NAVSWC TR 90-216

THE REENGINEERING OF NAVY COMPUTER SYSTEMS

BY TAMRA MOORE AND KATHERINE GIBSON

UNDERWATER SYSTEMS DEPARTMENT

SEPTEMBER 1990

NAVAL SURFACE WARFARE CENTER

Dahlgren, Virginia 22448-5000 • Silver Spring, Maryland 20903-5000

NAVSWC TR 90-216

THE REENGINEERING OF NAVY COMPUTER SYSTEMS

BY TAMRA MOORE AND KATHERINE GIBSON
UNDERWATER SYSTEMS DEPARTMENT

SEPTEMBER 1990

NAVAL SURFACE WARFARE CENTER
Dahlgren, Virginia 22448-5000 • Silver Spring, Maryland 20903-5000

FOREWORD

Reengineering is the process of using information about existing computer systems to support the maintenance and development of new systems. During the months of October 1989 through April 1990, an analysis of the current state of reengineering was performed at the Naval Surface Warfare Center (White Oak) with an emphasis on how it applies to Navy computer systems. The Reengineering of Navy Computer Systems summarizes this analysis by defining the terminology used in this technology area and discussing the different capabilities of reengineering.

This analysis resulted in the identification of distinct capabilities in reengineering technology and concise definitions of these capabilities. Automated support was also examined by identifying commercial and research products that support the different capabilities of reengineering. Reverse engineering was selected from the capabilities for further analysis. Current and proposed techniques for performing reverse engineering were explored in detail.

The current state of reengineering was examined using a systems engineering approach, by observing how reengineering should be applied to all components of the system including software, hardware, and human interaction, and how reengineering affects each of these components. This analysis explored techniques for integrating methods for reengineering with the forward systems engineering environment. The analysis then concentrated on how reengineering applies to Navy tactical computer systems.

This report defines the different processes of reengineering, establishes guidelines for reverse engineering, and suggests automated products for supporting the reengineering processes. A thorough examination of all of the available reengineering products was not within the scope of this research, but has been designated for future examination at NAVSWC. Information gathered on the products available during this project is presented in the appendices of this report so that the reader can contact the affiliated agencies directly for specific technical information.

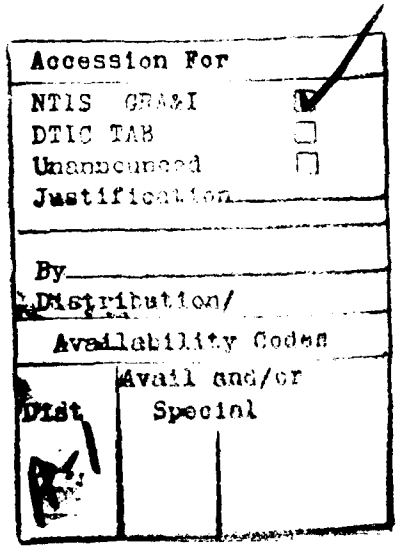
Reengineering has a potentially large payoff for a relatively low risk. A large amount of information is encapsulated in Navy tactical computer systems and the ability to use the knowledge stored in these systems would benefit the Navy. At a minimum, reengineering is used to establish a better understanding of computer systems and, at best, provides cost-effective support in both the maintenance of current computer systems and the development of future computer systems.

This research was primarily funded through the Independent Exploratory Development (IED) program at NAVSWC and transitioned to the Engineering of Complex Systems Program sponsored by the Office of Naval Technology (ONT).

The authors wish to thank Commander Jane Van Fossen (ONT) and Dr. Frankie Moore (NAVSWC) for their programmatic support; Steven Howell, John Maphis, and Phillip Q. Hwang for their technical support; and Adrien Meskin of Advanced Technology and Research Corporation for support in preparing this manuscript.

Approved by:

C.A. KALIVRETENOS, Deputy
Underwater Systems Department



CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

ILLUSTRATIONS

CHAPTER 1

REENGINEERING TECHNOLOGY

The technology of reengineering provides an environment for supporting the maintenance and development of computer systems. It defines formal techniques and new capabilities for using existing information to modify and enhance computer systems through abstraction. It supports system development through the isolation of reusable parts for migrating proven functionality and the implementation from existing systems to future systems.

Reengineering provides capabilities for solving many of the fundamental problems of maintaining and developing computer systems, while introducing new innovative ideas which will benefit the way systems are analyzed, designed, coded, tested, and maintained.

SYSTEMS REENGINEERING CAPABILITY TAXONOMY

A number of authors have provided definitions of varying levels of maturity for the software and system reengineering fields.[1,2] These definitions are very generic, making them applicable to any computing environment. For example, Reference 3 defined software reverse engineering as the "examination and alteration of an existing computer system to reconstitute it in a new form and the subsequent implementation of the new form." This definition does not specify what the alterations are, nor how they should be performed. It also does not define what the new form is, nor how this new form should be generated. This definition should be expanded to include additional information based on the specific computing environment in which reengineering will be applied. The generality of proposed reengineering taxonomies has caused confusion and has resulted in various interpretations of how reengineering is to be accomplished. A more comprehensive and concise taxonomy and perspective is needed to put various research, approaches, methods, tools, etc., in context.

The research performed at Naval Surface Warfare Center (NAVSWC) identified distinct capabilities and developed concise definitions for reengineering Navy tactical computer systems in the NAVSWC systems engineering environment. This environment addresses the maintenance and development of all components of the system including software, hardware, and human interaction. The capabilities include the following: reverse engineering, restructuring, retargeting, reuse, redocumentation, and engineering mental models.

Reverse Engineering

Reverse engineering is the process of analyzing a subject system to identify the system components and their interrelationships, and to create representations of the system in another form or at a higher level of abstraction. These forms include reports, tables, or matrices describing different aspects of the system, such as data structures, control structures, or module hierarchy. The abstractions are either high level designs themselves or are used with the other forms to create a high level design which can be updated with new system requirements, restructured for optimization, and even retargeted for new architectures. The high level design must be analyzable for consistency and completeness, and must be usable in the forward systems engineering process to generate a new implementation of the system.

Reverse engineering is performed on both software and hardware. Software is abstracted to design representations which are either textual such as a program design language (PDL), or graphical such as data flow diagrams. Hardware reverse engineering is commonly implemented by component replacement from existing systems; this is actually hardware reuse. Hardware configurations can be abstracted to design representations using a hardware description language such as the Very High-Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL). The VHDL description can be used to analyze modifications to the corresponding hardware much the same way software designs predict software behavior.

Restructuring

Restructuring is the transformation from one form of system representation to another at the same relative abstraction level while preserving the subject system's functional requirements.[3] The goal of restructuring is to *improve* the current implementation while maintaining and supporting current system requirements. The components are generally kept intact and the improvements can be based on dynamic and/or static criteria. Dynamic improvements are supporting nonfunctional requirement attributes including performance, reliability, fault-tolerance or security. Static improvements address physical structural factors such as introducing better systems engineering practices.

Systems restructuring is composed of the combinations of component restructuring. As an example, software restructuring can be applied to the system specification, high level design, or final source code; source code may be restructured to eliminate "dead code," and to improve overall software structure by eliminating "gotos" and combining data structures where possible. Hardware restructuring may be performed on the design or actual configuration of the architecture in order to improve performance and fault-tolerance. As an example, a restructuring may be performed on the interconnection network architecture to provide better communication connectivity configuration for the software without changing the existing computational components.

Retargeting

Retargeting normally refers to modifying system software to meet a new environment, such as new hardware. In a much broader sense, retargeting can include other modifications, in addition to software changes, which are necessary to meet

the new environment. Historically, software is considered easy to change and inexpensive when compared to hardware modifications, therefore many of the volatile processes performed by a computer system are performed in the software. Software is retargeted when new requirements that are introduced to the system are not met through hardware modification. Alternatively, software retargeting may be necessary when the existing hardware of a computer system is changed. For example, the changes required for the current software implementation to execute on the new hardware are already major resource drains when both systems are uniprocessor. The problem may grow exponentially when the new target architecture differs dramatically as in distributed or parallel processing. Furthermore, it may not be possible to meet new requirements by modifying the existing software, making it necessary to alter the hardware (e.g., introducing a special processor) via hardware retargeting.

Reuse

Reuse is the identification, isolation, categorization, and retrieval of system modules for use in future system development. These modules can be extracted from the existing system, classified and stored in a reusable format in a repository. Because of the vastly differing sets of applications in any complex system development, reuse is generally associated with well-scoped domain analysis. Reuse is generally considered to be one of the most beneficial activities associated with systems reengineering technology.

Reusable modules are concise software components, hardware components, or system components at the design or implementation level. Examples of these modules include requirements, design specifications, architectures, interconnects, programs, subroutines, hardware designs, devices, chips, boards, etc.

Repositories provide a means for classifying, storing and retrieving useful information that is represented in a reusable format. They are an indispensable and vital part of any systems engineering and component engineering environment. In fact, the success of these environments in terms of efficiency, quality, productivity, etc. may directly depend on the quality of the repositories that the environment contains or interfaces with. Major industry participants in the field of reuse have specified a standard format for repositories. Repository/MVS by IBM and CDD/Repository by Digital Equipment Corporation (DEC) are two examples of such contractor's repositories. Reuse technology is also a high priority for the prime of the Software Technology for Adaptable and Reliable Systems (STARS) environment work.

Redocumentation

Redocumentation is the creation or revision of a semantically equivalent representation. The resulting form of this representation is usually considered an alternative view intended for a human audience which describes the functions and behaviors of the system. Redocumentation is useful in generating users guides, specifications, and any other report that describes some aspect of the system at the highest level of abstraction possible. Redocumentation uses any available information about the system, including any existing system documentation, system specification, high level design, implementation, user knowledge, or developer knowledge in order to generate these reports. Reports, users guides, and

specifications that are obtained through redocumentation provide quick insight into the functionality and behavior of the system that would be difficult to obtain from examining the system itself.

Engineering Mental Models

Engineering mental models are models representing how the engineer(s) visualized the engineering process throughout the development of the existing system. These models are useful for capturing information about the engineering process used, such as design decisions or implementation structures. They provide a documented history of previous development issues and how requirement ambiguities were resolved. This information is captured and represented in a formal, repeatable and analyzable manner. The objective is to solve many of the current difficulties in performing maintenance on an existing system stemming from an inability to understand how the system was developed, and why certain design and implementation decisions were made. Procedures for determining these decisions and long hours of examining the requirements and resulting implementation are often repeated prior to any modification of the system, frequently with poor results. A lack of system understanding has contributed significantly to the excessive time and manpower needed to be spent during maintenance. Mental models increase system understanding through descriptions of the development process, which will lead to significant benefits in overall system maintenance. The University of Minnesota and the STARS effort have examined the technology of mental models.[4]

REENGINEERING BENEFITS

Reengineering provides a number of benefits to developers and maintainers by using information from current computer systems. The large number of computer systems existing today increases the probability that a proposed system is very similar to an existing system, increasing the amount of relevant information available. The benefits of reengineering will also help reduce time, manpower, and the expense of maintaining and developing new computer systems by improving system understanding, preparing for future transitions, supporting system analysis, facilitating reuse, documenting design and implementation decisions, and improving the maintenance of existing systems.

Understanding

The major benefit of reengineering is its ability to provide a clearer understanding of the system. This is accomplished through generating high level abstractions of system information through reverse engineering and redocumentation. Additional help for understanding the system is achieved through documenting the decision-making processes which occurs during development and maintenance in engineering mental models.

Future Transitions

Transitions to new environments are usually expensive, requiring increased personnel and redevelopment for current systems to remain useful in the new

environment. Reengineering eases the transition to the environment of future systems. It provides a mechanism for injecting new structure or technology into an existing system. Reengineering addresses the translation of very large systems to new programming languages, or the changes necessary to port systems to new hardware configurations. Reengineering also supports development of new systems while capturing proven functionality of an existing similar system.

System Analysis

Reengineering assists in the analysis of computer systems by facilitating the generation of high level designs. Modifying the high level design first promotes early detection of errors and predicts the effects of change on the final system implementation. High level abstractions of the system components are useful for analyzing more complicated implementations.

Reuse

Reuse facilitates the full use of computer system modules and minimizes duplication during the design and implementation processes. It also encourages the development of optimized solutions for system requirements that allow the most benefit to be obtained from the modules by frequent use.

Design and Implementation Decisions

Design and implementation decisions are usually not recorded in any form for future maintainers. Reengineering provides a means of extracting this information and capturing this information in a formal manner that is used to educate maintainers and developers of similar systems. Many hours spent arriving at these decisions are often duplicated during maintenance or development of a similar system. The ability to quickly recall or learn how these decisions were made is achieved through reengineering, recording these mental processes for future use.

Improving Maintenance

Traditional development methods may not result in the development of maintainable systems. Reengineering provides a mechanism for recovering important information and defining processes for using this information to improve the maintenance process.

CHAPTER 2

BACKGROUND INFORMATION

Reengineering addresses the maintenance and development of Navy tactical computer systems. This chapter provides background information on the following issues: Navy tactical computer systems, the maintenance process, the development process, and systems engineering.

NAVY TACTICAL COMPUTER SYSTEMS

Most Navy tactical computer systems are strikingly different from other computer system applications. Tactical computer systems are typically environment driven rather than data driven. Navy tactical computer systems are subjected to external stimuli including natural phenomena such as radiation and salt water conditions, as well as man-induced phenomena such as hostile activity and man-machine interface. Current systems and future systems must address the issues deriving from these differences.

Current Tactical Computer Systems

Current tactical computer systems consist of software, hardware, and humanware. Tasks that are expected to change over time are frequently implemented in software because it can be cost-effectively changed as requirements change. Tasks that can be solved with a static solution are frequently implemented in hardware. The large amount of interaction with humans requires the system to provide suitable displays, and other man-machine interfaces; this is called "humanware." Each system component has unique characteristics making the overall system very different from most commercial systems.

Software Characteristics. The software for Navy tactical computer systems has unique characteristics. The code is traditionally highly mathematical in nature, and the outputs from the system are not merely data but also control signals to other computers or peripherals. A large number of Navy tactical computer systems are implemented in the CMS-2 programming language; other languages used include FORTRAN, C, and Pascal. These systems are classified as large software-intensive systems, typically having more than one hundred thousand source lines of code. The software must be designed around restrictions imposed by the hardware, including constraints of memory and central processor units.

Hardware Characteristics. Hardware for Navy tactical computer systems primarily consists of specialized shipboard computers, such as AN/UYK-43s or AN/UYK-44s, and

peripherals, such as tape drives, sensors, weapons, or displays. These hardware configurations have very unique maintenance and upgrade problems.

The computers are ruggedized, made for special environments, and are bulky and heavy. Additionally, they are composed of generalized uniprocessors and are networked in a point to point configuration; i.e., if a system contains 5 computers, 20 cables are needed to connect each computer to each of the other computers in each direction. (For N computers N*[N-1] cables are needed.)

These computers also have a high level of interaction with task specific peripherals such as sensors, weapons, and displays.

Humanware Characteristics. Humanware is all aspects of the computer system which are dependent on human interaction. Navy tactical computer systems have a high level of human interaction requiring sophisticated interfaces. Since these systems generally require an operator, the design and implementation of efficient, well-structured man-machine interfaces is one of the most important aspects of tactical computer systems.

System Characteristics. Tactical computer systems have unique system characteristics which are reflected in all components of the system. These characteristics evolve from issues including the life cycle, concerns about fault-tolerance, and real-time.

LIFE CYCLE. Tactical computer systems typically have extremely long system life cycles, requiring many modifications and enhancements. These systems are heavily relied upon by the Navy, requiring the systems to remain in an operational capacity while changes are being made to the system.

FAULT TOLERANT. These systems must be extremely fault tolerant, requiring recovery from errors or graceful degradation at a minimum. Many of these systems are involved in life-threatening situations requiring correct performance at all times.

REAL-TIME. Navy tactical computer systems are either soft or hard real-time systems. The majority of the current systems are soft real-time systems which are recoverable; i.e., one missed deadline does not fail the system. A hard real-time system is a system that is non-recoverable; i.e., one missed deadline may fail the entire system.

Future Tactical Computer Systems

Future tactical computer systems will take advantage of advanced systems engineering technologies; however, unique characteristics of the software, hardware, humanware, and overall system will still exist. The Navy has begun a new computing resources standardization effort, the Next Generation Computer Resources (NGCR) program, which is designed to fulfill the Navy's need for standard computing resources while allowing it to take advantage of commercial products and investments and to field technological advances quickly.

Software Characteristics. The software in future tactical computer systems will be different from the software in current tactical computer systems. Many of these

differences will be the result of using modern programming languages, techniques, and practices.

The software for these future systems will be written in a common programming language and will have encapsulated functions. The Department of Defense has mandated that all new software will be written in Ada. Many of these new systems will be very large, having greater than one million source lines of code.

Modern programming techniques include object-oriented design and object-oriented programming which is supported by the Ada programming language.

Hardware Characteristics. The NGCR approach is an open systems approach based on the establishment of interface standards. The application of these standards will change the Navy's approach from one of buying standard computers to one of procuring computer resources which satisfy the interfaces defined by the standards.

Programs such as NGCR proliferate alternative architectures for future Navy systems. These architectures include multi-processor distributed computer architectures and parallel processing architectures. Sophisticated hardware with faster and more efficient architectures will be used, but these machines will still reside onboard ship, making it necessary for these systems to withstand environmental conditions. The amount of memory in these computer systems will become larger as the availability of memory chips and devices increases, thus providing the means to store, process, and transfer increasing amounts of data.

Future computer systems will tend to rely on specialized processors for individual tasks; e.g., a system might use a neural network chip, or a dedicated graphical coprocessor chip. Thus there will be a greater reliance on application-specific and specialized hardware.

Humanware Characteristics. Humanware will have increased importance in future systems that are highly operator driven. There will be distinct and different approaches to human interface problems and graphics, in general. For instance, more emphasis will be put on user displays, since this is what the user sees and understands. There will also be a trend to make these systems more user friendly and easier for an operator to manage.

System Characteristics. Future system characteristics will reflect the transition to advanced technologies such as computer-aided engineering environments, cognitive processing, and concurrent processing. Additional characteristics will result from increased fault-tolerance and real-time concerns.

COMPUTER-AIDED ENGINEERING ENVIRONMENTS. Computer-aided engineering environments include Computer-Aided Software Engineering (CASE) products, automated configuration management support, documentation, code generation, and testing. These automated products help the designer to develop more modular and better organized systems. They also help with system modeling and in the rapid prototyping of future systems. A list of CASE products is provided in Appendix A.

COGNITIVE PROCESSING. Future tactical computer systems will have more cognitive processing as compared to current systems which use mathematical modeling. There is a trend to model functions in the way that humans do. This trend can be seen in

research areas such as neural networks and expert systems. Modeling and rapid prototyping will become commonplace in the systems engineering environment.

CONCURRENT PROCESSING. Future systems will be structured to take advantage of parallel processing by distributing concurrent operations across processors. Programming languages, such as Ada, will support this concurrent processing, and new architectures will consist of many processors to support this distribution. The high level of concurrent processing requires critical timing considerations to be met.

FAULT-TOLERANCE. Future systems will display increased levels of fault tolerance.

REAL-TIME. A major change in future systems will be the transition from soft real-time systems to hard real-time systems because of the increased reliability on tactical systems, the advancement of technology, and the changing threat to the Navy.

SYSTEM MAINTENANCE ISSUES

Maintenance consumes 70 to 80 percent of the overall effort spent throughout the life cycle of the system. A large portion of this percentage is spent on gaining an initial understanding of existing systems which will undergo many modifications and enhancements. These changes are addressed by the following types of maintenance: corrective, perfective, adaptive, and conversion maintenance.

Understanding

The major problem in maintenance is lack of system understanding. Frequently, the maintainers of the system are not the original developers who have a better understanding of the system. This problem is augmented because little or no documentation is available and the existing system implementation is not used effectively.

Human Factors. Understanding the system is intrinsic to the maintenance of systems. Very large, complex systems are extremely difficult to understand without supporting documentation. Developers come closest to achieving this understanding through the system development process; however, the original developers are often not available to help in the maintenance of the system once it is delivered to the end user. The developer's knowledge and expertise gained through the development process could be very helpful during maintenance and could be transferred to the maintainers of the system through documentation. Unfortunately, maintenance is usually performed by an intermediate engineering group which has little knowledge of the original development plan and little documentation.

Documentation. Documentation, which includes users guides, static reports, and initial and detailed designs, enables a system maintainer to understand the system quickly. Documentation, however, may be incomplete, incorrect, or not even exist. Incomplete documentation references system components which have been deleted from the current system implementation, or fails to reference new system components. In addition, incorrect documentation may not include modifications or enhancements made

to system components. Lastly, documentation does not exist: it was never generated, or has been misplaced.

System Implementation. The system implementation consisting of hardware and software may be the only representation of the system used in maintenance. There may exist a high level design which may have become inconsistent with modifications to the system that were not documented in the design. The existing design may have a difficult notation to understand. It may be beneficial to convert this design or generate a new one which can take advantage of current technology (e.g., CASE, design simulation).

The system implementation represented in an abstract form gives the maintainer the ability to understand the system quicker than by examining just the implementation. The high level information, rarely obvious in the system implementation, contains hardware and software design decisions and relationships between hardware and software components.

Types of Maintenance

The research performed at NAVSWC examined maintenance by categorizing the different processes into four distinct types: corrective, perfective, adaptive, and conversion maintenance.

Corrective Maintenance. Every computer system is tested to ensure that requirements are met, but even the most vigorous testing procedures may not insure that a computer system is completely error free. Additional problems may be found only after the system is put into use. Problems in the system which conflict with the original system requirements are addressed by corrective maintenance. These problems include errors in calculations, abnormal system halts, and any other errors which may be uncovered after repeated use.

Perfective Maintenance. After the system is used for a period of time, the needs of the users may change or increase. The system must be changed and enhanced in order for it to continue to benefit the users and prolong its life cycle. Perfective maintenance adds new capabilities to the system and optimizes the existing system. These modifications include adding new functional requirements as well as enhancing existing functions.

Adaptive Maintenance. The environment in which the system operates is highly volatile, requiring Navy systems with long life cycles to undergo adaptive maintenance. New types of data may need to be addressed by the system. Alterations to the input/output routines of the system to accept and return this data are performed by adaptive maintenance.

Conversion Maintenance. When the modifications to an existing system consist of translation to a new programming language, the maintenance required is sometimes called conversion maintenance. (Translation is often not recognized as maintenance.) This type of maintenance is generally applied to only small programs, particularly when the two programming languages are similar. The process of converting a large system to a new programming language is extremely difficult and time-consuming, requiring extensive manpower and funding to complete the effort.

Simple code translation is not feasible for the large, complex real-time systems existing in the Navy.

SYSTEM DEVELOPMENT ISSUES

The choice of a development process is the most important maintenance decision that will be made during the life cycle of the proposed system. The way in which a system develops directly determines the resulting system's ability to accept future modifications. Computer systems that are well-documented are easier to modify later. Many of the development processes can be automated which improves efficiency and adherence to standards.

Development Process

The computer system development process often begins with a set of well-defined requirements, followed by the development of an initial design, advancing to a detailed design, and concluding with a final implementation of the computing system. This process is applied to both hardware and software. Other processes which support the development phase include risk analysis, prototyping, simulations, and design executions. There are several formal models which define the development process for computer systems including the Waterfall (Figure 2-1), Enhanced Waterfall (Figure 2-2), and the Spiral Model (Figure 2-3).[5] For Navy tactical computer systems there are standard development processes which are primarily based on the Waterfall Model. These standards include: DOD-STD-1679, DOD-STD-2167, and, most recently, DOD-STD-2167A. Computer systems developed using structured techniques, well-defined models, and formal standards are easier to maintain.

Documentation

There are several formal methods for representing a proposed system in a high level design (e.g., PDLs for software, and VHDL for hardware). A well-documented system ensures that an eventual maintainer of the system will have the best possible advantage to successfully modify the system. This documentation provides high level insight into the processes of the system and gives the foundation for understanding system complexity.

Automated Support

Many development processes are supported today by automation provided by CASE products. Some of these products attempt to address the entire life cycle of a system which can also help in the maintenance of the system. The statement of requirements, design generation, analysis and automatic code generation are supported by automation. Automated configuration management products are also available to assist maintenance. There are even automated products including CASE which support the automatic generation of documentation for computer systems.

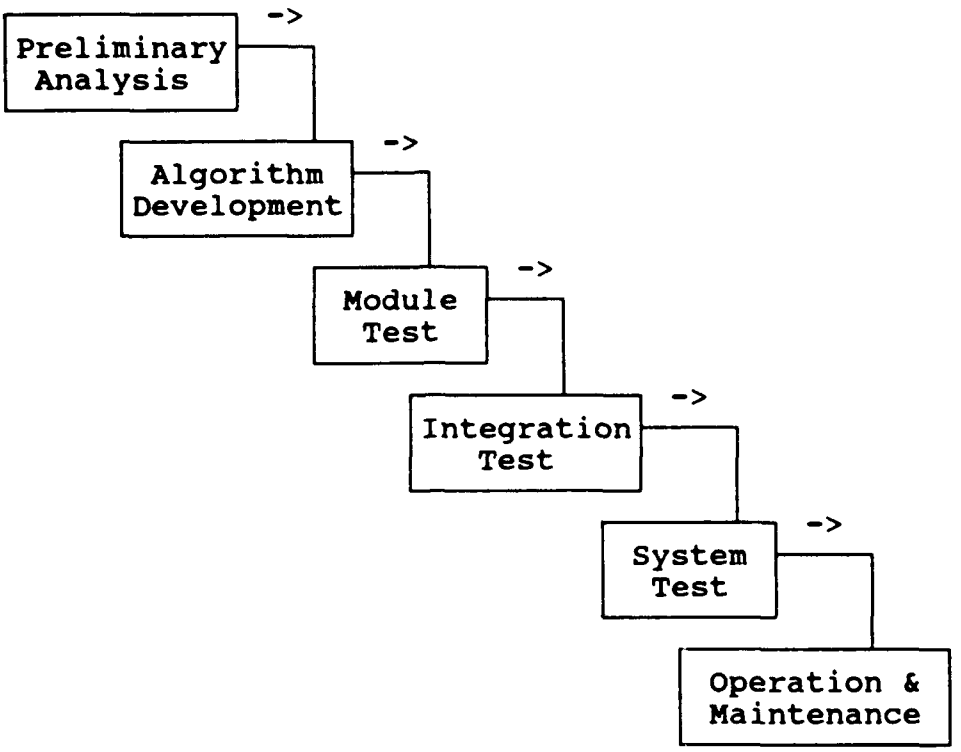


FIGURE 2-1. WATERFALL MODEL

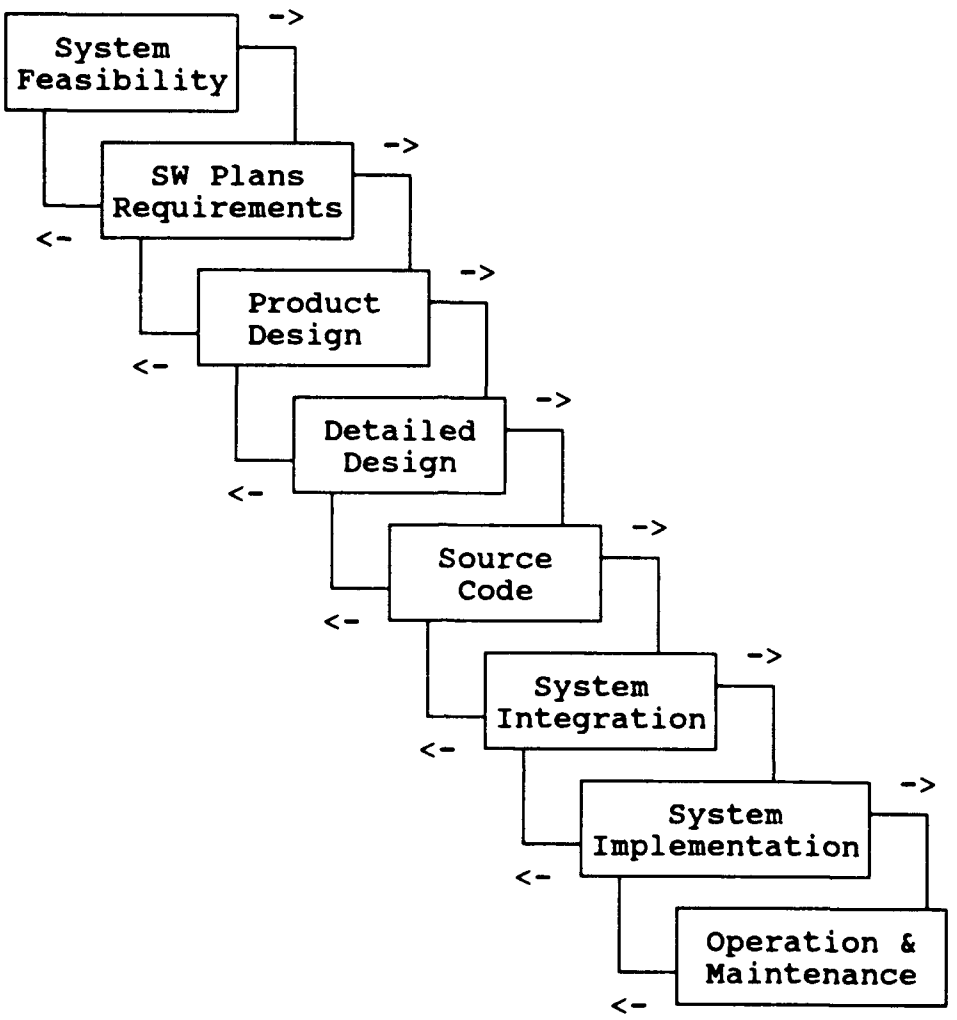


FIGURE 2-2. ENHANCED WATERFALL MODEL

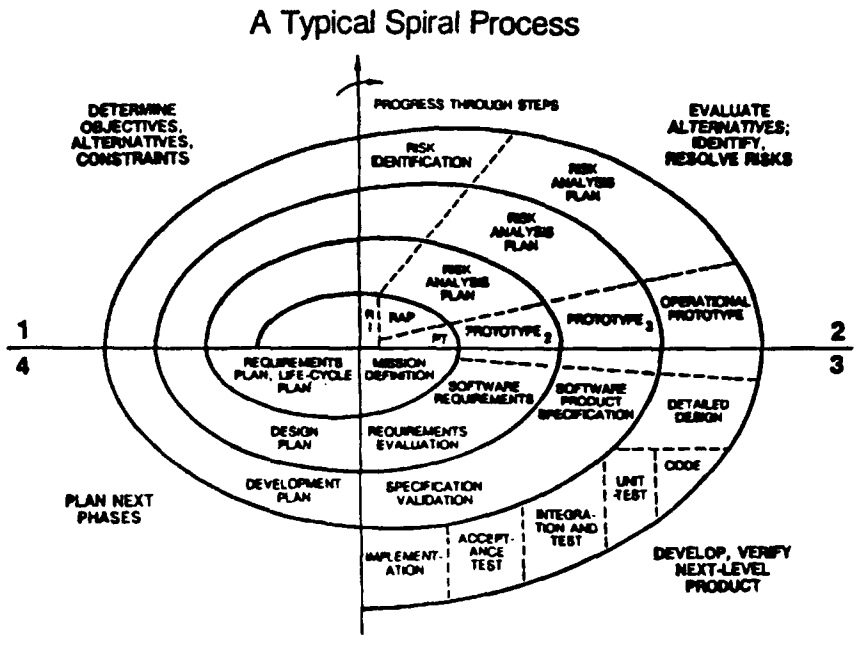


FIGURE 2-3. SPIRAL MODEL

CHAPTER 3

CURRENT STATE-OF-THE-PRACTICE

The current practice for maintaining existing systems relies heavily on directly modifying the final system implementation, ignoring any other available documentation. More complicated maintenance efforts usually result in complete redevelopment.

DIRECT MODIFICATION

The most common practice for supporting existing systems involves directly modifying the final system implementation, including direct code translation and direct hardware replacement. Existing documentation, such as users guides, specifications, and high level designs, are not examined prior to making changes to the system. The benefits of direct modification are predominately short-term and are very misleading. The drawbacks, often long-term, further complicate the maintenance problem and inhibit the development process.

Benefits

The short-term benefits of direct modification include a reduced development and test time for the new system since a proven implementation already exists and is being used. Direct code translation provides the quickest means for modifying current system software or obtaining a new implementation in a new programming language. Direct hardware replacement provides the quickest means for modifying the existing hardware or establishing a new hardware configuration.

Direct Code Translation. Direct code translation is often used to meet a new programming language requirement. It is very useful when the languages are similar, the functional requirements are the same, and the target architecture is the same. Similar languages have corresponding data and control structures for performing a one-to-one translation of the source code. The source code of an existing system which meets the functional requirements of a proposed system may be used in the process of translating the existing system into another programming language; e.g., a program originally written in FORTRAN can be directly translated into the C programming language. Direct code translation may be useful for immediate needs where the existing system is functioning well on the existing architecture and the proposed system will execute on the same type of architecture.

Direct code translation can be used to quickly achieve new software which can then be optimized using restructuring techniques. The restructuring techniques can be used to introduce new capabilities of the new programming language.

Direct Hardware Replacement. Direct hardware replacement is appropriate for meeting dynamic requirements (e.g., improved system performance) when no new functionality is required. Individual hardware components can be replaced with like components if there is minimal impact on other system components (software and numanware).

Drawbacks

The most crucial drawback of direct modification is that it does not support the structured development process espoused in standard software and systems engineering practices, and is generally frowned upon in most structured software and systems engineering environments. With this strategy there usually is no mechanism for injecting new structure or technology into the new system. It does not address new functionality requirements which most frequently need to be addressed. Direct modification encourages the habit of incompletely and incorrectly documenting computer systems. The drawbacks are reflected in applying direct code translation and hardware reconfiguration.

Documentation. Direct modification of the system does not use any additional information available in the form of documentation. Users guides, specifications, and high level designs all provide high level views of the systems functionality. These documents provide additional sources which help in achieving a clear understanding of the system in order to modify it.

Failure to use existing documentation prior to implementing changes to the final system usually means that the corresponding documentation is not updated to reflect the changes made to the system. Documentation becomes incomplete and inconsistent with the system it should accurately represent. The final system implementation becomes the only true representation of the system.

Direct Code Translation. Code translation does not optimize the new system for the new functional requirements in the software. The original functional requirements are still addressed in the new programming language. New requirements will need to be introduced once the conversion is completed.

Code translation also does not address new architecture requirements. For example, if the new system is on a parallel processor machine, but the current system is on a uniprocessor machine, the translated code would also be written for a uniprocessor machine. Architecture requirements addressed by the software will no longer be effective in the new implementation for a new architecture. For instance, an AN/UYK-44 computer has 64K paging due to lack of available main memory. If the new hardware were in the Motorola 68020 class, there would more than likely be 8MB of memory and the page swapping would be highly unnecessary and may cause timing or other problems.

The idiosyncracies of the original implementation language are inherited by the new implementation. For example, if the target implementation is to be in Ada and the original implementation was in FORTRAN, the translated code will be "FORTRAN like" Ada; i.e., Ada that is written to look like FORTRAN.

Some of the unique characteristics of the original language may not even be translatable in the new language. Data types such as records and pointers which are

central to a language such as Pascal, do not exist in a language like FORTRAN/77. If the original implementation language uses these features, then there will be major problems in translating the code to a target implementation where these features are not available.

The unique features of the target implementation language are not used. For instance, features such as tasking and generics which are part of the Ada programming language are not available in a language such as FORTRAN.

Hardware Reconfiguration. Major replacement of hardware components and modification to the overall configuration of the hardware affects the other system components. Replacing components which alter the functionality of the system requires that all other system components (software and humanware) be examined to ensure their ability to perform is not jeopardized.

REDEVELOPMENT

Complicated maintenance and development efforts are often addressed by a complete redevelopment strategy. Numerous new requirements justify developing a system without using all of the available information from similar existing systems. A system developed from inception provides a fresh approach to meeting requirements and permits the use of recent technology advances. Experience gained from previous similar developments, however, could provide insight into how requirement ambiguities were resolved and decreases the overall development effort.

Advantages

The greatest advantage of this technique is that the current software engineering techniques which support modern programming languages such as Ada could be used. Additionally, object-oriented design techniques could also be used, thus reducing the complexity of the code and increasing the ability to maintain the code. In the long-term, the code generated by this technique would be high quality, reliable, and cost-effective. However, the cost associated with this technique is very high and the resources needed to accomplish this are not always available.

Disadvantages

The greatest disadvantage of redevelopment is that the years of experience gained from developing an existing similar system are not used. Ambiguities in the requirements must once again be resolved, and major design issues such as input/output, interrupt handling, and data typing must once again be decided. The most time consuming disadvantage is that the complete testing and integration process must be completely redone from scratch. This results in increased man-power and material costs.

CHAPTER 4

REENGINEERING FOR NAVY TACTICAL COMPUTER SYSTEMS

Reengineering for Navy Tactical Computer Systems is a structured plan addressing the development and maintenance of large, complex systems. This plan uses reverse engineering to extract high level information from an existing software intensive system, representing this information as a detailed design. This reengineering strategy is linked to the systems engineering process to support the development of new systems. There are three phases in the strategy: the initial definition phase, the analysis phase, and the transformation phase.

INITIAL DEFINITION PHASE

The initial definition phase is the examination of the subject system as required by the basic definition of reengineering. Feasibility is assessed using metrics; reengineering environments are defined; automatable processes identified; and automated products selected during this phase.

Feasibility Using Metrics

Prior to any reengineering effort, the feasibility of such a project must be confirmed and a plan for performing reengineering established early. Not every computer system is a viable candidate for a reengineering effort. There are many factors which must be considered prior to altering an existing computer system using reengineering techniques. Feasibility can be determined through a checklist of criteria which can be used to determine the computer system's potential for successful reengineering. Although no formal checklist has been proposed at the time of this writing, suggested items for consideration when determining the feasibility of reengineering software might include the following:

- Was the software developed using standards?
- How many lines of source code exist?
- What data structures are used?
- How much data coupling exists?
- Are multiple programming languages used?
- What peripheral devices are interfaced?
- Are the original developers available for consultation?
- What supporting documentation exists?

The criteria contained in a checklist could be measured using metric analysis.[6-8] Additional common sense observations about the system also predict the feasibility

of reengineering, including the recognition of "spaghetti code," use of "gotos," and evidence of "hacking."

Metrics. Metrics determine the feasibility of a candidate computer system. The problem of judging the quality and productivity of a computer system is a major issue in software development and software maintenance. Before a system can be reengineered, it must undergo a preliminary assessment of the current system to determine the feasibility of performing reengineering in the first place. Not every system is an appropriate candidate for reengineering. Metrics can be used to determine whether reengineering is appropriate. Metrics are divided into program and process measurements. There are three major areas that need to be explored when cumulating quality and productivity statistics: the current system implementation, the system development of current system, and the system development of future system.

CURRENT SYSTEM IMPLEMENTATION. The current system implementation needs to be judged on both design and performance qualities. Design characteristics are static and can be done without having the code executed. On the other hand, the performance characteristics are dynamic and need to be examined while the system is executing.

The ideal computer system is non-defective, correctly meeting the requirements for the existing specification. This system is the product of a structured systems engineering environment, having been developed using system programming standards. These characteristics will ensure that the system is understandable, easy to modify, and verifiable.

The design qualities that need to be examined include such attributes as modularity, system clarity, information hiding, encapsulation, and dependency isolation. In addition the user might ask: is the system non-defective (does it perform within the allotted time with high reliability). The amount of low-level code as well as its complexity may also be an important measure for some systems. Additionally, both the size of the system, in terms of the number of source lines of code, and the number of lines of comments are important measures.

Performance characteristics include such characteristics as real-time constraints (timing deadlines), reliability, response to load conditions, and security. However, critical path analysis is a static method of determining performance characteristics of a given system.

SYSTEM DEVELOPMENT OF CURRENT SYSTEMS. The process of judging the system development of the current system is more subjective. This process gives us insight into how the system was developed and how good a base we may have for using this system to develop a new system. The characteristics that are judged in this process include: whether the system was developed using standards; is a high level design document available; is the documentation consistent with the implementation; is the design matched to the implementation; and are the requirements matched to the design. The effort expended during each phase of the development process in terms of man-hours and computer time may also be necessary to evaluate.

SYSTEM DEVELOPMENT OF FUTURE SYSTEMS. The future system development is the most difficult to judge, because it is hard to predict what effort will be required to develop a new system. The four most reasonable criteria for judging a system of

this type concern: reasonable estimates, minimum costs, realistic schedules, and system improvement; e.g., performance, reliability.

Additional Observations. Additional observations about the system also help in judging the feasibility of reengineering. The subject system should be error free and contain good quality code in order for the system to be a viable candidate for reengineering. When the ultimate goal of the reengineering is to generate a new system, the changes necessary for the existing system to meet the requirements of the new system must be considered. The number of modifications should be minimum in order for the existing system information to be useful.

Environments

There are three environments affecting the reengineering process. These environments must be defined early in the Initial Definition phase and well-understood by the reengineer who must serve as both maintainer and developer of the system.

Original System Environment. The existing computer system was developed and currently executes in the first environment. The description of this environment includes the programming language in which the existing system is implemented, any existing information about this system; i.e., old documentation or requirements specifications. A description of how this system was developed may also include standards and design methods, and it should be determined whether automated products were used in the development process.

New System Environment. The second environment will support the new computer system or in the case of maintenance, describes the changes of the existing environment which necessitate reengineering. This description includes new requirements specifications: hardware requirements (e.g., behavioral requirements, space requirements), and any new software requirements (e.g., implementation language, optimization requirements, new functionality, faster processing), humanware requirements (e.g., input/output, displays, man-machine interfaces).

System Reengineering Environment. The selection of the methods for reengineering which will be applied to the candidate system must also be completed early in the Initial Definition Phase. This selection represents the reengineering environment which is defined by the reengineering capabilities. In the case of reverse engineering this would include a plan for performing metrics, the formalism(s) for design capture, the methods for abstraction, and any automated support for performing reverse engineering.

Automated Support

Certain processes in reengineering can be performed or supported by automated products. Automation provides several benefits when properly applied in a systems engineering environment through three levels of support. Prospective products should be evaluated by a well-defined set of criteria which supports this environment.

Benefits. The benefits of automated support affect the following areas of computer systems: cost, process (development and maintenance), and personnel. Automation is cost-effective, accelerating the required process and limiting the number of technicians required to perform the tasks. Automation can help improve the maintenance process by providing a mechanism for enforcing standards and providing support throughout the life cycle. A well-designed automated product insures a single interpretation of the methods and techniques it automates. Automation lessens the number of people involved in the process minimizing errors. Most maintenance is performed manually, requiring extensive manpower over a long period of time and resulting in many errors.

Levels of Support. Various levels of support are provided through automation. Automated products are divided into three levels of support: front-end, analysis and design, and back-end.

FRONT-END AUTOMATED SUPPORT. Front-end automated support products perform preliminary analysis on the software of a system. They provide information on the complexity of the code by describing the number of conditional statements, recursion, loops, and volume of lines of code. Parsers provide detailed information describing the data and control structures in the source code. These products are often called front-end compilers.

ANALYSIS AND DESIGN AUTOMATED SUPPORT. Analysis and design automated support is supplied by CASE, one of the most highly evolving areas of computer system development. In the past decade CASE products have progressed from simple front-end graphical support products primarily used for drawing diagrams to environments which include more sophisticated components such as data bases, code generators, report generators, and capabilities such as configuration management support, static and dynamic analysis, simulation, validation and verification, and more. The effectiveness and correctness of CASE has been the subject of extensive evaluations.

BACK-END AUTOMATED SUPPORT. Back-end automated support links the reengineering process to the traditional forward engineering environment, facilitating the generation of the final system. Automatic code generators produce source code from designs. Test case generators provide sample data to test the execution of the final system. Verification and validation products provide more detailed testing of the final system.

Evaluating Products. Automated products should be selected which best support the existing engineering environment where they will be used. Some products may provide efficient support stand-alone, while others may require additional manual effort to complete the process. All of these factors should be taken into consideration before selecting any automated product for use. A list of reengineering products and manufacturers is included in Appendixes B and C with a brief description of the product and the reengineering processes which it automates.

ANALYSIS PHASE

The analysis phase consists of a reverse engineering process which abstracts existing system information to a high level design. Once the high level information is captured in a formal notation, the design can then be analyzed.

Design Abstraction

Information is abstracted from the system and captured in a high level design. The design must be maintainable, including all of the information necessary to adequately represent the existing computer system. It must be independent of the current implementation of the system since it may be used to generate a new computer system. All or part of this design can be used, and it can be changed or enhanced to maintain the current system or meet the requirements of the new system.

Design Analysis

The design is analyzed to verify that the design completely and correctly represents the existing system. A complete design is achieved when all requirements are allocated to the design, all associated descriptions of components are present, and all associated parts of the design are appropriately connected. The design is correct if it contains no semantic or syntactic errors in the notation, cross-references are available where appropriate, and there are no conflicting control structures.

TRANSFORMATION PHASE

The transformation phase links the reverse engineering process to the forward system engineering process. The modifications necessary to optimize the design through enhancements or alterations for new system requirements are performed during this phase. The optimized design is used to generate the final system during the forward systems engineering process.

Optimization

The design generated during the analysis phase is optimized to ensure the most concise design for the existing system or is modified for new system requirements. Optimizations at the design level produce concise representations of the design and ensure design completeness. Modifications prepare the design for a resulting implementation on a new architecture. Specific parts of the design are mapped to the target architecture components where possible for the best system implementation.

Forward Systems Engineering

The latter part of the transformation phase is primarily composed of standard computer system development practices using forward systems engineering processes to generate the new system. The system software is produced from the high level software design. The hardware is modified or configured using the new hardware design. Verification and validation of the resulting computer system are performed to ensure the system performs as required. The design may be used to generate a complete computer system or only part of an entire system.

CHAPTER 5

REVERSE ENGINEERING

Reverse engineering was selected to implement reengineering for Navy tactical computer systems for two reasons: (1) the use of high level designs is intrinsic to structured computer system development processes and standard engineering principles and (2) the generation of a current high level design links reengineering to this structured development environment.

The design is generated using reverse engineering to extract high level information from the implementation, capturing this information in an analyzable form which can be used to generate high level system designs for maintaining the current system or developing a new system. There are several techniques for performing reverse engineering.

EXTRACTED INFORMATION

Information from an existing system can be extracted and represented in a reusable format. System information is divided into two categories: static and dynamic.

Static

Static information is lifted directly from the system implementation where it is represented as a constant part of the system. Static information is relatively easy to measure and represent statistically. It is composed of measurable information about the software and hardware.

Software. Software contains data information such as variable names and data structures, as well as compositional information such as subprograms, procedures, and functions.

DATA. The data which the system accepts, processes, and outputs to other devices is also represented in the software. Well-defined descriptions of the inputs to the system and outputs from the system (e.g., data, signals) are important to understand the system. It is also important to understand how this data is stored once inside the system (e.g., data structures), the relationships between various data items, and how the data progresses throughout the system (data flow).

COMPOSITIONAL. Compositional information describes the structure of the software (e.g., number of procedures, functions), the functions which the system must perform (e.g., process descriptions), and the order in which these functions are performed

and how they interrelate (control flow). The implementation of a system represents the division of the system into components which perform the system requirements. How these components interact with each other and share data is also important to understand. The system may also communicate with other devices and peripherals which require efficient, well-defined interfaces. Descriptions of these interfaces and how they were implemented is important to fully understand the system.

Hardware. Hardware contains information about the configuration, architecture, peripherals, and individual hardware components. The number and types of hardware components, external devices, and how they are connected are all measurable aspects of the system hardware. Performance characteristics of the hardware are regarded as dynamic information.

Dynamic

Dynamic information must be inferred from the system implementation such as how the system performs. The software has dynamic characteristics describing speed and reliability; hardware dynamics describe speed and power. Much of the dynamic information is influenced by the interaction between the software and the hardware. Real-time issues are dynamic characteristics that are critical in Navy systems. Certain tasks must be completed in a specified time. Sometimes tasks must be performed very fast, but more importantly the system must be predictable; i.e., the amount of time necessary to complete a task must be measurable and consistent. There is interaction between subsystems, components, even individual processes. The timing of completion for these entities must also be predictable and consistent for this interaction to occur. The system must also be reliable, able to handle errors, and have the ability to recover in the necessary amount of time and in the appropriate manner. The physical components of the system must meet any space requirements that may result from the placement of the system at a location onboard ship, aircraft, or even in an onshore facility.

DESIGN CAPTURE ENVIRONMENT

Several design notations are currently proposed for capturing high level information from a computer system. The methodology for capturing the complete high level design of the subject system must be defined in the Initial Definition phase and this methodology must be well-understood by the reengineers. The method for capturing the design should be correct and provable. The method will use a notation or combination of notations to represent this high level design. This representation should be formally analyzable and should intuitively abstract the high level information of the system to the design level. The representation should be implementation-independent but traceable back to the requirements level and also easily mapped to the existing or a new computer system implementation.

There are two types of design capture that most fully represent the system: textual and graphical. There is also additional information which supports the generation of the more complete system representations. This information must also be encapsulated in a formal representation.

Textual Representations

Textual representations capture the high level information of a computer system in the form of English-like text, including pseudocode, specification language, or a PDL. The hardware of the system may be represented using a textual representation such as VHDL.

Pseudocode. Pseudocode is very similar to a PDL. Pseudocode has characteristics similar to high level programming languages, but is not required to adhere to any programming rules. The designer has the flexibility to create a design which is very similar to the final implementation, but will not have to meet compiler requirements. However, implementation-dependent designs have little potential for reuse, and errors traditionally detected by a compiler may signal design faults.

Specification Language. Specification languages are often highly mathematical, incorporating specific mathematical notations within the context of the language itself. These languages are often difficult to learn and interpret. Some specification languages include: MODEL[9] and Larch.[10]

Program Design Language. A PDL is similar in format to pseudocode and high level programming languages, but does not necessarily result in an implementation-dependent representation. Disciplined designers can use this textual representation to produce a simple description of the system or a more detailed design. Compilers are also available for error-checking of some well-documented PDLs.[11]

VHSIC Hardware Description Language. Hardware configurations can be represented at a design level using VHDL. Although VHDL is primarily used to represent the more detailed design of specific hardware components such as a chip or circuit board, it has also been suggested that this notation could be used to represent the high level design of the hardware system configuration.

Graphical Representations

Extensive research in the area of computer system development design using graphical representations has resulted in many different views of a computer system. Most experts agree that multiple system design views (rather than one single view) are needed in order to fully capture the essence of large scale complex system.

There are several notations used to represent the different views of the system. Some of the notations result directly from the view they support, while others can be used to represent more than one view. The notations described below are well-documented and the most widely used. They are supported by a majority of automated products available for computer system design. Other notations exist, but most are considered a subset or variation of one of the views described below. The research performed at NAVSWC was limited to the notations described below because of the widespread support and documentation available.

Structured View. The structured view depicts the actual form of the computer system and how it is divided into parts that perform the required processes of the system. Structure charts provide a view of how the computer system is partitioned into the various sections which together perform the function of the system.[12]

Object-Oriented View. The object-oriented view of a computer system is the newest view, showing a set of objects and the relationship of these objects. Grady Booch has outlined the process of developing computer systems using an object-oriented view in Reference 13. The object-oriented view does not fully represent the decomposition of a large system, and usually does not represent any behavioral characteristics, such as timing constraints between processes. This view does, however, support the visual identification of data elements and their relationships. The view itself supports an object-oriented approach to design and analysis of the represented system.

Entity relationship diagrams depict the system as a composition of objects and the influence on one another that each object possesses. These diagrams support the object-oriented view. Peter Chen outlines how to model a system through entity relationships in Reference 14.

Functional View. The functional view is the most popular view of a computer system, tracking the movement of data throughout the system. This view is also called the data view because it is usually represented in data flow diagrams. Tom DeMarco[15] is often referenced for his Structured Analysis technique using data flow diagrams; Edward Yourdon and Larry L. Constantine[16] describe their system design technique as Structured Design; Michael Jackson[17] is another researcher in the field of structured design with his Data Structured Design technique; Chris Gane and Trish Sarson[18] have also outlined techniques and automated products for structured system analysis.

The functional view represents the "real work" of the system by describing the functional requirements of the system. The requirements are represented as processes with the data necessary to perform these functions passing between these processes. The processes are represented hierarchically so that analysis can be performed from the top level down to a reasonable level selected by the designer. The view, however, does not depict any information concerning the required behavior of the system. This type of information may be included in additional annotations which can be tagged to the individual processes. These annotations may be relied on too heavily by the systems designer which could lead to early design errors which would go undetected.

The functional view is implementation-independent. The view depicts a minimum amount of data flowing between processes while it is also rigorous enough to avoid many analysis phase errors.

Control View. The control view represents the behavior and timing of the system by depicting the sequence of activities which the system performs and the status of the system throughout execution. This view is often used in conjunction with the functional view to fully represent the system from the highest level (what kind of work the system will perform) to the lowest level (how the work will be performed). Control flow diagrams show which parts of the computer system obtain control and when this control is released to other parts. Boeing and Lear-Seigler developed a control flow method which also supports real-time design.

The strongest capability of the control view is its ability to represent the behavioral characteristics of the system, including the order and timing of process activation. Strategies for depicting control flow are defined by Paul Ward and Stephen Mellor in Reference 19. The drawbacks of this view consist of a dependency

on a specific implementation language as well as the limitation of potential repartitionings of system processes.

Concurrency View. The strongest ability of the concurrency view is its ability to represent behavioral characteristics of the system and identifying the concurrent processes of the system. The notation symbols generally used to represent this view (Buhr diagrams) do not have a rigorous semantic interpretation which results in difficulty learning and understanding the notations. This view also is easily supported by automatic code generation. There are enough constraints with this view, however, that when applied properly errors can be detected early during analysis. There is, however, little emphasis placed on the data in the system and there is usually inadequate representation of data relationships which may be key to many systems. This view usually reflects the final system implementation.

Real-Time Characteristics. The issues of real-time systems must be addressed during the design and development process of a system, requiring equal consideration in the reengineering of a computer system. The representation of real-time attributes in a system design is not adequately addressed in current research. Real-time Extensions, one of the few results of research in this area, were developed by Derek Hatley and Imtiaz Pirbhai[12] to provide a means for including some real-time characteristics to an existing system design. More recently, Melir Page-Jones discusses these issues in Reference 20.

ANALYSIS TECHNIQUES

Analysis is performed throughout the reengineering process. Standard software engineering practices support several techniques for analyzing computer systems. Traditionally, the computer system design is checked for completeness and correctness with respect to data flow and control flow. CASE products provide an automated environment for performing analysis; simulations and prototyping also assist in analyzing system designs.

CASE Environment

The introduction of CASE products in the software engineering environment has provided the automation of analysis through both traditional checks and additional means of analysis of computer system designs. Automated design analysis can ensure that the design correctly captures the functionality of the existing system. This analysis can be supported by a CASE product (or any structured environment for maintenance and development). Although CASE provides the most benefit in the development of computer systems, the support of an existing system can also be performed once the design is ported to the CASE environment. Modifications necessary to meet new requirements for a proposed system can be made within the CASE environment.

CASE provides automated support for the design and analysis of computer systems, as well as many other support activities. Moving an existing system into the CASE environment is not a fully automated process, requiring some manual effort. Designs can be "executed" to examine the performance of a design given new sets of data and various environmental parameters.

Simulations

Simulations of a computer system are another means of analyzing the proposed system. There are several simulation languages and automated simulation products available which support the development of simulations.

Prototyping

Prototyping has historically been criticized in the structured engineering environment. Recently, critics have reviewed prototyping as a suitable means for analyzing proposed systems. Parts or all of the proposed system can be implemented to examine how the proposed system will perform.

Once the initial code is generated from the high level design, it may need to be optimized. The system will need to be verified and validated as to the requirements specification. Additional testing will also prove that the requirements have been met.

REVERSE ENGINEERING TECHNIQUES

There are several proposed techniques for performing reverse engineering. Although most techniques fit to one of three categories: semantic/syntactic, mathematical, or artificial intelligence, each technique for reverse engineering may or may not necessarily denote a specific method for capturing the high level information in a design and the eventual notation for representing the design. A method for capturing the design of a computer system may be used singularly or in conjunction with another in order to fully capture the system at the design level. For example, semantic/syntactic techniques may be used to capture the data flow view of a computer system while an expert system may be used to extract the functionality of the same computer system.

Semantic/Syntactic Techniques

Semantic/syntactic language grammars extract high level information from an existing system. This information can then be represented in a system design. The language grammars use semantic and syntactic information contained in the actual code to determine the functionality of a system and provide high level information about the system. Automated processes that are based on semantic/syntactic techniques include data cross reference, data and control flow analysis, metric analysis, and procedure hierarchy.

Mathematical

Mathematical techniques use mathematical-based equations for interpreting computer system functionality. These techniques for extracting information from a computer system include Boolean algebras and Backus-Naur form (BNF).

Boolean Algebras. Boolean algebras can be used to describe the processes of a computer system. These mathematical expressions define the functionality of the

system independent of the programming language in which it is implemented.

Backus-Naur Form. Another mathematical technique is the translation of data definitions into BNF. This form is also implementation-independent and can be used as a high level representation of the system. Several manufacturers of automated products have developed a methodology for mathematically interpreting computer systems. These methodologies define processes for translating the high level information of the system into mathematically-based equations. These methodologies are then supported and automated in the products produced by the manufacturer. There is also research at the University level which is studying the benefits of representing computer systems through math-based techniques.

Artificial Intelligence

In addition to these techniques, applications in artificial intelligence can be used to extract high level information from the computer system. Research in the area of knowledge extraction, hypertext, and expert systems reveals that the technology of artificial intelligence can be applied to the field of reengineering. Research is also examining how computer systems are developed in an effort to learn better ways to maintain the existing systems and ways to develop new systems. Models of the development process may establish a better understanding of a computer system and how it is created.

CHAPTER 6

LESSONS LEARNED

The technology of reengineering provides capabilities for solving many of the fundamental problems of maintaining and developing computer systems. Although the technology is evolving quickly, the theories for supporting these capabilities are incomplete, failing to address specific problem areas which are important to Navy tactical computer systems. Additional research in reengineering is summarized in Appendix D. Automated reengineering products also reflect the inefficiencies in the proposed theories of reengineering and additional inadequacies resulting from the automation process itself.

REENGINEERING THEORY

Proposed theories for reengineering do not address the following Navy-specific areas: Navy-specific languages, real-time issues, abstraction, systems engineering, and large systems. These areas are of vital concern to Navy computer systems and must be addressed in order to effectively reengineer these systems. Most reengineering theories were developed for business applications, making these theories more difficult to apply in Navy applications.

Navy Specific Languages

Navy specific languages such as CMS-2, and other languages which are not predominant in the development of Management Information Systems (MIS) have special considerations which are not typically addressed in current proposed reengineering techniques. A list of available support products for CMS-2 is located in Appendix E. Assembly language is also not addressed by these techniques. Many scientific systems as well as weapons systems contain assembly directives which are not easily interpreted or abstracted to the design level. These directives are difficult for humans to interpret, restructure, and retarget, and the understanding of these directives is often prone to errors. Obtaining an understanding of the assembly commands is intrinsic to successfully reengineering the system.

Real-Time Issues

Real-time issues should be considered during reengineering, abstracting real-time characteristics to the design level during reverse engineering. These issues are often requirements of the computer system and must be represented in any design or other representation which provides an understanding of the system.

Abstraction

There are few formalized techniques for obtaining high level abstractions of the system. Standard software engineering practices teach that efficient computer system development and maintenance is supported by the existence of a good design. The design of an existing system can be captured through reverse engineering techniques that extract appropriate information from the system and other available documentation.

Systems Engineering

The systems engineering environment is seldom considered in the reengineering process. Software is addressed in isolation from the architecture system during most current reengineering practices, ignoring new architecture requirements.

The characteristics of current and future computer systems demands that the development and maintenance of these systems be addressed from a systems engineering point of view rather than a traditional software engineering approach. Software is only one part of a system and since it cannot work alone, it is always dependent on the hardware on which it operates, as well as the people who interface with it.

Large Systems

Large Navy computer systems typically have between 100,000 and 1,000,000,000 lines of source code. These complex systems are very difficult to understand and support. Most reengineering theories are applicable to small computer systems and are not scaleable to larger systems.

Business Systems

Reengineering business systems in the commercial environment has progressed farther than scientific or defense systems reengineering. Many MIS or other business systems are based on models that have existed for many years and are well-understood. The type of reengineering used for business systems is predominantly simple restructuring of an existing implementation language which is usually COBOL, or conversion between commercial data bases. System conversion to a new architecture is seldom a requirement for business systems which normally are executable on a wide-range of architecturally equivalent hardware configurations.

AUTOMATED SUPPORT PRODUCTS

There are several products available which support reengineering, and the number of products continues to increase. Many of the traditional CASE manufacturers are beginning to enhance current products with additional capabilities that support reengineering. Automated products are based on the proposed theories in reengineering, failing to address many Navy-specific issues. In addition, automation emphasizes direct code translation, utilizing only the software of the system. Finally, most automated products are designed for business rather than scientific applications.

Navy-Specific Issues

Many Navy-specific issues such as real-time design, large system development, and Navy language support are not addressed by most automated products since these issues are not dealt with in the proposed reengineering theories.

Code Translation

Direct code translation is the current practice for utilizing existing system information to develop a new system. This also is a result of disregarding systems engineering by examining only the software and not the system architecture. Automatic code translation often generates poor quality code which still must be optimized. The process of code translation still does not prepare the software for a new architecture or hardware which increases the amount of effort necessary to reach to the desired system implementation. Repeated code translations result in the weakening of the code structure and the eventual inability to use the software. Emphasis is placed on automatic code translators instead of supporting abstraction techniques through reverse engineering and the development of high level designs.

Business Systems

There is a large effort in the commercial environment to produce products which support reengineering of MIS and other business systems. Data base reengineering is another area that has potential for automated support.

RELATED TECHNICAL ISSUES

In addition to lessons learned in reengineering theories and automated products, there are issues which must be recognized as a result of the research performed at NAVSWC.

Maintenance

Current maintenance practices cannot adequately address the translation of very large systems to new programming languages, or the changes necessary to port systems to new hardware configurations. Software usually represented in high level programming languages is very difficult to read and understand. Many aspects of the software may be overlooked and more complicated algorithms may be misunderstood. Data structures and control sequences in the software may not be visible, and modifications may result in unforeseen results.

Development

Very few systems are consciously developed with any thought as to how the system will eventually be maintained. Ironically, most systems will inevitably require many changes to prolong the life of the system. Traditional development methods do not permit the development of maintainable systems. Formal engineering methods and standards for developing the system improve the maintainability of the system.

Documentation

Design and implementation decisions are usually not captured in any form for future maintainers. The developer's knowledge and expertise gained through the development process could be very helpful during maintenance and could be transferred to the maintainers of the system through efficient documentation.

Definitions

Inconsistent terminology has resulted in a miscommunication of what reengineering is and the capabilities that this technology will provide for computer system developers and maintainers. This problem exists in many kinds of environments including the software engineering environment and is a considerable problem when a new technology such as reengineering is in its infancy. The definitions provided in the beginning of this report attempt to clarify some of the terminology currently used in the field of reengineering and will be used consistently throughout this report.

The general definition of reengineering describes the alteration of an existing system to reconstitute it in a new form, and then the actual generation of this new form. This definition does not specify what the alterations are, nor how they should be performed. It also does not define what the new form is, nor how this new form should be generated. These questions were addressed by the research summarized in this report. This definition should be expanded to include additional information based on the engineering environment in which reengineering will be applied. The generality of the definition has resulted in various interpretations of how reengineering is accomplished. The research at NAVSWC defined the capabilities of reengineering and outlined a methodology for reengineering Navy tactical computer systems.

Applicability

Reengineering is not always appropriate in every maintenance or development process. This fact should not be considered a weakness of reengineering. By realizing the applicability of the technology, the best results from reengineering will be achieved. The technology can provide a wide range of benefits when weighed against alternative solutions under considerations of manpower, time, and cost.

The enormous number of computer systems existing today increases the probability that a proposed system is very similar to an existing system. Using this existing information efficiently is essential to adequately maintain these systems and improve future development of similar computer systems.

REFERENCES

1. Andrews, Dorine C., "Systems Re-engineering: A Critical Perspective," CASE Trends Magazine, Jul/Aug 1990, pp. 15-16.

2. Bohner, Shawn A., "Re-engineering: A Technology Perspective," Contel Technology Center, 15000 Conference Center Drive, Chantilly, VA, Naval Surface Warfare Center Reengineering Workshop Proceedings, Silver Spring, MD, 21-22 Feb 1990.

3. Chikofsky, Elliot J. and Cross II, James H., "Reverse Engineering and Design Recovery: A Taxonomy," IEEE Software, Jan 1990, pp. 13-17.

4. Johnson, P.E., Heisler, K.G. and Snyder, J.R., "Mental Models for Software Maintenance," University of Minnesota, Minneapolis, Minnesota, Naval Surface Warfare Center Reengineering Workshop Proceedings, Silver Spring, MD, 21-22 Feb 1990.

5. Boehm, Barry W., "A Spiral Model of Software Development and Enhancement," ACM SIGSOFT Software Engineering Notes, Vol. 11, No. 4, Aug 1986.

6. McCabe, Thomas and Butler, Charles, "Design Complexity Measurement and Testing," Communications of the ACM, Dec 1990, pp. 1415-1425.

7. Halstead, Maurice, Elements of Software Science, New York, Elsevier North-Holland, Inc., 1977.

8. IEEE Std 982.1-1988, IEEE Standard Dictionary of Measures to Produce Reliable Software.

9. Prywes, Dr. Noah and Ge, X., "Software Intensive Systems Reverse Engineering," Computer Command and Control Company, Philadelphia, Pennsylvania, Naval Surface Warfare Center Reengineering Workshop Proceedings, Silver Spring, MD, 21-22 Feb 1990.

10. Guttag, J.V., Horning, J.J. and Wing, J.M., "The Larch Family of Specification Languages," IEEE Software, 1985, pp. 24-36.

11. Caine, S. and Gordon, K., "PDL--A Tool For Software Design," Proceedings National Computer Conference, 1975, pp.271-276.

12. Hatley, Derek J. and Pirbhai, Imtiaz A., Strategies for Real-time System Specification, 1987.

13. Booch, Grady, "Object-Oriented Development," IEEE Transactions Software Engineering, Vol. SE-12, No. 2, Feb 1986, pp. 211-221.

14. Chen, Peter, "The Entity Relationship Model--Toward a Unified View of Data," ACM Transactions on Data Base Systems, Vol. 1, No. 1, Mar 1976, pp. 9-36.

15. DeMarco, Tom, Structured Analysis and System Specification, Englewood Cliffs, N.J., Prentice-Hall, 1979.

16. Yourdon, Edward and Constantine, Larry L., Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design, Second Edition, Englewood Cliffs, N.J., Prentice-Hall, 1978.

17. Jackson, Michael A., Principles of Program Design, New York, Acedemic Press, 1975.

18. Gane, Chris and Sarson, T., Structured Systems Analysis: Tools and Techniques, New York, Improved Systems Technologies Inc., 1977.

19. Ward, Paul T. and Mellor, Stephen J., Structured Development for Real-Time Systems, Englewood Cliffs, N.J., Prentice-Hall, 1985.

20. Page-Jones, Melir, The Practical Guide to Structured Designs, Second Edition, Englewood Cliffs, N.J., Prentice-Hall, 1988.

APPENDIX A

COMPUTER-AIDED SOFTWARE/SYSTEM ENGINEERING PRODUCTS

This appendix lists Computer-Aided Software/System Engineering (CASE) products which provide reverse engineering capabilities or are the focus of reverse engineering integration research today.

ADVANCED SYSTEMS TECHNOLOGIES, INC.
Englewood, CO 80112

PerSpective is a system level specification and analysis tool for evaluating the performance and reliability of computer hardware, software, and data design alternatives.

AGS MANAGEMENT SYSTEMS, INC.
King of Prussia, PA 19406

FirstCASE is an integrated development support environment which tightly couples proven life cycle methodologies, automated estimating tools and a comprehensive project management system into a distributed cooperative architecture.

AMERICAN MANAGEMENT SYSTEMS, INC.
Arlington, VA 22209

AMS relies heavily on CASE to make it one of the nation's top systems development firms. AMS offers a complete suite of consulting services to I/S organizations ranging from CASE Readiness & Implementation Planning to custom systems development.

ANDERSEN CONSULTING (primarily MIS)
Chicago, IL 60602

FOUNDATION provides a complete Computer-Aided Systems Engineering environment that controls and automates the tasks necessary to build and deliver quality systems that address the unique business problems of an organization. It is composed of an integrated set of software facilities that support the entire systems development life cycle from information planning through production systems support.

APLAN INFORMATION SERVICES, INC.
Newport Beach, CA 92660

ODYSSEY is an innovative, group development solution addressing the need of corporations for integrated information. Its use guarantees completion of high quality system specifications while ensuring key personnel concurrence and satisfaction. This process merges structured techniques, rapid development concepts, facilitation services, methodology contents, and software.

ATHENA SYSTEMS, INC.
Sunnyvale, CA 94086

Rapid prototyping software/simulation and modeling.

BACHMAN INFORMATION SYSTEMS (primarily MIS)
Burlington, MA 01803

Bachman Information Systems develops and markets software to facilitate maintenance, enhancement, migration and co-existence of existing information systems, as well as to develop new ones. BACHMAN's products, part of IBM's AD/Cycle solution, were developed for use in the IBM-MVS mainframe environment to apply the power of database technology to the tasks performed by many information systems professionals.

CADRE TECHNOLOGIES, INC.
Providence, RI 02903

Cadre provides workstation-based solutions for automating the analysis, design and code generation of high performance systems and software. Cadre's products provide comprehensive support for all stages of product development from hardware/software simulation and requirements definition through integration and test. The products are based on Open Architecture standards that provide seamless interfaces to existing software. Cadre works in cooperation with Microcase to provide tools which perform analysis of software applications.

CASE CONSULTING GROUP, INC.
Lake Oswego, OR 97035

CCG is dedicated to providing information products to assist in CASE technology transfer. Products include CASE OUTLOOK, a two-volume CASE BUYERS GUIDE and THE MANAGER'S GUIDE TO CASE, an overview of CASE technology trends. Consulting services include - software technology assessment, CASE strategy formulation and implementation, market research and competitive analysis.

CASE TRENDS
Worcester, MA 01605

CASE Trends, the magazine for CASE in practice, is an authoritative source for computer systems professionals who want to learn how to take advantage of the latest developments in software automation.

CASE STRATEGIES
Arlington, MA 02174

The monthly CASE Strategies Newsletter is a practical, hands on tool for Information Services Executives, MIS/DP Managers and anyone involved in design, development or implementation of software systems.

CASEWARE, INC.
Costa Mesa, CA 92626

CaseWare develops CASE products such as AMPLIFY CONTROL and provides UNIX professional services. AMPLIFY is a graphic development environment and Configuration Management system with an open architecture for tool integration.

CHEN & ASSOCIATES
Baton Rouge, LA 70808

Chen & Associates provides one of the most comprehensive tool sets for data modeling and database design. CHEN CASE workbench supports forward engineering (DFDS, ERDS, normalization, schema generation), reverse-engineering and linkages to data dictionaries and other CASE tools.

COMPUTER SYSTEMS ADVISORS, INC.
Woodcliff Lake, NJ 07675

POSE (Picture Oriented Software Engineering) is the first affordable CASE solution addressing the full development life cycle. POSE consists of eleven integrated and modular PC-based modules that support both data-driven and process-driven methodologies. The newest modules: Screen Report Prototyper, Planning Matrix Diagrammer, and Source Code Generator, form the basis of a complete life cycle systems designer's workbench.

COMPUTROL, INC.
Chesterfield, MO 63017

Master Financial System is a CASE tool, based on reusability, that represents a totally new approach to software technology - one that preserves existing application code and dramatically reduces development and maintenance time.

ECS-CASE TOOLS & SERVICES
Torrance, CA 90505

ECS provides leading CASE tools, training and services for developers that address the VMS, UNIX PC, and OS/2 environments. Selected tools and services cover the entire life cycle of the development process, including front-end CASE support, project and configuration management, electronic publishing for cost reduction, and support for reverse engineering.

EVERGREEN CASE TOOLS
Redmond, WA 98052

EasyCASE Plus is an inexpensive, easy to use, full featured CASE tool which addresses systems and software analysis, design and data modelling.

FUTURE TECH SYSTEMS, INC.
Auburn, WA 98002

Envision 2.0 provides a significant advance over today's existing CASE Environments by addressing the following engineering concerns: ease of use, extensibility, performance, and support for team activities. Envision 2.0 operates on a PC platform executing under Microsoft Windows.

ICONIX SOFTWARE ENGINEERING, INC.
Santa Monica, CA 90405

Iconix Power Tools is an extensive set of CASE modules which span analysis, design, coding and management of complex software systems and cover a much greater portion of this software lifecycle than competitive products. Power Tools supports the DeMarco method of structured analysis, along with realtime extensions proposed by Ward-Mellor and Hatley.

INFORMATION ENGINEERING SYSTEMS CORPORATION
Arlington, VA 22202

Information Engineering Systems Corporation (IESC) provides training, consulting and software in support of the Information Engineering methodology. IESC was founded by Clive Finkelstein, sometimes referred to as the "Father of Information Engineering" and is widely recognized as the technological and market leader in the Information Engineering field.

INGRES CORPORATION
Alameda, CA 94501

Ingres Corporation is a worldwide leader in quality information integration products and solutions for mission critical applications. The company's primary product is the INGRES RDBMS, which enables organizations to quickly access data spanning a variety of computer systems connected in a

network. The INGRES product suite is available for all strategic applications and gateways for accessing data across heterogeneous micro, mini, and mainframe computing environments.

INTERACTIVE DEVELOPMENT ENVIRONMENTS, INC.
San Francisco, CA 94105

Software through Pictures (StP) from Interactive Development Environments, Inc., is an integrated, workstation-based multi-user CASE environment aimed at large-scale project development. StP comprises a family of products that support software development from requirements analysis through code generation.

INTERLEAF, INC.
Cambridge, MA 02141

Interleaf's Technical Publishing Software (TPS) is widely used as a documentation tool with CASE environments. Highly interactive and easy-to-use, TPS simplifies the production of compound structured documentation, such as that required by DOD-Std-2167. Interleaf's booth will also feature a CASE "IntegrationFest." Participating in this demonstration will be: Atherton Technology, Cadre Technologies, i-Logix, Interactive Development Environments (IDE), Mark V Systems, Ready Systems, Teledyne Brown Engineering and Verilog USA.

INTERPORT SOFTWARE CORPORATION
Burke, VA 22015

InterPort Software products allow software engineers to maintain their investment in utilizing applications through advanced reverse engineering technology. Its Reverse Engineering Workbench and Migration Workbench allow the migration of both data and process models, including JCL and mapsets, to different hardware platforms, CASE repositories, client defined dictionaries and a standard structure database supplied with the product.

JAMES MARTIN ASSOCIATES
Reston, VA 22091

James Martin Associates pioneered development of an Information Engineering Methodology (TM) which allows corporations to link information systems development with the organization's business and its objectives. JMA provides support for this methodology with a public seminar program, a CBT module series and hypermedia documentation of the IEM, in addition to the company's direct consulting services.

KNOWLEDGEWARE, INC. (primarily MIS)
Atlanta, GA 30326

The KnowledgeWare tool set is, according to James Martin, "the world's most comprehensive CASE tool set." KnowledgeWare provides a complete, Integrated Computer-Aided Software Engineering (I-CASE) environment for the planning analysis, design, construction, and maintenance of information systems. With KnowledgeWare tools, users are building higher quality information systems in less time.

LBMS, INC.
Houston, TX 77092

LBMS provides an automated methodology covering all aspects of the systems development life cycle including strategic planning, implementation and maintenance. LBMS products include the Strategic Planner, System Engineer, and Automate Plus CASE tool set. This tool set provides full support for the development life cycle from strategic planning through to the generation of physical database designs in DB2, Oracle, IDMS and Adabas.

META SYSTEMS, LTD
Ann Arbor, MI 48104

Meta Systems, Ltd announces a repository-based CASE system that runs on IBM PCs and, for the first time, embeds the functional power of a mainframe CASE repository within the PC itself. The active repository in Meta's new product, Structured Architect Workbench, provides system developers with levels of intelligence previously reserved for a large-scale host computer repository.

METIER MANAGEMENT SYSTEMS-ARTEMIS
Irving, CA 92714

Metier provides software developers with new methods, techniques and tools to plan and control projects using the worlds most popular project management software development products.

OPERATIONS TECHNOLOGY CORPORATION
Marlboro, MA 01752

Operations Technology Corporation provides consulting, training and implementation of CASE tools and structured methodologies, as well as joint User Application Development (UAD) using CASE.

ORACLE CORPORATION
Belmont, CA 94002

Oracle Corporation markets a complete line of CASE products, including CASE Dictionary - a shared repository, CASEDesigner - a graphical workbench, and CASE Generator - a comprehensive suite of application generators.

P-CUBE CORPORATION
Brea, CA 92621

P-Cube Corporation is a leading developer of software tools for Information Services (IS) managers. Its products support a broad range of management activities from strategic planning for information technologies to measuring the contribution of the IS function to organizational objectives.

PANSOPHIC SYSTEMS, INC.
Lisle, IL 60532

TELON is the most widely used application code generation technology for the IBM environment. TELON generates stand-alone (COBOL or PL/1) programs from design-level specifications. TELON/TEAMWORK is the first full function CASE offering that exploits the powerful capabilities of IBM's OS/2. Pansophic's Life Cycle Manager (PAN/LCM) is a workstation-based version manager that provides change control and configuration management.

POPKIN SOFTWARE & SYSTEMS, INC.
Stamford, CT 06901

System Architect is a PC CASE tool that runs under Microsoft Windows. Methodologies supported include Yourdon/DeMarco, Gane/Sarson, Ward-Mellor (real-time), Decomposition, Object-Oriented Design, Structure Charts, Entity Relationships, and State Transition diagrams. System Architect's data dictionary/encyclopedia is integrated with its diagramming facilities and includes the powerful "User Definable Attribute" (metadata) facility.

PRENTICE HALL-YOURDON PRESS
Englewood Cliffs, NJ 07632

Prentice Hall is the leading publisher in the area of CASE. Yourdon Press titles include Object-Oriented Analysis, Structured Development for Real-Time Systems, and Structured Analysis and Systems Specifications. Other titles on display: James Martin's three-book series, Information Engineering, and Carma McClure's CASE is Software Automation.

SOFTLAB, INC.
Rosemont, IL 60018

Maestro 11 is an open software development environment, built on a multi-user object-oriented database, which fully incorporates multi-phase and

life cycle functions. It integrates, manages and controls front- and back-end tools and human resources for large-scale software development projects, targeted for multiple-host production environments.

SOFT-SET TECHNOLOGIES INC.
Vancouver BC V6J 1Y6 Canada

Soft-SET's product Aranda is a new CASE application that focuses on understanding programs that are already written and on providing support for code reuse and maintenance. Aranda produces automatic identifier documentation from existing source code and provides a facility to integrate all project management documentation, including design specifications, into a single linked database.

SOFTWARE AG, INC.
Reston, VA 22091

Software AG provides full life cycle, integrated CASE solutions for IBM, VAX, and Wang environments.

SOFTWARE PRODUCTIVITY RESEARCH, INC.
Cambridge, MA 02110

Use CHECKPOINT expert CASE tool for estimation, measurement and assessment of software projects. SPOR/20 is Software Productivity's cost estimation tool.

SYNTHESIS COMPUTER TECHNOLOGIES INC.
Long Beach, CA 90804

Syntek case/ap is a powerful CASE software tool designed to support the rapid prototyping software development methodology. The system provides an integrated development environment where the analyst can specify both the system's data structure and the required application processing, utilizing carefully engineered, user-friendly interfaces.

SYSCORP INTERNATIONAL
Austin, TX 78759

MicroSTEP is a computer aided software engineering (CASE) tool that produces 100% C source code and executable programs directly from graphic specifications. It improves software development productivity through four unique features: an interactive graphic design environment, automatic specification analysis, generation of executable code, and production of high quality technical documentation.

TEXAS INSTRUMENTS
Plano, TX 75085

The Information Engineering Facility (IEF) CASE product automates the complete systems development lifecycle. It consists of powerful, graphically-based toolsets. All components are available today: strategic planning, analysis, design and construction. Texas Instruments was the first vendor to demonstrate 100% code generation from diagrams and is the leading integrated CASE vendor today for business applications.

VERILOG USA, INC.
Alexandria, VA 22312

Verilog provides tools for addressing the life cycle through requirements engineering, system testing using rapid prototyping, and test case generation features. GEODE, a real-time systems design tool, includes syntax-oriented graphic editor, check simulator/debugger, and code generator. LOGISCOPE, a source code analyzer, provides complexity analysis and test coverage analysis.

VIRTUAL PROTOTYPES, INC.
Montreal, Quebec H4A-3S5 Canada

VAPS is a streamlined software tool used in control and display technology in Man-Machine Interface applications. VAPS runs on a graphics workstation in the UNIX operating system, and incorporates the features of CAD/CAM, CAE and CASE, as well as automatic code generation.

VISIBLE SYSTEMS CORPORATION
Waltham, MA 02154

Visible Systems Corporation is a leader in the development of micro computer-based CASE products. Its principal product, the Visible Analyst Workbench, is a powerful yet easy-to-use systems analysis and design tool.

XA SYSTEMS CORPORATION
Los Gatos, CA 90530

XA Systems Corporation delivers an integrated set of tools and services that automate or assist in the application design, development, testing and maintenance processes. XA Systems addresses application life cycle capabilities for existing systems, as well as upper and lower-level CASE products for forward engineering.

YOURDON-A KODAK COMPANY
Raleigh, NC 27615

Yourdon's CASE products include the Analyst/Designer Toolset for PCs and Cradle for UNIX workstation platforms. All Yourdon CASE products support the Yourdon Structured Method for real-time and commercial systems, with integrated support of object-oriented methods.

Technical information on the following items can be obtained directly from the manufacturer.

CASE 2000
Nastec Corporation
Southfield, MO 48075

KeyOne and Key Lab
LPS
10124 Torino, Italy

PROMOD
ProMod, Incorporated
El Toro, CA 92630

Ready Systems
Sunnyvale, CA 20770

STATEMATE
i-Logix Corporation
Burlington, MA 01803

SuperCASE
Advanced Technology International, Inc. (ATI)
New York, NY 10036

TAGS
Teledyne Brown Engineering
Huntsville, AL 35807

VAXset and DECDesign
Digital Equipment Corporation (DEC)
Pre-Sales Technical Support
1-800-842-5273

APPENDIX B

REENGINEERING PRODUCTS AND MANUFACTURERS

This appendix lists some of the commercial products currently available that support reengineering. Many products currently available address only COBOL source code and are not applicable to Navy tactical computer systems, but are included in the following listing simply for reference.

SUPERSTRUCTURE
Group Operations, Inc.

Superstructure is a COBOL restructuring automated support product that turns spaghetti code COBOL programs into Structured COBOL. Superstructure turns existing code into a series of performed modules with a single entry and single exit point. It eliminates interparagraph GOTOs, eliminates fall throughs between modules, converts dead code to comments, corrects PERFORM range violations, and removes ALTER statements and altered GOTOs.

SPAG - THE SPAGHETTI UNSCRAMBLER
Polyhedron Software Limited

SPAG is a FORTRAN reconditioning tool designed primarily for use with aging FORTRAN spaghetti code. SPAG analyzes the bewildering jumble of GOTOS, arithmetic IFs, and computed GOTOS, which are characteristic of old-style FORTRAN, and rewrites the program using modern, structured programming constructs. SPAG accepts ANSI-standard FORTRAN (-66 or -77) with local extensions as input and produces a neatly indented and structured output which is exactly equivalent logically to the input spaghetti code.

PAC
Eckert Research International Corporation

PAC provides tools to analyze programs written in the C language. It collects information about a program's functions and global variables and produces reports based on that information in a format specified by the user. PAC reports include a modular tree showing the relationships of functions (how they are nested), a chart or table showing each call/use of each function, a chart or table showing the definition, substitution, and references for each global variable, and quantitative information on appearance of keywords, comments and standard 1/0 functions.

ADAGEN
Mark V Systems Limited

Adagen is a CASE tool tailored to Ada and MIL-SPECS. It supports requirements analysis DFDS, structure charts with Booch and Buhr notations, object oriented requirements and design approaches, and can be tailored to the user's own methodology. Adagen generates Ada code templates from Ada structure chart diagrams. It can also reverse-generate diagrams from the Ada source code.

ANALYSIS OF COMPLEXITY TOOL
BATTLEMAP ANALYSIS TOOL
McCabe & Associates, Inc.
Columbia, MD 21045

The Analysis of Complexity Tool (ACT) automates the structured testing methodology and aids maintenance and system re-engineering. It is driven by and analyzes source code, producing information about module structure, McCabe complexity metrics, and the basic set of test paths that should be exercised for each module within the source code. Outputs of ACT include flowgraphs of code, test paths, test conditions, annotated source listings, and the McCabe Complexity Metric.

The Battlemap Analysis Tool (BAT) is a software reverse engineering and maintenance utility. It uses structural charts of foot-print graphs to display the design structure of the physical code of a system or subsystem. BAT abstracts structure charts from source code and indicates complexity and reliability.

DOCGEN
Software Systems Design, Inc.

DocGen works in conjunction with the ADADL processor to automatically produce documentation required by military and DOD standards. DocGen and ADADL analyze the Ada source code and any Ada/PDL pseudo-code to determine as much information as possible from the Ada source file. The analysis of the Ada yields these sections of documentation automatically: inputs, outputs, local data, global data, interrupts, exceptions, calls, and called-by. The pseudo-code is used to determine algorithm and processing.

REVENGG
Advanced Systems Technology Corporation (ASTEC)

Revengg is a software package for abstracting the structure and interactions of ill-documented programming code to its specification language/dictionary system (e.g., PSL/PSA of Meta Systems, Ltd. or IRDS). With the program abstraction entered into the dictionary system, it can be examined and evaluated for inconsistencies and gaps in traceability. The resulting specification of the program can be designed for eventual documentation that conforms to any mandated documentation standards. At

present, Revengg can reverse engineer COBOL and FORTRAN source code to the CaMERA dictionary. Once the programs have been abstracted into CaMERA, the dictionary is used to navigate across the set of programs, examining the structures of the program, evaluating the usage of the data, and deciding how and where any changes could be made.

INTERCASE REVERSE ENGINEERING
Interport Software Corporation

Based in the C programming language, the InterCASE product extracts specifications, design information, and reusable components from existing software applications. Other components of InterCASE permit manipulation of this data and allow for the regeneration of source code.

REENGINEERING BUSINESS SYSTEMS
Rittersbach, G.H,
Peat, Marwick, Mitchell & Co.,

The cost of developing replacement business systems is astronomical. The choices of software packages or custom development seem to be multiyear, multimillion-dollar, and multirisk projects. Automated tools can substantially reduce the time and cost of producing target systems through reengineering of in-place applications. The reengineered product generates a building-block prototype that can be extended functionally and technically. Users of this approach claim significant improvement in building the systems they need.

XINOTECH PROGRAMMING ENVIRONMENT INCLUDING THE DESIGN ABSTRACTOR
Xinotech Research
Minneapolis, MN

The reverse engineering capabilities of the Xinotech Programming Environment will support transformations between various language constructs as well as the capability to abstract control flow in a language independent fashion. The combination of these features may make it possible to turn spaghetti code into structured code in FORTRAN as well as in other languages.

The Xinotech Design Abstractor will abstract the design information from the source code and comments into a PDL. The PDL generated by the Abstractor will be user-specifiable; it will be possible to generate a PDL, such as ADADL, which supports documentation generation.

APPENDIX C

INDUSTRY PARTICIPANTS IN REENGINEERING

Many companies are currently active in reengineering technology. The following list of companies provide products or services which support reengineering. Specific information on their products was not available at the time of this publication, but is designated for further research.

Adpac Corporation
340 Brannan
San Francisco, CA 94107
415-974-6699
Product: PM/SS

A+ Software Inc.
16 Academy St
Skaneateles, NY 13152
315-685-3928
Product: ADS/MVS

ADR
Rt. 206 & Orchard Rd.
Princeton, NJ 08540
201-874-9000

Advanced Computer Concepts
8051 N. Tamiani Trail #28
Sarasota, FL 34243
813-355-0450
Product: APDL

Advanced Software, Inc.
1095 East Duane Ave
Sunnyvale, CA 94086
408-733-0745
Product: DocuComp

Advanced Systems Concepts
1350 Remington Rd.
Schuamberg, IL 60195
312-310-1881
Product: Probe

A. K. Inc.
P.O. Box 7457
San Jose, CA 95150
408-374-4663
Product: Trace

AGS Management Systems Inc.
880 First Ave
King of Prussia, PA 19406
215-265-1550
Product: SDM

Aldon Computer Group
405 14th Street
Oakland, CA 94612
415-839-3535
Product: Analyzer

Allen Systems Group
35 Rockridge Rd
Englewood, CO 45322
513-832-0154
Product: SHOPMON

American Management Systems
1777 N. Kent St
Arlington, VA 22209
703-841-6000
Product: LPS

Application Development Systems, Inc.
6840 78th Ave North
Minneapolis, MN 55445
612-560-8633
Product: XPEDITER

Application Development Services
25371 Hugo Rd.
Laguna Niguel, CA 92677
714-495-6321
Product: Docu/master

Applications Programming Company
11 W 2nd St
Moorestown, NJ 08057
609-234-0099

Asyst Technologies Inc.
1680 Beaver Hall
Montreal, QB
Canada H2Z 1S8
Product: Integrated Workstation

AUSTEC, INC
1740 Technology Dr.
San Jose, CA 95110
408-279-5533
Product: RM/FORTRAN
RM/COBOL

B I Moyle Assoc
5788 Lincoln Dr.
Minneapolis, MN 55436
612-933-2885
Product: BIMWINDOW

BBN Software Products Inc.
10 Fawcett St.
Cambridge, MA 02238
617-873-8199
Product: RPL Toolkit

Bachman Information Systems
4 Cambridge Center
Cambridge, MA 02142
617-354-1414
Product: Re-engineering Product Set
Bachman Data Analyst
Bachman Data Administrator

BlueLine Software Inc.
1500 S. Lilac Dr. #240
Minneapolis, MN 55146
612-542-1072
Product: RDShare

Blackhawk Data Corporation
307 N Michigan Ave.
Chicago, IL 60601
312-236-8473
Product: Hawkeye

BMC Software Inc.
Box 2002
Sugar Land, TX 77487
713-240-8800

Britz Publishing
1814 Capital Towers
Jackson, MS 39201
601-354-8882
Product: INDOC

Business Software Technology, Inc.
114 Turnpike Rd.
Westborough, MA 01581
617-870-1900
Product: ENDEVOR

Candle Corp
1999 Bundy Drive
Los Angeles, CA 90025
213-207-1400
Product: OMEGAMON

CAP Gemini
1133 Avenue of the Americas
New York, NY 10036
212-221-7270

CATALYST
303 E Wacker Drive
Chicago, IL 60601
312-938-5352

CGI Systems
P.O. Box 1645
Pearl River, NJ 10965
914-735-5030
Product: Pacbase

Cincom
2300 Montana Ave.
Cincinatti, OH 45211
513-662-2300
Product: NetMaster

Cinnabar Software
2704 Rio Grande
Austin, TX 78705
512-477-3212
Product: Checkmate

Clarity Concept Systems
14 Washington Place
New York, NY 10003
212-254-3358
Product: ENFORCER I

Computer Application Services
15560 Rockfield Blvd
Irvine, CA 92718
714-859-2274
Product: Abend-catcher

Communications Sciences Inc.
100 N 7th St.
Minneapolis MN 55403
612-332-7559

Computer Corporation of America
4 Cambridge Center
Cambridge, MA 02142
617-492-8860
Product: Symdump

Computer Associates
711 Stewart Ave.
Garden City, NY 11530
516-227-3300
Product: CA-Transit
CA-OPTIMIZER
TOPSECRET
MetaCobol/QA
Look
Librarian
CA-UNIPACK/PPS
CA-OPTIMIZER/CMO
Autoflow
ACF2

Computer Concepts, Inc.
6443 SW Beaverton Highway
Portland, OR 97221
503-297-3567

Computer Task Group
800 Delaware Ave.
Buffalo, NY 14209
716-882-8000
Product: Conversion services

Compuware
31440 Northwestern Highway
Farmington Hills, MI 48018
313-737-9353
Product: File-Aid
CICS Radar
CICS Playback
CICS Debug-Aid
CICS Abend-Aid
ABEND-AID

Corinthian Software, Inc.
590 Commerce Park Dr. #150
Marietta, GA 30060
404-499-7399
Product: Sourceplus

Consumer Systems
2 East 22nd St.
Lombard, IL 60148
312-495-8822
Product: JCLflow

David R. Black & Assoc.
PO Box 335
Munroe Falls, OH 44262
216-688-2741
Product: COBOL Converter

Data Administration Inc.
301 N Harrison St. #203
Princeton, NJ 08540
609-924-0471
Product: Data Expediter

DAPREX, INC
2001 W Main St. #220
Stamford, CN 06902
203-324-2474
Product: Flowcharter

DABrask Systems Inc.
4805 Pershing Rd.
Downers Grove, IL 60515
312-971-3081

Davis, Thomas & Assoc.
8800 Highway 7
Minneapolis, MN 55426
612-938-7669

Digital Equipment Corporation
Product: VaxSet

DBMS Inc.
1801 Mill St
Naperville, IL 60540
312-961-5700
Product: TP Sessions

DTA, Inc.
505 N. County Rd. 18
550 Waterford Park
Minneapolis, MN 55441
612-591-6100
Product: DTA/COBOL

Diversified Software Systems
996 Minnesota Ave.
San Jose, CA 95125
408-998-0414
Product: JOB/SCAN

Dynamics Research Corp.
60 Frontage Rd.
Andover, MA 01810
508-475-9090
Product: Adamat

Duquesne Systems Inc.
Two Alleghney Center
Pittsburgh, PA 15212
412-323-2600 P
Product: TPX

Foundation For Software Engineering
4339 N 39th St.
Phoenix, AZ 85018
602-955-1148
Product: ReadCOBOL

Eden Systems Corporation
1180 Medical Court
Carmel, IN 46032
317-848-9600
Product: Q/Auditor

General Electronics
Box 79
Lyons, IL 60534
312-447-2797

General Research Corp
5383 Hollister Ave.
Santa Barbara, CA 93111
805-964-7724
Product: RXVP80

General Services Administration
FSMSC, PWB Services Group
5203 Leesburg Pike
Falls Church, VA 22041
703-756-4500

Gimpel Software
3207 Hogarth Lane
Collegeville, PA 19426
215-584-4261
Product: Generic Lint

General Transformation Corporation
Box 10083
Berkeley, CA 94709
415-644-0702

Goal Systems International, Inc.
7965 North High St.
Columbus, OH 43235
614-888-1775
Product: Flee/VSE

GT Software Inc.
1111 Cambridge Sq.
Alpharetta, GA 30201
404-751-1400

Group Operations, Inc.
1100 Vermont Ave NW
Washington, DC 20005
202-887-5420
Product: SUPERSTRUCTURE

H & W Computer Systems International
Innovation Data Processing, Inc.
275 Paterson Ave.
Little Falls, NJ 07424
201-890-7300

H & W Computer Systems International
Box 15190
Boise, ID 83715
208-835-0336

Hypergraphics Corporation
Box 50779
Denton, TX 76206
817-565-0004

IBS CORP
4660 La Jolla Village Dr. #700
San Diego, CA 92122
619-452-6055
Product: Dump-edge

IBM Corporation
Armonk, NY
Product: RACF
PL/I Checkout Compiler
Interactive Debug
COBOL/SF

Information Processing Techniques Corp
1096 E Meadow Circle
Palo Alto, CA 94303
415-494-7500
Product: Fortran-lint
Lint-plus

Information Concepts
202-628-4400
1331 H St. NW
Washington, DC 20005

Innovation Data Processing, Inc.
Product: Eyewitness
Monitor

Interactive Solutions Inc.
53 W. Fort Lee Rd.
Bogota, NJ 07603
201-488-3708

I P Sharp Assoc
2 First Canadian Pl. #1900
Toronto, ON M5X 1E3
Canada
416-364-2910
Product: WSDOC

INWARE CORP
1100 5th Ave. S
Naples, FL 33940
813-649-6060
Product: DOC-AID

Interport Software Corp
6035 Burke Center Pkwy. #270
Burke, VA 22015
703-425-6425
Product: Intercase
Interport

International Market Resources Inc.
38 Garden Rd.
Wellesley, MA 02181
617-235-8830
Product: Graphasyst

Intermetrics Inc.
733 Concord Ave
Cambridge, MA 02138
617-661-1840
Product: XDB

KOALA DEVELOPMENT
1700 Sunset Dr. #101
Longwood, FL 32750
407-330-1704
Product: AutoDoc

KnowledgeWare
3340 Peachtree Rd. NE #1100
Atlanta, GA 30026
404-231-8575
Product: IEW Workstation Tools
IEW Mainframe Knowledge Coordinator
Gamma

Kolinar Corporation
3064 Scott Blvd.
Santa Clara, CA 95054
408-980-9411

Kozo Inc.
114 Sansome St.
San Francisco, CA 95054
415-989-7223

Landmark
8000 Tower Crescent Dr.
Vienna, VA 22180
703-893-6842

Language Technologies
27 Congress St
Salem, MA 01970
617-741-1507
Product: Recorder
Inspector

Lattice Inc.
312-916-1600
2500 S Highland Ave.
Lombard, IL 60148

Lexeme
Product: Re-eng & Re-host

Macro 4, Inc.
Brookside Plaza
Mt. Freedom, NJ 07970
201-895-4345
Product: Dumpmaster

MacKinney Systems
2674 South Glenstone
Sprinceville, MO 685
417-882-8012
Product: Cobol Glossary

Manager Software Products, Inc.
131 Hartwell Ave
Lexington, MA 02173
617-863-5800 Product: Testmanager

Maintec
Product: Auditec

MB & Assoc.
2525 W Main St. #205
Littleton, CO 80120
303-794-1740
Product: Dataquick

Marble Computer, Inc.
101 Cloffton Dr.
Martinsburg, WV 25401
304-267-2941
Product: DCD II
CSA

MBP Software and Systems Technology Inc
1131 Harbor Bay Pkwy.
Alameda, CA 94501
415-769-5333

Meta Systems Ltd.
315 East Eisenhower Pkwy.
Ann Arbor, MI 48104
313-663-6027
Product: Reverse Engineering
PSL/PSA

Microfocus
2465 East Bayshore Rd.
Palo Alto, CA 94303
415-856-4161
Product: Micro Focus COBOL

Morino Assoc, Inc.
8615 Westwood Center Dr.
Vienna, VA 22180
703-734-9494
Product: TSO/MON
MICS

National Database Software, Inc.
5488 Centerbrook
West Bloomfield, MI 48033
313-851-5560
Product: H-CHART II

Network Concepts
2 Ridgedale Ave.
Cedar Knolls, NJ 07927
201-285-0202
Product: Control

NOI Systems
1877 Broadway
Boulder, CO 80302
303-447-0099

On-Line Software International
2 Executive Dr.
Fort Lee, NJ 07204
201-529-0009
Product: Verify
InterTest

Optima Software, Inc.
1010 Hurley Way
Sacramento, CA 95825
916-646-6800

Oregon Software
2340 SW Canyon Rd.
Beaverton, OR 97006
503-645-1150
Product: SourceTools

Panoramic
1340 Saratoga-Sunnyvale Rd.
San Jose, CA 94501
408-973-9700
Product: OnGuard
Connect

PBL Assoc
Product: TAPMap

Pansophic Systems Inc.
709 Enterprise Dr.
Oak Brook, IL 60521
312-986-6000
Product: XPF/COBOL
Panvalet

Performance Software, Inc.
575 Southlake Blvd.
Richmond, VA 23236
804-794-1012
Product: Track

Peat Marwick
303 E Wacker
Chicago, IL 60601
312-938-5367
Product: Silverrun
Retrofit
DataTEC
Pathvu

Pertaine Systems
805 Veterans Blvd. #110
Redwood City, CA 94063
415-367-9827
Product: EXPLAIN/3000

Polytron Corporation
1700 NW 169th Place
Beaverton, OR 97006
503-645-1150
Product: Poly Librarian

Productive Software Systems, Inc.
612-920-3256
5617 Countryside Rd.
Edina, MN 55436

Programart
1280 Massachusetts Ave.
Cambridge, MA 02138
617-661-3020
Product: STROBE

ProMod Inc.
23685 Birtcher Dr.
Lake Forest, CA 92630
714-855-3046
Product: ProMod

Proximity Technology Inc.
305-566-3511
3511 NE 22nd Ave.
Ft. Lauderdale, FL 33308

Quality Systems Development Corp.
875 N Michigan Ave.
Chicago, IL 60611
312-266-6068
Product: RESQ
RES-Q

Rand Information Systems, Inc.
1201 Harbor Bay Pkwy.
Alameda, CA 94501
415-769-9000
Product: TIME

Rapitech Systems Inc.
Montebello Corporate Park
Suffern, NY 10901
914-368-3000
Product: Fortrix-C
COBOLIX-C
Fortrix-ADA
Fortrix-C
Cobolix-C

Raxco
2440 Research Blvd.
Rockville, MD 20850
301-258-2620
Product: Rabbit-2

Realia, Inc.
10 S Riverside Plaza
Chicago, IL 60606
312-346-0642
Product: Realia COBOL

Related Computer Technology
Box 523
154 S Main St.
Keller, TX 76248

REM Assoc.
Box 527
Village Station
New York, NY 10014
212-243-2416
Product: REMDOC

Science Applications International
1213 Jefferson Davis Hwy.
Arlington, VA 22202
703-979-5910

Singer Dalmo Victor Division
6565 E Tanque Verde Rd.
Tucson, AZ 85715
602-721-0500

Softlab GmbH
188 The Embarcadero
San Francisco, CA 94105
415-957-9175

Softlab GmbH
Zamdorferstrasse 120
8000 Munchen 8 Germany
089 93001
Product: Maestro

Softool Corporation
340 S Kellog Ave.
Goleta, CA 93117
805-683-5777
Product: FORTRAN Documentor
COBOL Testing Instrumenters
COBOL Programming Environment
COBOL Documentor
CCC

Softron, Inc.
Box 26220
Austin, TX 78755

Software Concepts
404-659-1331
250 Piedmont Ave.
Atlanta, GA 30308

Software Professionals
Product: Enlighten

Software Research, Inc.
625 Third Street
San Francisco, CA 94107
415-957-1441
Product: TCAT

Software Technologies and Research
160 West St.
Cromwell, CT 06416
203-529-7128
Product: UTLX

Softworks, Inc.
7700 old Branch Ave.
Clinton, MD 20735
301-856-1892

Startech Software Systems, Inc.
80 Beaver Street
NY, NY 10005
212-943-9800
Product: CICS-TDS

Southwest Software Services, Inc.
817-792-3024
813 Great Southwest Parkway
Arlington, TX 76011

Sterling Software Dylakor Division
17418 Chatsworth St.
Granada Hills, CA 91344
818-366-1781

Sterling Software Systems Software Marketing Division
11050 White Rock Road
Rancho Cordova, CA 95670
916-635-5535
Product: COMPAREX

Syllogy Corporation
One University Plaza
Hackensack, NJ 07601
201-568-9700

Texas Instruments
PO Box 655012
Dallas, TX 75265
800-527-3500
Product: IEF

System Strategies, Inc.
225 W 34th St.
NY, NY 10001
212-279-8400
Product: EX3278

System Design & Development Corp
Product: DCATS

System Connections, Inc.
56 Pine St 12th Floor
NY, NY 10005
212-943-8790
Product: Changemaster

Syncsort, Inc.
560 Sylvan Ave.
Englewood Cliffs, NJ 07632
201-568-9700

Tektronix, Inc.
CASE Division
Box 14752
Portland OR 97214

TONE Software Corporation
1735 S. Brookhurst
Anaheim, CA 92804
714-991-9460
Product: TONE

Transform Logic
8502 E Via De Ventura
Scottsdale, AZ 85258
602-951-8179
Product: Transform
TLC/Reveal

Travtech Inc.
1 Tower Square
Hartford, CT 06183
203-277-9595
Product: Scoreboard
TRAPS
Analyzer

Triangle Software Company
408-554-8121
4340 Stevens Creek Blvd.
San Jose, CA 95129

Unisys
Product: InfoGuard

United Security Software

University Computing Services Company
245 E Main St
Newark, DE 19711
302-368-0508
Product: Interactive COBOL Debugger

Verilog USA Inc.
6303 Little River Turnpike
Alexandria, VA 22312
703-354-0371

Viasoft
3033 N 44th St.
Phoenix, AZ 85018
602-952-0050
Product: VIA/INSIGHT
SMART/TEST

VM Systems Group, Inc.
901 S Highland St.
Arlington, VA 22204
703-979-3952
Product: XDebug

VM Software Inc.
1800 Alexander Bell Dr.
Reston, VA 22091
703-264-8018
Product: VM/Secure

Westinghouse Management Systems
PO Box 2728
Pittsburgh, PA 15228
412-256-2900
Product: MULTSESS

XA Systems
983 University Ave.
Los Gatos, CA 95030
408-395-1800
Product: TEST-XPERT
IMS-XPERT
Data-XPERT

ZyLAB Corporation
233 E Erie St.
Chicago, IL 60611
312-642-2201

APPENDIX D

REENGINEERING RESEARCH AND AUTOMATED SUPPORT PRODUCTS

This appendix lists various reengineering research projects underway in universities, industry, and government agencies. Available information on each project is given below. Some of these efforts are described in technical reports that are available to the public; an abstract of the technical report is given where applicable.

THE C INFORMATION ABSTRACTOR

Yih-Farn Chen
C.V. Ramamoorthy
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, CA 94720

Program understanding is one of the most time-consuming processes in software maintenance. This is partially due to the human inability to memorize complex interrelations among the software entities of a large software system. The basic idea in information abstraction is to extract relational information among the software entities of programs, store the information in a database, and make it available to users in a form that can be easily understood. We have implemented an information abstractor to extract relations information from C programs and store it into a program database. High level access utilities are provided so that program maintainers or developers can easily retrieve the information they need. Besides program understanding, the availability of the program database can also promote the research in four software engineering areas: multiple software views, software reusability, software metrics, and software restructuring.

AN OBJECT-ORIENTATED MAINTENANCE ORIENTED MODEL FOR SOFTWARE

K.G. Heisler
W.T. Tsai
P.A. Powell
Computer Science Department
University of Minnesota
Minneapolis, MN 55455

This paper introduces an object-oriented model of software that is derived semi-automatically from the actual code. The model and its associated object methods are used to support basic tasks required during software maintenance. The object-oriented model provides a series of successively more abstract views of the code structure. Object methods provide an approximation to the functionality (specification) of the software as well as a two-way map between the functionality and the code structure. Object methods also provide the means for detailed connectivity analysis associated with the ripple effect and program slicing types of activites during maintenance.

SOFTWARE MAINTENANCE PERSPECTIVE

W.T. Tsai
B.F. Chavez
S. Chen
K.G. Heisler
Computer Science Department
University of Minnesota
Minneapolis, MN 55455

Software maintenance has now become the most costly part of the software life cycle and methods are urgently needed to overcome the expense. This paper discusses maintenance activities and some of the current techniques addressing this problem.

IMPLEMENTING RELATIONAL VIEWS OF PROGRAMS

Mark A Linton
Computer Systems Laboratory
Department of Electrical Engineering
Stanford University
Stanford, CA 94305-2192

Configurations, versions, call graphs, and slices are all examples of views, or cross-sections, of programs. To provide a powerful mechanism for defining, retrieving and updating theses views, the OMEGA programming system uses a relational database system to manage all program information. We have built a prototype implementation of the OMEGA-database system interface. This implementation includes the design of a relational schema for a Pascal-like language, a program for taking software stored as text and translating it into the database representation, and a simple, pointing-oriented user interface. Initial performance measurements indicate that response is too slow in our current environment, but that advances in database software technology and hardware should make response fast enough in the near future.

AN APPROACH TO MEASURING DATA STRUCTURE COMPLEXITY

W.T. Tsai
M. A. Lopez
V. Rodriguez

D. Volovik
University of Minnesota
Department of Computer Science
Minneapolis, MN 55455

Most proposed software metrics concentrate on source code so they are available late in the software life cycle. However, it is important to have estimates of functionality and data complexity as early as possible. Furthermore, many software design methodologies start by modeling the system data. Thus, there is a need for metrics to express the characteristics of the system data in objective and quantitative terms. Data structures have static and dynamic components. In this paper, we propose a metric to measure the complexity of the dynamic aspects of the data; we also develop an algorithm to compute it. The metric proposed conforms to intuition and is based on mathematical foundations. We also discuss the strengths and limitations of the proposed data metric.

SUPPORT FOR REUSABILITY IN GENESIS

C.V. Ramamoorthy
Vijay Garg
Atul Prakash
Computer Science Division
University of California
Berkeley, CA 94720

Genesis is a software engineering-based programming environment geared to support big software projects. We first discuss a reusability-driven development methodology that advocates software development based on reusability considerations. Then, we discuss the automated support products and techniques provided in Genesis to support this methodology. We suggest techniques for improving retrievability, composabiltiy and understandability of software resources. Retrievability is improved by use of Entity Specification Language for tying resources through attributes and relations. Composability is improved through a mechanism called Functional Composability that is more general than UNIX pipes for composing programs. Understandability is improved by use of program abstraction.

THE REALIZABLE BENEFITS OF A LANGUAGE PROTOTYPING LANGUAGE

Robert M. Herndon
Valdis A. Berzins
University of Minnesota
Department of Computer Science
Minneapolis, MN 55455

The uses and advantages specifically for the description and construction of the translators are considered. The major features of the Kodiyak prototyping language are described. The Kodiyak language was designed to be comprehensive.

REUSE OF CLICHES IN THE KNOWLEDGE-BASED EDITOR

Richard C. Waters
Artificial Intelligence Laboratory
MIT
Cambridge, MA

The knowledge-based editor in Emacs (KBEmacs) is the current demonstration system implemented as part of the Programmer's Apprentice project. The purpose of the system is to experiment with a number of AI-based programming support capabilities. It is expected that most of these capabilities will appear in one form or another in the programming environments of the future. The most important of these capabilities is the reuse of cliches; i.e., the reuse of stereotyped algorithms and other pieces of programming knowledge. Using KBEmacs a programmer can build up a program rapidly and reliably by selecting various standard algorithms and combining them. Other key features of KBEmacs include a semantic, as opposed to syntactic, representation for programs and support for an assistant-like interaction between the system and the user.

PERFORMANCE ANALYSIS TIGHTENS CODE

R. Whatley
Digital Designs
USA

Software performance analysis (SPA) allows system developers to determine the efficiency, speed and compactness of software programs. Monitoring a program's behavior becomes very important when developing complex programs or when a real-time application demands minimized CPU time in particular program modules. SPA automated support products that execute on logic analyzers have been available during the last few years. However, these software analysis programs are now migrating to development systems. In this environment, SPA programs extract execution information from the real-time emulation of the target system, typically via an in-circuit emulator. Operating at a higher level, SPA draws on data collected in a trace buffer and converts that information into indices of program performance.

DESIGN OF ADA SYSTEMS YIELDING REUSABLE COMPONENTS -
AN APPROACH USING STRUCTURED ALGEBRAIC SPECIFICATION

S.D. Litvintchouk
Mitre Corporation
Bedford, MA

A.S. Matsumoto
ITT Advanced Technology Center
Stratford, CT

Experience with design of Ada software has indicated that a methodology, based on formal algebra, can be developed which integrates the design and

management of reusable components with Ada systems design. The methodology requires the use of a specification language, also based on formal algebra, to extend Ada capabilities.

PROGRAM TRANSLATION VIA ABSTRACTION AND REIMPLEMENTATION

Richard C. Waters
Artificial Intelligence Laboratory
MIT
Cambridge, MA

Essentially all program translators (both source-to-source and compilers) operate via transliteration and refinement. The source program is first transliterated into the target language on a statement by statement basis. Various refinements are then applied in order to improve the quality of the output. Although acceptable in many situations, this approach is fundamentally limited in the quality of output it can produce.

SOFTWARE REENGINEERING

Butler, C.W.
Hodil, E.D.
Richardson, G.L.
Department of Computer Information Systems
Colorado State University
Fort Collins, CO

Maintaining production systems can be expensive and time-consuming. One way to improve maintainability is through software reengineering, automated revision and redocumenting. A carefully selected tool-kit combined with the appropriate management skills are required for effective systems management.

GIVING APPLICATIONS NEW LIFE (SOFTWARE RE-ENGINEERING)

Patel, F.
Canadian Data Systems
Canada

Software re-engineering holds appeal for extending the life of applications. But there are other issues that should be considered before making a move. The term applies to a relatively new process through which applications written in either Assembler language or unstructured COBOL can be converted to structured COBOL through the use of automated conversion packages. This process can be used to rejuvenate applications which no longer meet the user's current business needs. This review addresses a number of issues, some of which can be addressed by a conversion to structured COBOL and others that cannot be resolved by re-engineering.

A DEMAND-DRIVEN, COROUTINE-BASED IMPLEMENTATION OF A NONPROCEDURAL LANGUAGE

C. A. P. Denbaum
Iowa University
Iowa City, IA

A nonprocedural language based upon the Lucid family of languages is described, and semantic models of the language's data and sequence control aspects are presented. These models provide a general conceptual framework for an operational understanding of Lucid-like nonprocedural languages; their utility is demonstrated by their use as the basis of a demand-driven, co-routine based implementation of ANPL.

FEASIBILITY ASSESSMENT OF JOVIAL TO ADA TRANSLATION JTTL

Ehrenfried, D. H.
Air Force Wright Aeronautical Laboratories
Wright-Patterson AFB, OH

The Ada program is part of a DoD policy change towards incrementally reducing the number of languages in use by the DoD from many to only a few and then eventually to just one: Ada. Jovial (J73) is the current Air Force standard language for use in the embedded application domain. One approach towards an earlier transition of all software written in Ada would be the development of an automatic J73 to Ada translation system. With the translation of all J73 software to Ada, J73 software development systems could be phased out of use, the cost of maintaining the J73 system could be recovered, and programmers would be freed earlier for their eventual transition to Ada. This paper will examine the feasibility and cost effectiveness of developing a J73 to Ada translation system.

GUIDE TO SOFTWARE CONVERSION MANAGEMENT

M. Skall
CRC Systems, Incorporated
Fairfax, VA

This guideline was developed to provide federal ADP managers a better understanding of the entire process of software conversion. Software conversions have life cycles with distinct phases and activities that occur in each phase. Understanding the order or sequence of a conversion and of the associated costs should help managers to plan and execute software conversions efficiently, effectively, and with minimum operations disruption to federal agencies. Although extensive references were consulted in preparing this guideline, the most important sources were interviews conducted at 14 federal agencies that had completed or were involved in software conversion projects. These interviews influenced the structure and organization of this guideline in an attempt to present, in logical order, activities that must be performed to achieve a successful conversion.

CONVERTING TO ADA PACKAGES

B. A. Wichmann
O. G. J. Meijerink
National Physical Laboratory
Division of Information Technology and Computing
Teddington, England

The conversion of a small routine from FORTRAN (or PASCAL) to Ada programming language is discussed. It is shown that there is a substantial difference between a naive transliteration of a subroutine and an ideal Ada package, because of the increased capabilities of Ada which allow for greater functionality. The increased functionality that is not available in the FORTRAN version includes: automatic selection of two variants according to the machine characteristics; improved security of the seed to incorrect access; automatic default initialization of the seed by use of an internal clock (in a machine-independent fashion); and ability to provide multiple sequences giving security in a tasking environment.

JOVIAL (J73) TO Ada Translator

M. J. Neiman
Proprietary Software Systems, Incorporated
Beverly Hills, CA

This document contains the functional description and system/subsystem specifications for a JOVIAL J73/Ada translator, and guidelines for J73 programmers who anticipate their programs will be converted to Ada at a later date. The functional description specifies the maximum JOVIAL J73 subset that can be converted to Ada. Techniques for the optimum automatic translation of the source code are specified. The J73 constructs that cannot be automatically translated are identified. The system/subsystem specification provides a more detailed breakdown of the proposed translator.

APPENDIX E

CMS-2 PROGRAMMING LANGUAGE SUPPORT PRODUCTS

This appendix lists the automated support that is available for the CMS-2 programming language. The point of contact for each item and a brief description of the support capability is included.

CMS-2M PARSER AND IV & V TOOLKIT
ARC Professional Services Group
Defense Systems Division
Attn: Mr. John P. Leckey, III
601 Caroline Street
Fredericksburg, VA 22401
(703) 371-2100

The CMS-2M source code is manually merged with the input file(s) that contain global data, in order to create a complete CMS-2M source program. This program is then input to a CMS-2M parser and file/report generator. Its outputs include four reports (GOTOs, Jumps, Interrupts and Data set/used reports) and four files (Segment, Calls, Code and Data set/use files). The reports are not used for further processing while the files are input to the IV & V TOOLKIT.

The IV & V integrated tool set (TOOLKIT) is a collection of functions called modules specifically designed for verification and validation analysis. The IV & V TOOLKIT will create code paths and analyze data and control coupling. The TOOLKIT incorporates the following menu/modules of FLOW Analysis: Design Verification, Flow Analysis, Test Correlation, Report Preparation and Utilities. (* New product COPE replaced IV & V TOOLKIT as of 15 January 1991)

CMSTOOL
Naval Surface Warfare Center
Code E32
Attn: Michael Peeler
Dahlgren, VA 22448
(703) 663-8836

CMSTOOL is a CMS-2 source code metrics tool (quality measurement). This tool provides a McCabe-based complexity measurement, subtasking level,

commenting level, direct code usage factor, sizing factors, and structured coding standards. The tool is VAX Pascal-based and includes source program, executable code, DCL command files to support batch and interactive run modes, installation/user notes, and a program description document formatted for laser print.

CMSTOOL is a handy tool to analyze CMS-2 source code procedures and functions. CMSTOOL output which was extracted from the input file CMS-2, allows us to evaluate computer program modularity, understandability, complexity, portability and maintainability. Tool output can also be utilized in the test planning strategy.

OLTOOLS
Naval Surface Warfare Center
Code N23
Attn: Mr. Robert Bartholow
Dahlgren VA 22448
(703) [illegible]-1182

On-line tools program (OLTOOLS) written in ANSI standard C is a collection of analyzer utilities that can be used to generate reports using one or more CMS-2 source files as input. The source files analyzed by the OLTOOLS program are assumed to be syntactically correct and compilable under the MTASS-I CMS-2 compiler. The tools parse both CMS-2 syntax (MTASS I, L, M) and Direct code (UYK-7, 20, 43, 44). The output is static design reports. The following is a description and output of different utilities.

AEGIS has developed many tools to support either design or tactical field, including ADDS, Generic ADDS, MicroADDS. ADDS is based on the Entity-Relationship-Attribute (ERA) Model. Generic ADDS is a CMS-2M Reverse Engineering tool that includes both syntax and lexical analyzers in the same program. Generic ADDS reads the CMS-2 code and converts it directly to PSL/PSA (a tool of the database management system which is based on the ERA Model to produce reports). PSL/PSA was developed by the META Corporation in Michigan. MicroADDS is a PC-based system that offers the capability to perform string searches through large volumes of AEGIS text. With its PC-base, it is portable and fast.

The Naval Air System Command Software Engineering Environment (NASEE) Toolset includes several tools which are related to reengineering including CMS-2 and Ada Cohabitational Tool, LOGISCOPE Tool and COMPOSER Tool. Mr. Coyle and Mr. Corsin of Naval Air System Command (NAVAIR) are in charge of the NASEE tools.

CMS-2 AND ADA COHABITATION TOOL
Unisys Corporation
Attn: Ms. Kari Kruempel
P.O. Box 64525, MS WS17
St. Paul, MN 55164 - 0525
(612) 456-7352

The Unisys Cohabitation Tool provides the capability of implementing computer system software in both Ada and CMS-2 programming languages. Unisys could not fully support the conversion of existing CMS-2 systems to Ada-based implementations while meeting certain real-time requirements. One solution to this problem was keeping the original CMS-2 code that met these requirements, and creating a bridge to interface with Ada software to implement other functional requirements. Phase I of the Cohabitation demonstration will be available on video-tape in Summer 91.

CMS-2 IMPORTATION TOOL
Control Data Corporation (BLCS1M)
Attn: Mr. Jerry Hess
P.O. Box 1305
Bloomington, MN 55440
(612) 921-7214

The Importation Tool imports CMS-2/Assembly routines to make them accessible to Ada applications by using the Foreign Code Importer of ALS/N to modify the relocatable object module produced by a CMS-2 processor so that it can be used as input to the LNKVAX linker (i.e., converting an IMTASS Target System File into an ALS/N Phase Linked Container).

Syscon Corporation
San Diego, CA 92110

Syscon has several CMS-2 and other reengineering products, and provides various services in the area of reengineering.

VITRO Corporation
Attn: Morris J. Zwick
(301) 231-2784
14000 Georgia Avenue
Silver Spring, MD 20906-2972

VASTT is the VITRO Automated Structured Testing Tool. Vitro has several CMS-2 testing and verification products.

DISTRIBUTION

	Copies
Defense Technical Information Center Cameron Station Alexandria, VA 22314	12
Library of Congress Attn: Gift & Exchange Div Washington DC 20540	4
Office of Naval Technology Attn: Code 227 (Elizabeth Wald) 800 N. Quincy Street Arlington, VA 22217-5000	1
Advanced Technology & Research Corp Attn: Adrien J. Meskin 14900 Sweitzer Lane Laurel, MD 20707	2
Computer Command and Control Company Attn: Evan Lock 2300 Chestnut Street, Suite 230 Philadelphia, PA 19103	1
Center for Naval Analyses 4401 Fort Avenue P.O. Box 16268 Alexandria, VA 22302-0268	2
Software Productivity Consortium Attn: Dr. Robert S. Arnold 2214 Rock Hill Road Herndon, VA 22070	1
GE Corp R&D Center Attn: Jeanette Bruno P.O. Box 8 River Road K1 2C1 Schenectady, NY 12301	1

Internal Distribution	Copies
D4 (H. Crisp)	1
E	2
E30	1
E32	1
E231	2
E232	3
E342 (GIDEP)	1
F	1
F01	1
F30 (J. Sweigart)	1
F31	1
F31 (W. Laposata)	1
F31 (D. Yeagle)	1
G	1
G07 (F. Moore)	1
G11 (W. Lucas)	1
G42 (T. Dumoulin)	1
G42 (A. Farsaie)	1
H	1
H023 (G. Moore)	1
H32 (J. Holmes)	1
K	1
K01 (J. Giaquinto)	1
K10	1
K51	1
K505 (J. Henderson)	1
K54 (W. Farr)	1
N	1
N23	1
N35	1
N35 (S. Masters)	1
N35 (M. Kuchinski)	1
R	1
R02 (B. DeSavage)	1

DISTRIBUTION (Cont.)

	Copies
U	1
U02	1
U04 (M. Stripling)	1
U042	1
U10	1
U20	1
U30	1
U33	1
U302 (P. Hwang)	20
U33 (D. Choi)	1
U33 (R. Duran)	1
U33 (M. Edwards)	1
U33 (K. Gibson)	1
U33 (N. Hoang)	1
U33 (S. Howell)	1
U33 (M. Jenkins)	1
U33 (T. Moore)	10
U33 (C. Nguyen)	1
U33 (T. Park)	1
U33 (H. Roth)	1
U33 (M. Trinh)	1
U33 (P. Wallenberger)	1
U40	1
U042 (J. Bilmanis)	1

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1990	

4. TITLE AND SUBTITLE
The Reengineering of Navy Computer Systems

5. FUNDING NUMBERS

6. AUTHOR(S)
Tamra Moore and Katherine Gibson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Surface Warfare Center (Code U33)
10901 New Hampshire Avenue
Silver Spring, MD 20903-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
NAVSWC TR 90-216

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report defines the different processes of reengineering, establishes guidelines for reverse engineering, and suggests automated products for supporting the reengineering processes. A thorough examination of all of the available reengineering products was not within the scope of this research, but has been designated for future examination at the Naval Surface Warfare Center. Information gathered on the products available during this project is presented in the appendices of this report so that the reader can contact the affiliated agencies directly for specific technical information.

14. SUBJECT TERMS
Maintenance
Systems Engineering
Reverse Engineering
Reengineering
Software Engineering
Methodology
Design Capture

15. NUMBER OF PAGES
89

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and its title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

BLOCK 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If Known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief ***(Maximum 200 words)*** factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code ***(NTIS only)***

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.