis phase. This phase encompasses all information necessary to lay the groundwork for subsequent stages of a system development project. The output from this phase will state why a system is needed, what features it will offer to satisfy the need, and a basic definition of how the system will be constructed under what constraints.

There are two subphases of the Systems Analysis phase. They are Requirements Analysis and System Specification. During Requirements Analysis, "the eventual user of a system discovers a need for which an information system seems to be the answer."[FREE80a] The primitive needs of the user are analyzed within the context in which the system will operate, and user problems are analyzed. In this subphase, the feasibility of creating and installing such a system is studied, and the system's justification is established. Also, the major functions and constraints of the proposed system are identified. The output of Requirements Analysis is a set of general requirements. In the software life cycle followed in industry, this subphase may or may not be formally described or have checkable output.
The second subphase of Systems Analysis is Systems Specification. This subphase uses the general requirements obtained from Requirements Analysis to produce more explicit requirements and statement of constraints. In this stage resource and economic constraints of both the system and the system development process are considered. Functional descriptions of the system are developed along with the constraints on the structure of the system. The constraints are narrowed down from those identified in Requirements Analysis so that only those that will be operational for the new system are included in the specifications. The output from this second subphase and therefore from the entire Systems Analysis phase is the set of specifications of system functions, constraints, and objectives. There is normally much iteration between the Systems Analysis phase and the next phase in the life cycle, the design phase.[FREE80a]

"During design we are working with logical properties and the overall structure of the system."[FREE80a] This phase of the software life cycle is concerned with abstracting operations and data structures of each system task, determining in a precise manner what is to be done by the software, establishing an overall structure and interfaces, and making decisions based on tradeoffs dictated by system constraints. There are two subphases of the Design phase. They are architectural design and detailed design.

In the architectural design subphase, the underlying structure of the problem is formalized. Based on the specifications output from Systems Analysis, and a general
knowledge of any other similar system to the one being designed, major components of the system are identified and relationships between them established. The design of the system at this stage is very abstract, as the total system is being considered. The output from this subphase is a structural description of the system with identification of basic algorithms and major data representations.

Next, the detailed design subphase describes the refinement of the architectural description into more detailed descriptions. This phase stops just short of spelling out all programming details. Algorithms and data structures are spelled out precisely after taking problem structure and programming environment details into consideration. This stage, as well as the architectural design stage, may require several levels of refinement. Decisions on hardware are made in this phase, as well as on the precise detail of interfaces between system parts. Blueprints for the programs are the checkable outputs from this subphase.

A subset of detailed design having increasing importance is data design. Data design involves the design of data structures to insure that program structures will have data independence. Data independence ensures that changes in data formats will not cause major changes in the software. From the specifications and architectural design plans, logical representations of data objects are identified. A data dictionary is compiled to record the data objects and also the modules that operate on them. The abstract data designs are incorporated into the program blueprints.
The total output for the design phase of the software life cycle includes the structural design of the system, blueprints for program production, and a data dictionary identifying major data representations.[FREE80a]

The next major phase in software creation is the Development phase. This is the phase where most of the software engineering techniques developed have been implemented. The Development phase can be broken down into two functions: Coding and Testing.

The Coding subphase encompasses just what its name implies: the actual coding of the modules. The architectural, detailed, and data designs are translated into the selected language. Depending on how the designs were recorded, this may or may not be an easy translation. The output of Coding is a set of coded program modules.[BERS80] It is hoped that the coded modules will be independent enough to be stored and re-used in other systems.

The Testing subphase takes the program modules and conducts test. Usually Testing uses three different types of tests. First, modules are tested with driver routines and data supplied by the programmer. Next, groups of modules are tested to check the interfaces between them. Finally, the completed system is tested and evaluated by an independent group. At any point in the testing, iteration could take place back to any phase or subphase in the life cycle, as errors develop. The output, after all iteration and re-testing is complete, is the working system.[ZELK79]

This is not the end of the software life cycle. After a
A working system is created, it must be implemented. The Implementation phase includes the installation into the user environment. There it must be tested to check organizational interfaces, and users must be trained in its usage. The system documentation must be verified for completeness. After user approval, the output of this phase is a system that is a functional part of the user organization.

The next phase of the software life cycle is an on-going phase. It is in operation as long as the system is a viable part of the organization. The Operation phase is divided into four subphases. Each subphase may or may not be initiated, and does not have to be initiated in the order in which it is listed. Also, once a subphase is initiated, it may never be absolutely completed, or may be initiated again. The four subphases of Operation are the system audit, performance measurement, maintenance, and modification.

A systems audit is initiated by the receipt of an audit request from the users or from operations personnel, or as part of a standard audit. An audit will determine if the implemented and perhaps modified system still satisfies the specifications. Auditors study records of any changes made to the system, the output of the system, and the original specifications. The audit process included verifying that modules are what they were designed to be, validating the results output by the system, and confirming that the system solves the problem in accordance with its specifications. The output of this phase is an audit report that includes any recommendations that the auditors may have for system modification. [BERS80]
Performance measurement is a subphase that is initiated regularly. Its objective is to study the resource use and the efficiency of an implemented system. By using system statistics as a basis, those measuring can see how well a system is operating within its resource-use constraints. Benchmark programs could be used as a basis of comparison. Performance measurement includes the evaluation of execution speed; size; throughput; data transfer rate; compilation, assembly, and generation speed; overhead; and response time. After measurement, a statistics summary is produced along with recommendations for efficiency upgrades. [GUNT78]

A fine line exists separating the final two subphases of the operation phase. Maintenance is defined as making changes to the software to make it run and fixing faulty information. The phase is initiated through user maintenance requests or error reports from the operations staff. During maintenance, the faulty code is corrected; all design documents, source code, and object code are updated; and changes are recorded in an maintenance log.

Modification involves changing the software to add enhancements and improvements to an implemented system. This subphase is also initiated by users who have a modification request or by recognition of an improvement opportunity by the data processing staff. An analysis of the requested modification takes place, then the new code is added. Again, all design documents and code must be updated, and the addition recorded in a modification log.

While other life cycles may be divided differently, the
basic operations of each phase of the cycles should be equivalent with the one presented above. The next section will present some representative software engineering techniques for the Systems Analysis, Design, and Development phases of the life cycle.
III. TECHNIQUES FOR SOFTWARE CREATION

Software Engineering techniques have been developed for each phase of the software life cycle in order to provide a disciplined approach to and checkable output from each phase. In the following sections, some of the major techniques or conceptual approaches for each phase will be discussed, with attention given to who developed the techniques, their stage of development, goals, methods, and applications.

A. REQUIREMENTS ANALYSIS AND SPECIFICATION

1. SADT. Structured Analysis and Design Technique (SADT) was developed by Douglas T. Ross and associates at SofTech. It has been in use since 1974 by several organizations, so is a fairly well-known software engineering technique. "It is a general-purpose modeling technique that is applicable to a wide range of problems, not just computer applications."[WASS80a] SADT has processes for Requirements Analysis and Specification, plus design. It also has a process for applying the techniques to systems development.

The goals of SADT are the following:

1. to provide a method for thinking in a structured way about large and complex problems.

2. to implement the concept of working as a team with effective division and coordination of effort and roles.

3. to communicate interview, analysis, and design results in clear, precise notation.

4. to document current results and decisions in a way which provides a complete audit history.
5. to control accuracy, completeness, and quality through frequent review and approval.

6. to plan, manage, and assess the progress of the team effort.[ROSS80b]

"A SADT model is an organized sequence of diagrams, each with concise supporting text."[ROSS80b] Each diagram is made up of boxes and arrows, with boxes representing parts of the whole that the diagram is defining, and the arrows representing interfaces between the parts. Each model is a series of diagrams in a hierarchical organization, each diagram gradually exposing more detail of the system. The diagrams can be used for both functional and data decomposition of a system.¹

Along with the SADT graphical techniques, division of the personnel involved in project development is also spelled out. Divisions include:

1. authors personnel who study requirements and constraints, analyze system functions and represent them by models based on SADT diagrams.

2. commentors usually authors, who must review and comment in writing on the work of the other authors.

3. readers personnel who read SADT diagrams for information but are not expected to make written comments.

4. experts persons from whom authors obtain specialized information about requirements and constraints by means of interviews.

¹Refer to figure 1 in Appendix B for a SADT diagram.
5. technical committee

A group of senior technical personnel assigned to review the analysis of every major level of decomposition. They either resolve technical issues or recommend a decision to the project management.

6. project librarian

A person assigned the responsibility of maintaining a centralized file of all project documents, making copies, distributing reader kits, keeping records, etc.

7. project manager

The member of the project who has the final technical responsibility for carrying out the systems analysis and design.

8. monitor

(or chief analyst) person fluent in SADT who assists and advises project personnel in the use and application of SADT.

9. instructor

A person fluent in SADT, who trains authors and commentors using SADT for the first time.[ROSS80b]

The division of the development personnel ensures that each activity is carried out by people who are specially trained in that area. Through interviews with experts, authors review the primitive needs and context of the proposed system and begin to produce SADT diagrams that specify major system functions. The commentors will review these diagrams and take them to senior technical or management personnel to be approved. Approved documents are passed on to the design phase, where SADT diagrams are further refined.

The use of SADT encourages effective and documented communication between all members of the development team, and also enforces many checkpoints in system specification. The graphical representations of the system can be easier for users
to understand than the narrative descriptions. These checkpoints, along with the graphical documentation, provide a disciplined approach to Systems Analysis.

The feeling of in-house development companies today is that the formality of the SADT process is only justified for larger projects, but the concept of well-defined models is important to any size project.

Structured Analysis and System Specification is a software engineering technique developed by Tom DeMarco, working for Yourdon, Inc. It provides tools to be used in analysis and a process to be followed in order to produce a structured specification document to be passed to the design phase.

"Structured Analysis is the use of these tools: Data Flow Diagrams, Data Dictionary, Structured English, Decision Tables, [and] Decision Trees to build . . . the Structured Specification." [DEMA79a]

"The principal goal of Structured Analysis is to minimize the probability of critical analysis phase error. The tools of Structured Analysis are defensive means to cope with the most critical risk areas of analysis." [DEMA79a]

The first tool, the Data Flow Diagram (DFD), is a network representation of an automated, manual, or combination system. Circles, or bubbles, represent component pieces of the system, while arrows with supporting text show the interfaces between the components.¹ A Data Flow Diagram is very similar to a SADT

¹Refer to figure 2 in Appendix B for a Data Flow Diagram.
After completing the seven studies of structured analysis, concise descriptions of the bottom level bubbles on the New Physical Data Flow Diagrams can be written as specifications for the new system. These descriptions are called "mini-specs" by DeMarco. "Each mini-spec must describe rules governing the transformation of data flows arriving at the [low-level bubble] into data flows leaving it."[DEMA79a]

The third tool, Structured English, is used in specification writing. Structured English is a subset of English, using only a limited vocabulary and specified syntax. Structured English uses imperative English verbs, terms identified in the Data Dictionary, and certain reserved words for logic formulation in simple declarative sentences, closed-end decision constructs, or closed-end repetition constructs. Structured English makes specifications concise, precise, and readable. It can be written quickly and comes more naturally than some automated languages.

Along with Structured English, Decision Tables or their graphic representations, Decision Trees, can be used to make specifications clearer. A Decision Table is most useful in explaining a selection to be made from a group of alternatives that depends on a combination of conditions.

After completion of the seven studies involved in Structured Analysis, and the writing of the Structured Specification, the system is ready to be moved after approval to the Design phase. An updated version of DeMarco's Structured Analysis and System Specification entitled Structured Systems Analysis has been published by Chris Gane and Trish Sarson. Most of the changes to
the process have been cosmetic, such as changing the shapes and notation on Data Flow Diagrams. Because of the great similarity, this technique will not be discussed in this paper.[DEMA79a]

3. **PSL/PSA.** PSL/PSA stands for Problem Statement Language/Problem Statement Analyzer and is a tool for the description and analysis of system definitions. "PSL/PSA was developed by Professor Daniel Teichroew and has been used by a limited number of organizations for approximately five years on a range of projects."[FREE80b] PSL/PSA is basically a computer-aided documentation tool that assists in the communication of Requirements Analysis and System Specification to the non-user. It is not intended to provide an approach to the analysis of the problem. It is thought that PSL/PSA should be used along with another formal technique, thereby enhancing that technique's analysis directives.

The goals of PSL/PSA are the following:

1. To record the results of each activity in the system development process in computer accessible form as they are produced.

2. To maintain all basic data about the system on a computerized data base.

3. To use the computer to produce hard copy documentation when required.[TEIC80]

PSL is a language used to describe proposed systems. Its objective is to express the information generally produced from the Systems Analysis phase in a syntactically analyzable form.¹

PSA is a software package used to record descriptions in the documentation data base, modify such descriptions, perform on

¹Refer to figure 3 in Appendix B for a PSL example.
demand analysis, and report production. The four different types of reports produced by PSA are data base modification reports, reference reports, summary reports, and analysis reports. Data base modification reports constitute a log of all changes made to the documentation. Reference reports list the documentation on the data base in a number of user-specified formats. Summary reports present summaries of a number of different relationships in the documentation. Analysis reports can provide various types of analyses of the information on the data base. Similarities of inputs and outputs, gaps in the information flow, and unused data objects can be detected. [TEIC80]

PSL/PSA can only reach its full potential when it is used with well-defined Systems Analysis techniques. It can provide much information about a systems definition on demand, and should be considered a valuable tool in user communication.

B. ARCHITECTURAL DESIGN

1. Modularization. In architectural design, the underlying structure of the problem to be solved is sought out and the general structure of the system is decided upon.

The concept of modularization is very important as the structure of the system is formalized. The system must be broken down into smaller tasks, so that these tasks can be designed and programmed. Systems can be broken into modules using many varied criteria. The traditional method used of decomposition is having each module accomplish one flowchart function. "But in many systems there is such a wide range of functions that the resulting modules vary greatly in size and complexity and thus lose some of their pragmatic usefuleness." [FREE75]
D. L. Parnas, a professor at Carnegie-Mellon University, has studied modular decomposition and has suggested a strategy to follow. The decomposition is made by encompassing one design decision in one module, such as the representation of a data structure or an algorithm for searching a table. Modularization should begin "with a list of difficult design decisions or design decisions which are likely to change."[PARN80] Then, each module can be designed around a decision, and be designed to hide its particular decision from the other modules. This way, if a design decision does change, it will only affect one module. Upon implementation, subroutines and functions should be allowed to be assembled in various combinations of code for modules.

This concept provides a strategy to complete the formalization of the system structure.

2. **Structured Design.** Structured Design is a concept and practice developed by Larry Constantine and Edward Yourdon. Its development was inspired by the recognition that the use of structured programming was not solving all data processing problems. Structured Design has been around for over ten years and its concepts have been implemented in many organizations.

"Structured Design is the [process] of designing the components of a system and the interrelationship between those components in the best possible way."[YOUR75] The goals of Structured Design for software are the following:

1. **efficiency** programs that make optimal use of scarce resources.

2. **reliability** programs that achieve required results when needed.
3. maintainability: programs that make the discovery, analysis, redesign, and correction of "residual bugs" easier.

4. modifiability: programs that are easier to keep viable in a changing environment.

5. flexibility: programs that perform variations of their functions without modification.

6. generality: programs with greater scope or range of function.

7. utility: programs that are useful and easy to use. [YOUR75]

Structured Design emphasizes the use of simplicity as a criterion for evaluating alternative designs. Simplicity can be achieved by dividing a system into modules. Two factors relating to modules must be considered when the system is being divided: coupling and cohesion.

"Coupling is the measure of strength of association by a connection of one module to another." [STEV80] Strong coupling means that a module is more difficult to change independently from other modules. The system should be divided into modules that control the level of coupling.

One way to control coupling is to increase the cohesiveness of the elements within a module. If the relationships between the elements in a module are maximized, the number of relationships an element has outside of its module will be reduced.
Structured Design uses a "structure chart" to graphically represent the module division of a system. These charts show hierarchical relationships between the modules without showing decision blocks. Structure charts can be constructed according to two different concepts for easy conversion to code and procedures. The first concept reduces a module to concern only input-process-output. The second concept is transaction structuring. Modules are considered as transformers for events, records, or inputs.\[STEV80]\]

After structure charts are drawn, they are checked and refined. The structure of the charts must match the structure of the problem. This implies that a change to a single part of the problem results in a change to only a single module. The scope of control for each module includes that module and all modules that are subordinate to it. The scope of effect of a decision is the set of all modules containing code whose execution will depend upon the outcome of the decision. A system is simple (and therefore of better design) when the scope of effect of a decision is within the scope of control of the module in which the decision is made. Structure charts must be checked for scope of effect violations. They must also be checked for optimum size, readability, and understandability. It must be determined if any modules need initialization and how it will be handled. Duplicate modules accomplishing the same function should be eliminated, and all dependencies on a particular data type, record layout, or index-structure should be isolated into

\[1\]Refer to figure 4 in Appendix B for a structure chart.
one or a minimum number of modules. Reduction of the number of parameters and amount of information passed between modules should also be a refinement goal. [STEV80]

It is suggested after completion of structure chart refinement that HIPO charts, discussed below, be constructed. These charts provide very useful information for detailed design.

3. HIPO. Hierarchy plus Input-Process-Output charts are a technique used when developing a system using a "top-down" strategy. HIPO was originally developed by the IBM Corporation as a documentation tool. There are two types of charts. The first is a hierarchy chart, and the second is generally a set of charts identifying each function in the hierarchy with respect to its inputs and outputs.1

HIPO charts can be very useful before, during, and after structured design. The drawing of HIPO charts causes the designer to identify the functions of the system on three levels. First is the system level that provides a conceptual view of the entire application. At the program level, the second level on the HIPO chart, the highest level of the segmenting of the system is shown. Each block may correspond to a task that must be initiated by the end-user when using the system. The module level is the level at which detailed program design takes place. "At the module level of the hierarchy, the design is sufficiently detailed that program code can be written directly from the design." [STAY80] This refinement at the module level usually occurs in the Detailed Design phase.

1Refer to figure 5 in Appendix B for a HIPO chart.
The benefits of using HIPO charts with Structured Design are numerous. User understanding and agreement on the functional content of the system are made easier, and missing or inconsistent information is identified earlier in the process. Modification and maintenance are easier because of increased understanding of every level of the system, better documentation, and simpler interfaces between modules.[STAY80]

C. DETAILED DESIGN

1. PDL. Detailed Design is the final step in software creation before code production. Precise designs for program modules are formalized. Program Design Language (PDL) was designed as a tool for the presentation of formal designs for coding. PDL also has associated with it a computerized processor. PDL was developed by Stephen H. Caine and E. Kent Gordon, and has been in extensive use since 1972.

PDL is a "pidgin" language and can be considered as a form of structured English. The language is somewhat formalized as it acts as input to its associated processor. Control information plus designs for procedures written in PDL act as input to the processor and the output is a working design document. The level of detail incorporated into each design is dependent upon the input to the processor. A very high-level design can be output first, and as it is easily understood by others, corrections and criticisms can be made. This level of design can be worked on until it is stable, then the next level can be started. Design decisions at every level will affect smaller and smaller segments of the system.
"A design produced in PDL consists of a number of 'flow segments,' each corresponding roughly to a procedure in the final implementation."[CAIN80] The statements that compose a flow segment, written in the formalized structured English, are entered free format into the processor.\(^1\) The processor underlines keywords, indents according to nesting levels and provides automatic continuation from line to line. A "text segment" can be used to make any comments to provide additional information. After all flow and text segments have been entered, the PDL processor outputs the following:

1. a cover page giving the design title, data, and processor identification.
2. a table of contents
3. the body of the design, consisting of flow segments and text segments.
4. a "reference tree" showing how segment references are nested.
5. a cross-reference listing showing the page and line number at which each segment is referenced.[CAIN80]

The two primary constructs used when writing PDL statements for flow segments are the IF and DO constructs. Using these two constructs, each flow segment can be directly translated into a program code construct.

The developers of PDL hope to improve its capabilities by allowing integration of data designs into the primarily procedural designs being produced at present. Also, PDL is a batch-oriented tool. The ability to code and modify on-line is a goal of future PDL releases.[CAIN80]

\(^1\)Refer to figure 6 in appendix B for a PDL segment.
2. **Nassi-Schneiderman Charts.** I. Nassi and B. Schneiderman introduced Nassi-Schneiderman structured flowcharts (N-S charts) in 1973. They are used to express and verify systems' design documentation. They also provide exact detailed design documents.\(^1\)

The goals of N-S charts are as follows:

1. Make the scope of program iteration well-defined and visible.
2. Make the scope of IFTHENELSE clauses well-defined and visible; also ensure that conditions or processes embedded with compound conditions can be seen easily from the diagram.
3. Make the scope of local and global variables immediately obvious.
4. Make arbitrary transfers of control impossible.
5. Enable the expression of complete thought structures on one page.
6. Enable the trivial expression of recursion.
7. Make charts adaptable to the peculiarities of the system or language for which they are used.

N-S charts use special symbols for processes, decisions, and DO constructs.

There are three basic uses for N-S charts. First, they are used to create logic design. The logic is designed while charts are being constructed, and the constraints on N-S charts such as one-page structure and no branch symbols force the development of a structured design.

Secondly, N-S charts provide a guide for programming and

\(^1\)Refer to figure 7 in Appendix B for a Nassi-Schneiderman chart.
testing. Their third function is to serve as program documentation.

N-S charts visually depict a model of the designed module. They force structured design while providing an excellent documentation tool.[YODE80]

3. Jackson Methodology. The Jackson Methodology was developed by M. A. Jackson around 1975. It was intended to be a more "constructive" method of detailed design. Jackson defines a constructive method to be a method "decomposed into distinct steps, and correct execution of each step must assure correct execution of the whole method and thus the correctness of its product."[JACK80]

Jackson's design methodology consists of three phases. During the first phase, the structure of the data is described. Each file, input, or output has its structure represented in tree notation. The second step is to arrange the identified data structures into a tree-like representation of the program structure. The program structure has the following properties:

1. It is related quite formally to each of the data structures. The tree for each data structure is directly imposed on the program structure.

2. The correspondences between input to output are determined by the problem statement. For example, one card file produces one report, while one group within the card file produces one detail line of the report.

3. The structure is vacuous, in the sense that it contains no executable statements.[JACK80]

"The third step in applying the method is to list the executable operations required and to allocate each to its right place in the program structure."[JACK80] Most of the operations
specified will have data objects for operands. From this program structure, a program written in any procedural language is easily written. The success of this method depends on the correct identification of the data structures and their fit into the program structure.

The output from this method is a list of operations in a program structure that fits the data. This detailed design is readily converted into code during the development stage.

D. DEVELOPMENT

1. Structured Coding. "The term 'structured programming' has for many years been used to connote a discipline for part of the design task together with a coding discipline."[DEMA79b] The term "structured coding" will be used to denote the coding alone.

Structured coding had its beginning in the late 1950s and early 1960s. It was during this time that the concern for the high use of limited CPU time in programming was brought forth for solution. Since most of the CPU time spent in program development was spent in debugging, it was thought that a way to prove program correctness before running the program was the answer.

At the time, there were no algorithms for proving program correctness, or any basic idea of how such an algorithm should work. A starting point was seen to be the avoidance of some of the more complex and difficult logic formulations, such as the ALTER, GOTO, SKIP, and REMOTE EXECUTE commands.[DEMA79b]

In 1966, a paper by Bohm and Jacopini introduced the concept that stated that any program could be written using only
process blocks plus two logical constructs. Programming using this format came to be known as structured coding.

The principles of structured coding are the following:

1. to write code with control (branching) formulations limited to the Bohm and Jacopini constructs (see figure B-8a) plus optional extensions (see figure B-8b).¹

2. to allow nesting.

3. to prohibit all other logical constructs.[DEMA79b]

The basic rationale behind structured coding is improved readability of code. This will lead to code that is more easily understood, maintained, and modified.

Also included in structured coding is the idea that all code should be reviewed by someone other than the developer. Code should be checked for both functional integrity and for adherence to the constraints of using only the specified allowable constructs.[DEMA79b]

2. Structured Walkthroughs. A structured walkthrough is "a peer group review of any [software] product."[YOUR77] The concept was developed by Edward Yourdon and associates to improve the quality of computer programs. Walkthroughs can help an organization find more program bugs in a faster and more economical way. Walkthroughs can also help find gross inefficiencies, and identify coding techniques that could detract from the programs' maintainability and modifiability.

There are six stages in coding and testing when walkthroughs can take place.

¹Refer to figures 8a and 8b in Appendix B.
1. before the code is keypunched.
2. after the code is keypunched, but before it is compiled.
3. after the first compilation or assembly.
4. after the first "clean" compilation.
5. after the first test case has been executed successfully.
6. after the programmer thinks that all test cases have been executed successfully. [YOUR77]

Walkthroughs are most productive when they take place in the middle of development. If done too early, reviewers tend to feel the time spent is wasted because programs are hard to understand before they are coded, and uncompiled programs do not convey much information. If walkthroughs first take place when the programmer feels that all testing is complete, any criticisms or suggestions for change will be taken as a personal attack or will require too much work to make the change beneficial.

"A successful walkthrough involves several people, each of whom plays a definite role." [YOUR77] The same person need not play the same role in every walkthrough, nor is a person limited to only one role during one walkthrough. It is important that each role is properly played in every walkthrough.

1. The presenter

   his task is to present the product. Usually the presenter is the producer or author of the product being reviewed.

---

1Keypunching is usually replaced by entry on a CRT.
2. The coordinator ensures that the activities of the walkthrough are properly planned and organized. During the walkthrough, the coordinator acts as the mediator. It is suggested that the coordinator be a "senior" member of the review team.

3. the secretary/scribe takes notes during the walkthrough to serve as permanent record of its results. The scribe should be someone who is a member of the programming team, but not be counted as a reviewer since he will be too busy taking notes.

4. The maintenance oracle is an actual reviewer of the product who reviews from the viewpoint of future maintenance. The maintenance oracle is usually someone who does not work in the same group as the developer, such as in the quality assurance group or maintenance group.

5. The standards bearer reviews the product with primary concern for the adherence to standards. The standards bearer acts as defender of all organization standards, and can be the same person as the maintenance oracle.

6. The user representative is someone who can ensure that the product meets the customer's needs. The user representative is especially important in a test walkthrough.

7. Other reviewers are one or more additional reviewers who give a "general" opinion of the correctness and quality of the product.
The producer initiates a walkthrough of his product. His first responsibility is to announce his intention of having a walkthrough at least two days in advance of the date he has selected. He next must choose a coordinator and participants for his walkthrough. "The producer is also responsible for providing appropriate documents for the walkthrough: listing, HIPO diagrams, specifications, etc."[YOUR77] The final responsibility of the producer is to choose a product that can be reviewed in 30-60 minutes.

Before a walkthrough takes place, the coordinator must select a place and an exact time for the event. This involves contacting all the participants so that a convenient time and place for all is arranged. Reviewing and distributing product documentation is also the coordinator's responsibility.

The actual walkthrough begins with the coordinator calling the group to order. The presenter then takes the floor and begins to present the product, piece by piece, making certain that every piece is reviewed. The reviewers' responsibility is to provide constructive criticisms, comments, and suggestions as each piece is presented. The reviewers should provide the presenter with a list of errors that need no explanation in order to save time. The presenter should avoid arguing in defense of the design or coding as this can lead to ego confrontations. "A basic principle to keep in mind is: The basic purpose of the walkthrough is error DETECTION, not error CORRECTION."[YOUR77]

At the end of the walkthrough, the group of reviewers is asked for its recommendations. Usually the recommendation is one of three choices: to accept the product as is; to accept the
product with the revisions suggested in the walkthrough; or to suggest that another walkthrough is necessary after revisions are made. [YOUR77]

In the next section, issues concerning the management of software creation and maintenance are presented.
IV. MANAGEMENT ISSUES

A. PROJECT MANAGEMENT METHODOLOGIES

Numerous methodologies have been developed and published for the management of the entire software life cycle. These methodologies break software creation and operation into phases, assign specific tasks to be completed and documents produced in each phase, and provide specific review checkpoints and procedures to ensure that each phase has been carried out completely. Some organizations choose to develop their own project management methodologies from combinations of the software engineering techniques presented above. Methodologies developed in-house fit the management needs and concepts of an organization very closely. There are also several methodologies available for purchase. The methodology packages usually provide training, formal definition of phases, and example documentation forms. While some modification is usually required before a methodology is implemented, it can provide a very good basis for management of applications software production.

In the following sections, the phaseology, documentation standards, and provisions for project review of three published methodologies will be discussed. The first will be Method-I published by Arthur Andersen & Company in Chicago, Illinois. Second will be the PRIDE methodology developed by M. Bryce & Associates in Cincinnati, Ohio. The final methodology will be one published by the IBM Corporation entitled Managing the Applications Development Process or MADP.
1. **Method-I.** Method-I is published and sold by Arthur Andersen & Company and has been copyrighted since 1980. It is the only complete methodology based on the Warnier-Orr structured design and programming technique. The package from Arthur Andersen includes training materials, classes, help in modifying the method to fit the specific installation, and consulting assistance on the first few projects developed using Method-I.

The basis of Method-I is the Warnier-Orr technique. This technique was not discussed before because of its relevance to this section of the report. Some of the major Warnier-Orr concepts are explained below.

1. Warnier-Orr is based on the principle that the structure of a program should be identical to the structure of the data.

2. Since Warnier-Orr takes "time-cycles" into account early in the design process, there is no difficult translation from user design to programming specifications.

3. Warnier-Orr notation is easy to understand and with certain software packages can be produced on a conventional printer.

4. Warnier-Orr program specifications result in virtually identical programs when written by different programmers.

5. Warnier-Orr has very sound, set theory mathematics underpinning its concepts.

6. Efforts are underway to develop a software package to generate COBOL code directly from computerized Warnier-Orr diagrams.[HEIK81]¹

Method-I divides the development process into four major phases. The phases are Systems Planning, Preliminary System

¹Refer to Figure 9 in Appendix B for a Warnier-Orr diagram.
Design, Systems installation, and Production Systems Support. These major phases are subdivided into 37 major segments and 140 tasks. "Each major segment has an overview which contains its objectives; the approach used to completing the work; a discussion of the major products and major inputs for the segment; advice regarding project control; relationship with other work; staffing considerations; and general guidelines." [HEIK81] The tasks are subdivisions of the phases, and are described with respect to objectives, a discussion, development steps, and documentation requirements.

Method-I provides for six different types of documentation: Administrative, data, management, resources, technical and user-oriented documentation. Method-I documentation uses only a few basic forms, each of which has several uses.

In the Systems Planning phase, planning charts showing the overall relationships between phases, segments, and tasks are completed. Work to be done is planned, combining and scheduling each task and segment. Estimating guidelines are provided by Method-I for planning project resource requirements. Also provided are project control procedures, which state how to initiate projects, set up work programs, control changes to the system, track project team efforts, report time, and maintain control over work papers and project files. As a review step to this phase, quality assurance check lists are used to verify the completion of all tasks of the phase.

The Preliminary Systems Design phase is broken down into functional design and technical design. Also included are evaluation of application software, hardware and system software
direction, cost/benefit analysis, hardware and software selection, development of an installation schedule, and preparation and presentation of reports to management. The major files, reports, and screens will be designed in this phase and all major design points and management policies will be decided upon. Detailed programming development does not take place in this phase.

Systems Installation includes programming, testing, conversion and post-conversion work. The documentation of these steps is made as additions to the documents produced in the Preliminary Systems Design phase. The steps of this phase are refinements of the details produced in Preliminary Design.

Method-I provides for the management of maintenance programming in the Production Systems Support phase.

Various sections discuss how to monitor production systems, how to respond to emergencies; how to report performance of the maintenance group; how to initiate systems investigation requests; how to log, analyze, classify, and acknowledge user requests; how to maintain information about the work back log; how to establish priorities with users; how to communicate the status of current requests, and how to plan future work. [HEIK81]

Method-I also addresses rapidly changing technology with procedures regarding prototyping, data modeling and data base design, on-line systems requirements, office automation, distributed processing and networking, and, most importantly, the purchasing of applications software packages. The procedures in Method-I include screening, selecting, confirming and installing purchased software packages.

"Used as a guide, not as a rigid recipe or 'paper generator', Method-I wil help any competent staff plan, control,
design, implement and maintain automated business systems with great success."[HEIK81]

2. PRIDE. The PRIDE methodology was written and published by M. Bryce & Associates, copyrighted in 1974. Its package contains many of the same items as the Method-I package contains, such as classes and training materials. The development process it describes uses phases of development, activities for each phase, and output in the form of reports from each phase.


Phase 1 of the development process is the System Study and Evaluation. There are eight activities that are completed in this phase, and each activity corresponds to a section of the System Study and Evaluation Report, which is the output of this phase. Two of the eight activities are review steps, where the results of the preceding activities can be checked and approved. The eight activities of System Study and Evaluation are:

1. Develop preliminary scope of project
2. Analyze current systems
3. Survey information needs
4. Prepare information requirements and project scope
5. Review information requirements
6. Develop system approach and feasibility
7. Prepare system evaluation

8. Review system approach and evaluation [BRYC74]

The System Study and Evaluation Report will be approved and signed by both developers and users and become part of the system's permanent documentation.

System Design is Phase 2 of the methodology. There are five activities in this phase with the last activity being the review step. The first activity is very important, because it is here that a system is decomposed into sub-systems. After this is completed, a system flowchart must be prepared along with logic and data management descriptions. The third activity is the preparation of illustrative output. This output is subject to review and approval. A system design evaluation must be prepared, and it is reviewed to complete this phase. The report output of Phase 2 is a System Design Manual.

Sub-System Design is Phase 3 of the methodology, and encompasses the refinement of the overall system design prepared in Phase 2. The four activities of this phase are the definition of sub-system processing by procedure; the preparation of sub-system flowcharts, procedure logic, and the finalization of data management descriptions and output reports; the preparation of a sub-system design evaluation; and the review of the sub-system approach and evaluation. The report output of this phase is a more detailed design manual, the Sub-System Design Manual.

The fourth phase of the PRIDE methodology is divided into two subphases. The Administrativie Procedure Design subphase is usually handled by analysts working with the users to design new procedures for personnel to follow. There are four steps to be
followed in developing new procedure. First, the procedural steps must be defined. New forms are designed in the next step. The next two steps are the actual preparation of the procedure and its review and approval. An Administrative Procedure Manual is prepared to document the completion of this subphase.

The second subphase of Phase 4 is the Computer Procedure Design. This is where programmers can become involved in the methodology. In this subphase, six activities are completed.

1. Define computer procedure into program steps
2. Design detail logic for program preparation
3. Prepare file layouts from file descriptions
4. Evaluate procedure for operating requirements
5. Prepare computer operating procedure
6. Perform technical review and issue program specifications

Along with program specifications, a Computer Run Book containing general design details is prepared.

Phase 5 of PRIDE is the Program Design phase. The first of the six activities included in this phase is a review of program specifications and file descriptions. This activity is not only a good check on these documents, but also serves to familiarize programmers who have not been involved with the system with the overall design of the system they will be programming. The next two activities are the preparation of block diagrams and source code. Next, program test data is developed and each program is tested and debugged. Each program is reviewed for procedure conformity. The report output of this phase is program documentation.
Phase 6 is the Computer Procedure Test. Groups of programs or subroutines making up a complete procedure are tested together. The three activities of this phase are preparation of the procedure test data, the testing and debugging of program steps, and the review of each computer procedure for sub-system conformity. Results of testing are written as additions to the Computer Run Book.

The System Test is Phase 7 of PRIDE. This phase involves the installation of the system and education of the users. Any required data base is installed at this time. Data for an entire system test is developed and a parallel test is conducted. The test results are reviewed and any additional comments are made to the System Design Manual.

After implementation of a system, there are two phases remaining. They are System Operations and System Audit. In the operations phase, system modification/improvement forms are prepared. According to the amount of change to be completed, certain of the first six phases of PRIDE will be performed again. The modification/improvement request will be reviewed and the modification performed upon approval. The System Audit involves the study of the system and preparation of a System Evaluation Report.

The PRIDE methodology does not address directly the purchasing of software packages, but certain phases can be selected for completion depending on the needs of the organization when software is purchased. [BRYC74]

3. MADP. "Managing the Application Development Process" is a methodology for both technical and management aspects of
software creation and operation. It was developed and published by the IBM Corporation, and was tested by the IBM data processing department. The MADP package includes training material and classes.

In MADP's phased development approach, Design and Implementation are the two major phases of development. The Design phase is further divided into three subphases: Requirements Definition, External Design, and Internal Design. The Implementation phase also has three subphases: Program Development, System Test, and Installation and Maintenance. The activities and output for each subphase are formalized in the MADP methodology.

"The major objective of the requirements definition phase is to establish and obtain formal agreement with users of the business functions to be included and excluded in the new system, and any constraints under which the system must be designed."

[IBM77] The outputs of this phase are as follows:

1. A list of the business functions included and excluded
2. Overall information requirements
3. Interfaces to other systems
4. Performance objectives
5. Audit and control requirements
6. A plan for external design

Overall information requirements include processing flow and volume of inputs and outputs. Interfaces refer to the information flow between the system and other departments in the organization. The above outputs of this phase are documented in
a Systems Requirements Report, which describes "what" a system will do, but not "how" it will do it.

After the requirements are completed and approved, the external design phase is begun. The two major objectives of external design are to provide an overall picture of the system from the users' point of view, and to validate that the defined system still meets the business requirements.

The parts of the documentation to be included from External Design are:
1. Functional information flow
2. Transaction and report formats and specifications
3. A plan for internal design

The functional information flow describes the flow of data processing from user to computer and back to the user, the interfaces with other systems already in production, and the functions that are a part of processing. This documentation needs to be completed and approved before moving on to Internal Design.

The major objective of Internal Design is to develop a detailed blueprint for planning and estimation of implementation. The System Design Report output from this phase contains:
1. Program function specifications
2. Hardware/software configuration
3. Preliminary system test plan
4. Data file specifications
5. A plan for implementation

plus information from the External Design phase. Again, this documentation must be approved before moving on.
The next phase is Program Development. Its objective is "to build and test each component of the system and to ensure that the components fit together into an integrated system." The documentation and products from this phase include:

1. program specifications
2. tested, integrated programs
3. system test plan
4. system test cases and expected results
5. system demonstration plan
6. data conversion plan
7. user and operator manuals

This phase generally requires the most development efforts. Results of this phase are preliminary, and will be confirmed during System Test.

The objective of System Test is to subject the system to a variety of tests and stresses and to verify that the results and documentation meet design specifications. The products of this phase are:

1. an operable system
2. tested user and operator manuals
3. a plan for installation
4. file conversion programs
5. a training plan

The Installation phase objective is to begin to process live data in the operable system. Documentation is prepared that will help in future maintenance and modification. The following are products of the Installation phase:
1. the operational system
2. maintenance documentation and plan
3. satisfied users
4. trained users

The system is now operational; it performs as expected using live data.

MADP provides for efficient project management by dividing management activities into three different areas: pre-project activities, during project activities, and post-project activities.

"Pre-project activities consist of defining project objectives and conducting an initial systems assurance review." [IBM77] The pre-project documentation is reviewed for completeness and resource commitments are made. During project activities are usually iterative. Project plans are made, the project progress is tracked, project status is reported to top management, and reviews of the project are conducted to analyze project status and quality. Post-project activities entail the collection and organization of all the data produced by the development process. This data should be analyzed and the results used to keep the MADP methodology up to date.

B. IMPLEMENTATION OF SOFTWARE ENGINEERING TECHNIQUES

The use of modern software engineering techniques usually involves a great amount of change in an organization. The conversion to a structured, disciplined approach to software creation and operation can have a major impact on internal organizations and the people who work in them. Utilization of
software engineering techniques would be impossible without the support and active cooperation of the workers involved.

The following paragraphs will discuss the problems that can be encountered when attempting to implement a new method for software creation. Problems can be divided into three types: technical, psychological, and economic. Management needs to pay special attention to these problems in order to obtain full benefit from the implementation of these techniques.

1. Technical problems. It is important to use specific criteria in the selection of the methodology or technique to be implemented. Selection of the techniques should be based on organization goals and practices. It should be decided which would better fit the data processing organization—a comprehensive methodology, a combination of techniques and theory, or possibly both. In order for any methodology or technique to be effective, it must be customized to the particular organization and must be integrated with existing procedures.

It is also important to carefully select the project on which the methods will first be used. "The choice of project is particularly critical, since it will be the basis for evaluating the methodology, determining its effectiveness, and studying whether it should be applied to other projects."[WASS80b]

The selected project should lend itself well to being used as an experiment. It is usually best to select a pilot project to test the validity of the methodology. The pilot project should have several characteristics. First, it should be relatively small. Smaller, less complex projects are easier to measure and
monitor, and it may be possible to carry the project out twice, using one time as a control measure. Also, a new development project is much better as a pilot project than the modification of existing code. Code that has been produced using non-structured techniques is much harder to modify using a structured technique than the production of new, pure code.

2. Psychological problems. The principle psychological problem with the implementation of new techniques is the "innovation represents a personal threat to each member of the programming staff."[WASS88b] Many of the people involved in the change do not understand the concepts behind the new techniques and especially do not understand the motivation behind their implementation. Therefore, the first step in the implementation process should be the education of all staff members and managers, both in the specific technique and in background information about modern software engineering. Also, everyone should be made to feel like a part of the group effort to upgrade development standards.

The personal advantages of using the new procedures must be stressed. Many people have a fear of failure in a new environment. There are several ways to help employees overcome their fears. First, with methods that employ team organizations, it must be emphasized that teams offer the opportunity of shared responsibility for system development. Failure on an individual level is de-emphasized.

Second, personnel will accept the changes more readily if a proper working environment is provided for them. The great diversity of skills existing among personnel working with
software calls for the careful examination of job roles. If a person is assigned a job in which he thinks he can succeed, he will be happier with the job and with the techniques he is using.

The most important way to accomplish personnel acceptance of new procedures is to make people feel that they are making a significant contribution to the group effort to upgrade standards, and have a chance to grow professionally while making their contribution.

Another psychological problem that must be dealt with is getting the support of top management for the new techniques. Management support will not only help with acceptance of the techniques by technical employees, but can also result in increased funds for professional development and education.

3. Economic problems. The primary motivation for switching to software engineering techniques is usually economic in nature. Therefore, close monetary watch is kept on the first few projects developed using the techniques. There are some economic considerations to be kept in mind when testing the techniques.

First, it is often difficult to measure the savings caused by the use of a techniques in actual dollar figures. Many of the benefits of a structured approach to Systems Analysis are intangible and long term. At first glance, a new technique may seem to be decreasing productivity. This is generally because of the increased paper work required. These results are short term. Systems Analysis with a structured technique can lead to systems that have a better design and require decreased and simpler maintenance and modification.
Secondly, when evaluating the economic value of a methodology, improved project management must be taken into account. A structured approach provides management with a better basis for estimation of project completion time and tools to track the progress of system development. Again, this is often difficult to evaluate economically in dollar terms.[WASS80b]

Conversion strategy. A strategy for conversion to new techniques must be developed to help control all the technical, psychological, and economical considerations. One approach that has been outlined involves using a pilot project within the following guidelines.

First, team members that are selected to work on the pilot project must be receptive to the changed techniques. Only workers who are dedicated to the success of the project will give the techniques a good test.

Second, budgets and timetables should be established by traditional means before the pilot project is begun. These should be modifiable as new information becomes available.

Third, checkpoints should be established for review of work completed. Time division should be monitored so that it will be certain that the percentage of time targeted to be spent in analysis is not cut short. The completion date on the project should not be enforced strictly, because experimentation with new techniques is often needed.

Finally, programmers should be encouraged to represent their programs in structured notations before coding. Early coding should be strictly forbidden. One of the most important goals of implementing new techniques is the changing of
programmer habits. A new method of attack on every problem should result from the new techniques, and the restriction of early coding will help obtain this result.

Once a new method is introduced and in operation, it is important to remain in contact with current technology that may lead to the improvement of the techniques. An organization must also be prepared to modify and test all techniques that might lead to the optimal approach to software creation.

The next section presents the results of a study conducted of corporate data processing environments concerning their use of software engineering techniques.
V. SURVEY OF COMPANIES

A. INTRODUCTION

The use of existing software engineering techniques and methodologies is easily justified in theory. The question that constitutes the basis for the following study is this: Are any of the software engineering methods being used in industry at the present time? Along with this question there are several others to be answered. Why were specific methods chosen? Are companies aware of others? Can relationships be drawn between company characteristics and which methods are used? Also, how do companies feel data processing centers will be organized in the future and how can students be prepared for this future?

To answer these questions, it was decided to conduct a study of in-house development companies in Indiana and draw conclusions based on the data collected. Next, research objectives were established. The objectives are as follows:

1. Determine if selected companies are aware of the modern software engineering techniques, and if they are, how many and which ones are in use.

2. Study the details of the processes and procedures followed using the techniques for software creation.

3. Study the details of the choice of techniques, and find correlation between company characteristics and technique choice.

4. Determine how selected companies view the educational programs of universities for preparing potential employees in software creation.

Participants were solicited for the study through the help of Professor William F. Brown of Ball State University's Computer Science Program. Dr. Brown drafted a letter to
prospective participants and it was mailed during the last week in August, 1982.

Prospective participants were contacted and interviews were set up with those wishing to participate. The interviews took place over the period of September 24, 1982 to October 11, 1982. Each interview lasted from one to three hours.

The questions to be covered in the interviews were prepared according to the research objectives.\(^1\) The questionnaire was mailed to each participant in advance of the interview, so that answers could be researched or prepared. The questionnaire contained questions concerning general information, company characteristics, software life cycle and techniques, documentation and standards, and personnel education.

Each of the nine participant companies were guaranteed complete confidentiality before interviews took place. Therefore, no specific company names will be mentioned. With the promise of confidentiality, the companies were free to be completely open about the organization and data processing procedures.

After the nine interviews were conducted, the data was analyzed against the research objectives. The results will be presented in the following manner. The companies will be presented in groups by stage of development, with mention being made of user involvement, techniques, training, and future of each group. Then the influence of company characteristics such as company size, staff size, hardware size, and percentage of production software that was purchased will be analyzed.

\(^1\)Refer to Appendix C for interview questionnaire and letter.
B. Evaluation of Data From Study

The companies participating in this study are first grouped by development of software engineering awareness and usage. The criteria used for the breakdown include the number of techniques in use by an individual organization and the formality of the software creation process. Three groups of companies were formed ranging from lowest development to highest development. After details of company attitudes toward software engineering; actual techniques, processes, and training; and the organization's view of the future are presented, the influence of company characteristics on these details will be examined and analyzed.

In this section and all following sections, it is important to note that no company will be mentioned by name. Due to the confidentiality of the information used for this report, no exact figures will be given, but they were used as input to the evaluation.

1. Low Development Companies. The software development process in the companies of low development has little formality, and software engineering awareness is in its first stages. The following table presents the findings of the study.

<table>
<thead>
<tr>
<th>TABLE 1</th>
</tr>
</thead>
</table>

| LOW-DEVELOPMENT COMPANY ANALYSIS |

| Data processing organization |
| These companies have programmer/analysts who communicate directly with the user. |

| Planning and Control |
| Top management and data processing management share responsibility for justification and prioritization of major projects. The data processing management is responsible for time estimates and work assignments. |
TABLE 1 (CONT.)

User Involvement
These organizations have user-initiated new development and modifications, with a contact person in the user department for user input. A committee of user and data processing personnel clear new developments and modifications, but no formal sign-offs by the users are required until project is completed.

Initiation of Projects
Projects are initiated by the user or the data processing department, usually through the submission of special forms. There is no formal procedure for handling projects after they are initiated.

Methodology or Techniques
Organizations in the low-development classification have some informal use of structured programming. One organization has studied a published methodology for possible implementation.

Attitude Toward Techniques
All organizations in this classification feel that much emphasis on structured programming is a waste of productive time. Also, comprehensive methodologies are too expensive to implement or do not address the purchase of software directly. Cost/benefit justification cannot be made for software engineering techniques.

Training
There is some informal training in ANS Structured COBOL plus on-the-job training with emphasis on technical rather than analytical skills.

Standards
The organizations have basic programming standards for simplicity and documentation. Flowcharts and maintenance logs are required.

View of the Future
Organizations in this classification feel that data processing departments are moving toward the information center concept, with more "end-user" programming using fourth-generation, "user-friendly" languages.

Education of Potential Personnel
These companies feel that college graduates are too sophisticated or over-qualified for work in their organizations. In general, they feel that schools should keep in better touch with industry for a more realistic picture of the job roles.
While companies in this group are aware of software engineering techniques, they do not feel that they are justifiable in their organizations. Many of the techniques such as structured programming and structured analysis are used informally, but no standards for documentation or use exist in the organizations.

2. Medium Development Companies. The companies in the medium-development classification are aware of software engineering's benefits, and have begun to implement some techniques. Some have begun reorganization of the data processing department to get the full benefit of the techniques. The results of the study for medium-development companies follow.

| TABLE 2 |
| MEDIUM-DEVELOPMENT COMPANY ANALYSIS |

**Data Processing Organization**

Companies in this classification have programmer/analysts, but the functions of programming and analysis are more separated. Although there are no pure analysts or pure programmers, as employees gain more experience, more analysis than programming is part of the job. One company is moving towards complete separation of analysis and programming, with a team handling the analysis.

**Planning and Control**

There exists a wide variety of methods for planning and control in this classification. A steering committee is used by one organization to prioritize and schedule the projects. One organization handles control by the use of a project team, including management, users, and data processing personnel. Another organization uses the concept of "user-ownership" of projects. Users are responsible for the management and control of their own projects, with data processing personnel acting as members of the users' team.
User Involvement

As mentioned before, "user-ownership" in one organization is an example of complete user involvement. The other organizations have user sign-off on design documents and users as contacts during analysis.

Initiation of Projects

Projects are user initiated and given to management for approval. Formal analysis and feasibility studies are to be initiated after being tested on pilot projects. Formality exists in the sense that projects are documented more fully before coding than was done before formality was added.

Methodology or Techniques

Formal comprehensive methodologies have not been introduced in any of these organizations. One organization uses procedures that were written by employees and compiled into a standard guide for operation. These methods are reviewed annually and updated. Structured programming and documentation techniques are not stressed in this organization, but they are looking into implementing Structured Design. In another organization, the Yourdon method of Structured Design and Analysis has been tested on a pilot project and is now approved for implementation. Structured programming is stressed and guidelines set for programming techniques. This organization also utilizes structured walkthroughs, but only at one point in the cycle, just after coding. Another organization is following "user-ownership" of projects, and is beginning to assign its data processing personnel to user departments permanently. They do not have a methodology as users handle project management. The organization is trying to build personnel skills in fourth-generation languages, but do not require any kind of standardized or structured programming.

Attitude Toward Techniques

The organizations feel that while structured programming may be beneficial, time would be spent more wisely in structured analysis and design. The implementation of Structured Design will shift the allotment of time from the coding into more analysis and design. One organization feels that programmers are becoming obsolete, and therefore time should be devoted to training users in the use of "user-friendly" languages and the data processing personnel in support and consultant roles.