

AD-A274 699**DOCUMENTATION PAGE**Form Approved
OMB No. 0704-0188

On average, it takes one hour per response, including the time for reviewing instructions, searching existing data sources, gathering and reviewing the information, and completing and reviewing this burden estimate. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE 13 May 93		3. REPORT TYPE AND DATES COVERED Final	
4. TITLE AND SUBTITLE Information Systems Criteria for Applying Software Reengineering: Guidelines for Identifying Information Systems for Software Reengineering				5. FUNDING NUMBERS	
6. AUTHOR(S) Tamra K. Moore					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Defense Information Systems Agency Center for Information Management Software Systems Engineering Directorate 701 South Courthouse Rd Arlington, VA 22204-2199				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING MONITORING AGENCY NAME(S) AND ADDRESS(ES) Director, Defense Information				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) Knowing when to reengineer is key to helping managers make decisions that are cost effective and beneficial to reaching their organizational goals. The Software Reengineering Criteria provides assistance in identifying automated information systems (AIS) that are candidates for reengineering and proposes how software reengineering technology can benefit AIS. Experiences in industry, government agencies, and academia fostered the development of this Criteria for identifying candidate information systems for software reengineering. This paper presents the Software Reengineering Criteria in four parts. First, a set of terms defines the activities within reengineering technology (Section 2: Definitions). An overview summarizes other sources providing guidance for when to reengineer (Section 3: Existing Guidance for Software Reengineering). The criteria characterizes software systems and software engineering environments that exhibit the potential to benefit from reengineering technology and presents recommendations for software reengineering strategies (Section 4: Software Reengineering Criteria). Finally, potential application areas for the Criteria are explored (Section 5: Application Areas for Selection Criteria).					
14. SUBJECT TERMS DOCUMENTS TO BE ADDED TO THE CIM ACCOUNT Software Reengineering, reverse engineering, maintenance, Software Process Improvement				15. NUMBER OF PAGES 103	
				16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT		

Defense Information Systems Agency
Joint Interoperability Engineering Organization
Center for Information Management
701 South Courthouse Road; Arlington, VA 22204-2199

DTIC QUALITY INSPECTED 8

Accession For	
NTIS	CRA&I ☒
DTIC	TAB ☒
Unannounced	☐
Justification _____	
By _____	
Distribution /	
Availability Codes	
Dist	Avail and / or Special
A-1	

*Center for Information Management
Information Systems Criteria for Applying
Software Reengineering*

May 1993

CLEARED
FOR OPEN PUBLICATION

OCT 7 - 1993

CLASSIFIED BY: [REDACTED]
AUTHORITY: [REDACTED]
DATE: [REDACTED]

Prepared by:

Software Systems Engineering Directorate
Reengineering Division

94-01408



94-1 12 002

93-1-3474

**Defense Information Systems Agency
Joint Interoperability Engineering Organization
Center for Information Management
701 South Courthouse Road; Arlington, VA 22204-2199**

***Center for Information Management
Information Systems Criteria for Applying
Software Reengineering***

May 1993

Prepared by:

**Software Systems Engineering Directorate
Reengineering Division**

FOREWORD

This document was prepared by the Defense Information Systems Agency, Joint Interoperability Engineering Organization (DISA/JIEO), Center for Information Management, sponsored by the Office of the Director of Defense Information. The Software Systems Engineering Directorate, Reengineering Division welcomes any comments concerning the contents of this document.

TABLE OF CONTENTS

SECTION	PAGE
Foreword	i
Table of Contents	ii
List of Figures	iv
Executive Summary	v
1. INTRODUCTION	1
1.1 Scope	1
1.2 Motivation	2
2. DEFINITIONS	3
2.1 Definitions of Terms	3
2.1.1 Software Reengineering	3
2.1.2 Reverse Engineering	3
2.1.3 Restructuring	4
2.1.4 Forward Engineering	4
2.1.5 Redocumentation	4
2.1.6 Translation	4
2.1.7 Software Reuse	4
2.2 Interfaces Between Reengineering Activities	5
2.2.1 Reverse engineering and Redocumentation	5
2.2.2 Restructuring and Reengineering	6
2.3 Reengineering Process	6
3. EXISTING GUIDANCE FOR REENGINEERING	9
3.1 STSC's Reengineering Candidate Selection Process	9
3.2 Joint Logistic Commanders Santa Barbara I Workshop	9
3.3 NIST Variables for Reengineering Effectiveness	10
3.4 Other Guidance Sources	10
3.5 Conclusions	11
4. SOFTWARE REENGINEERING CRITERIA	12
4.1 Existing Software System Criteria	13
4.1.1 Product Characteristics	13
4.1.1.1 Software Characteristics	13

4.1.1.2 System Characteristics	17
4.1.1.3 Environment Characteristics	18
4.1.2 Process Factors	20
4.1.2.1 Development Factors	20
4.1.2.2 Maintenance Factors	22
4.2 Reengineered Software System Criteria	24
4.2.1 Product Characteristics	24
4.2.1.1 Target Software Characteristics	24
4.2.1.2 New Support Characteristics	26
4.2.1.3 New Environment Characteristics	28
4.2.2 Process Factors	29
4.2.2.1 Organization Factors	29
4.2.2.2 Methodology Factors	31
4.2.2.3 Resource Factors	32
5. APPLICATION AREAS FOR SELECTION CRITERIA	35
5.1 Homogeneous Application	35
5.2 Heterogeneous Application	35
5.3 Results of Effort to Define Selection Criteria Method	36
6. CONCLUSION	37
List of References	38

APPENDICES

Appendix A - GLOSSARY OF TERMS	A-1
Appendix B - EXISTING SOFTWARE REENGINEERING GUIDANCE SOURCES ..	B-1
Appendix C - SOFTWARE REENGINEERING CASE STUDY DATA	C-1
Appendix D - SOFTWARE REENGINEERING CRITERIA TABLES	D-1
Appendix E - SOFTWARE REENGINEERING CRITERIA QUESTIONNAIRE	E-1
Appendix F - EXAMPLE SOFTWARE REENGINEERING CRITERIA APPLICATION	F-1

LIST OF FIGURES

<u>FIGURES</u>	<u>PAGE</u>
2-1. Comparison of Software Reengineering Activities.	5
2-2. Software Reengineering Activities Relationships.	7
2-3. Software Reengineering Activities Data Flow.	8
4-1. Software Reengineering Criteria Categories.	12

EXECUTIVE SUMMARY

Knowing when to reengineer is key to helping managers make decisions that are cost-effective and beneficial to reaching their organizational goals. The Software Reengineering Criteria provides assistance in identifying automated information systems (AIS) that are candidates for reengineering and proposes how software reengineering technology can benefit AIS. Experiences in industry, government agencies, and academia fostered the development of this Criteria for applying software reengineering technology to information systems by providing guidelines for identifying potential candidates for reengineering. The Criteria provides support for:

- (1) predicting the return on investments made by reengineering existing software systems, and
- (2) providing the business case for achieving the highest rate of return from software reengineering technology.

Reengineering emerges as a strategy for bringing the cost of developing and maintaining software under control. The need for a comprehensive plan to achieve goals, set objectives, develop strategies, and apply reengineering technology is the driving force in many new modernization efforts throughout the Department of Defense. Determining which software system can benefit from reengineering technology and how it will benefit is the first step in these efforts. The Software Reengineering Criteria will assist managers facing this situation.

Ultimately, the process of software reengineering must support the high-level goals of an organization, which include

- (1) elimination of non-essential products and processes,
- (2) increasing the value of those remaining; and
- (3) increasing the efficiency of those processes through streamlining, simplification and/or automation [Room92].

Two broad concepts guide software reengineering technology development in the DoD. The first is the prevention of unnecessary duplication by joint use of personnel, information systems, facilities, and services across DoD. The second concept is conformance to new regulations, policies, standards, and guidelines for software acquisition and support. Current practice within the Federal government is the application of reengineering to perform Pre-planned Product Improvement (P³I) efforts or modernize software systems to meet new

standards. These standards include using the Ada programming language, moving towards open software systems, and maintaining compliance with the Portable Operating System Interface Exchange (POSIX). Guidelines include integrating Commercial Off-the-Shelf (COTS) whenever possible, including Computer-Aided Software Engineering (CASE) into existing software engineering environments.

This paper summarizes the results of work defining a criteria for identifying automated information systems that may benefit from software reengineering. This criteria defines the information that currently serves as guidelines for making this determination. Future work will apply these guidelines to establish each as a formal criteria for identifying information systems as candidates for software reengineering.

This summary presents the Software Reengineering Criteria in four parts. First, a set of terms defines the activities within reengineering technology (Section 2: Definitions). An overview summarizes other sources providing guidance for when to reengineer (Section 3: Existing Guidance for Software Reengineering). The Criteria characterizes software systems and software engineering environments that exhibit the potential to benefit from reengineering technology and presents recommendations for software reengineering strategies (Section 4: Software Reengineering Criteria). Finally, potential application areas for the Criteria are explored (Section 5: Application Areas for Selection Criteria).

Until recently, various terms were used interchangeably to describe the activities in software reengineering. Industry, government, and academia are achieving a consensus on terminology for the high-level activities within software reengineering and this document presents a set of definitions that are representative of this consensus.

Several sources define guidance for determining when to reengineer, including Federal agencies and others working in the commercial arena. This guidance lists important high-level issues to consider prior to software reengineering, but does not discuss how the specific characteristics of a software system influence the decision to reengineer.

The Software Reengineering Criteria defines the type of information that is used to identify information systems as potential candidates for software reengineering and determines how reengineering technology can benefit computer software systems. The Criteria is composed of two categories: those that deal with the existing software system(s) and those that are desired in the reengineered software system. Existing Software System Criteria characterize existing software system(s), the environment in which it operates, and the process by which it was developed and is currently maintained. Reengineered Software System Criteria characterize the target software, the new support environment, the new operation environment, and the factors which influence the software reengineering process.

The Software Reengineering Criteria was developed using data gathered from completed reengineering projects. To date there is little documented experience estimating the effort involved in software reengineering. As experience builds, so will the understanding

of how software reengineering should be performed. This document serves as the basis for such guidance and provides information that will benefit organizations considering software reengineering technology.

(This page intentionally left blank.)

1. INTRODUCTION

The Software Reengineering Criteria provides assistance in identifying automated information systems (AISs) that are candidates for reengineering and proposes how reengineering technology can benefit these information systems. Knowing when to reengineer is key to helping managers make decisions that are cost-effective and beneficial to reaching their organizational goals. Experiences in industry, government agencies, and academia were used to develop this Criteria for selecting software systems to reengineer. The Criteria should be supplemented by cost/benefit analysis for determining whether reengineering is an appropriate means for modernizing software systems in a given environment. The Criteria assists in predicting the return on investments made by reengineering existing software systems, and supports the development of the business case for achieving the highest rate of return from reengineering technology.

1.1 Scope

The Software Reengineering Criteria addresses information systems, primarily those within the Department of Defense (DoD) and its jurisdictions. These software systems are data-intensive, often interacting with a database, perform many processes in batch mode, and are concerned with report generation. The term "software" will be used throughout this document to refer to the source code programs implemented to meet requirements of a given information system. The "software system" will refer to this software, excluding any Commercial Off-the-Shelf (COTS) products with which this software integrates to fulfill the overall requirements of the information system. The software system does not include the hardware platform on which these components execute; the hardware suite is considered part of the operational environment.

This paper summarizes the results of work defining a criteria for identifying automated information systems that benefit from software reengineering. Data gathered from completed reengineering projects [Hobb91, MITR92, Ruhl91] supported the development of the Criteria which recommends a reengineering strategy utilizing reverse engineering, restructuring, redocumentation, translation, and software reuse. These initial results succeed in identifying potential criteria that currently serve as guidelines for making this determination, future work will apply these guidelines for verifying each as a measurable criteria for applying software reengineering.

This paper presents the Software Reengineering Criteria in four parts. First, a set of terms for defining software reengineering activities is presented (Section 2: Definitions). An overview of other sources that have provided guidance for when to reengineer is summarized (Section 3: Existing Guidance for Reengineering). The Criteria is then presented along with

guidance for how data collected using the Criteria is used to determine how and when to reengineer (Section 4: Software Reengineering Criteria). This guidance is based on key practices within software engineering applicable to most organizations and therefore is not dependent on an organization's maturity. Finally, potential application areas for the Criteria are explored (Section 5: Application Areas for Selection Criteria).

1.2 Motivation

The Defense Information Systems Agency, Center for Information Management (DISA/CIM) has the mission of providing information management technical services to the DoD community. The DISA/CIM Software Systems Engineering Directorate is responsible for assessing and promoting current reengineering technology in DoD modernization efforts. The Software Reengineering Assistance Program has been established by the directorate to meet this objective. This Program uses experiences in industry, government agencies, and academia to develop the Criteria for selecting software systems to reengineer.

The DoD Information Management (IM) community has approximately 1.4 billion lines of operational software today. Many of these systems were developed prior to the availability of modern technologies, including methods, languages, and automation. The CIM initiative for reducing the number of software systems across functional domains and eliminating redundancy, has generated a strong need to modernize these systems. This modernization is supported by software reengineering strategies that minimize development costs, reduce maintenance expenses, and leverage existing software assets. To this end, the Criteria will support the choice of information systems that will benefit from reengineering.

Reengineering is used in Pre-planned Product Improvement (P³I) efforts or modernize software systems to meet new standards. These standards include MIL-STD-1815A, Ada Programming Language; FIPS 146-2, Government Open-Systems Interconnection Protocol; and FIPS 151-1, Portable Operating System Interface Exchange (POSIX). Reengineering is also used to improve the maintenance of existing software systems, by integrating Computer-aided Software Engineering (CASE) into the existing software engineering environments. The Software Reengineering Criteria will assist those managers faced with these situations.

2. DEFINITIONS

A set of definitions¹ for terms used to describe the activities within software reengineering is presented in the following section (Appendix A GLOSSARY OF TERMS). Until recently, the capabilities of reengineering were so tightly intertwined that these terms were often used interchangeably. However, there are unique qualities in these activities which distinguish them from one another, and these qualities form the distinction for each definition. Industry, government, and academia have begun to reach a consensus on the major terms in software reengineering and this document presents a set of definitions that are representative of this consensus.

Software reengineering activities are distinct in two ways: (1) the objects acted upon during the activity and (2) the products that are produced as a result of doing this activity (Figure 2-1). In addition, the terms have specific primary objectives that are the goal of each activity and there are other issues that distinguish each activity from the others.

2.1 Definitions of Terms

2.1.1 Software Reengineering. The examination and alteration of an information system to reconstitute it in a new form. The process encompasses a combination of other processes such as reverse engineering, restructuring, forward engineering, redocumentation, and translation. The goal is to improve the software system (functionality, performance, or implementation). The distinction is that additional functionality is often incorporated into the system during this process [Kerr91, Perr92].

2.1.2 Reverse Engineering. The process of examining an information system by analyzing its documentation, application software, and data structures within the environment in which the information system operates. This analysis is performed to (1) identify the system's components and their interrelationships, and (2) create representations of the system in another form or at a higher level of abstraction. The goal is to understand the existing software system (functions, performance, or implementation). Extracted information is represented in a format which can be integrated into the life cycle for development of a software system [Kerr91, Perr92].

¹ The definitions for reengineering, redocumentation, and restructuring are variations of the definitions originally presented by Chikofsky and Cross [Chik90].

2.1.3 Restructuring. The transformation of a software system from one representation form to another at the same relative abstraction level, while preserving the system's external behavior (functionality and semantics). The goal is to improve a representation of the existing software system. The distinction is that restructured systems are functionally equivalent to the existing system, but should be easier to support [Kerr91].

2.1.4 Forward Engineering. Within the context of reengineering, forward engineering is the software engineering activities that consume the products of reengineering activities primarily reverse engineering, reuse, and new requirements to produce a target system. The goal is to create a software system via reengineering. This term primarily refers to the process of generating new software systems from reverse engineered designs. This term has evolved within reengineering to refer to those software engineering activities (traditionally performed during development) that are performed during or as a result of reengineering.

2.1.5 Redocumentation. The creation or revision of a semantically equivalent representation with the same relative abstraction level. The goal is to understand a given representation of the existing software system (whether it be a specification, design, or implementation). The distinction is that this activity does not alter the existing software system representation, nor does it generate any new representation to replace any part of the existing representation. Redocumentation produces supplementary information that provides understanding of the existing system and its sub-parts. This activity is usually performed to assist in maintenance of existing system.

2.1.6 Translation. Transformation of source code from one language to another or from one version of a language to another version of the same language. The goal is to improve the linguistic implementation of the software. This process is most successful when the two languages are similar or have a defined mapping between syntax.

2.1.7 Software Reuse. The application of existing software work products, including source code, documentation, designs, test data, tools, and specifications, in a software development effort other than the one for which each was originally developed. The goal is to facilitate the return on investment (ROI); improve software quality and reliability; shorten system development and maintenance times; increase productivity and minimize software-related risks. Software reuse should be employed during reengineering and reengineering should be applied to identify candidate reusable assets.

Term	object	product	Goal
Forward Engineering	reverse engineered design	software	generate software system from reverse engineered products
Reengineering	existing software	new software	improve software system
Reverse Engineering	existing software	reverse engineered design	understand software system; prepare for development of replacement system
Restructuring	existing software	improved existing software representation	improve a given representation of software system
Redocumentation	uses software, existing documentation, and available expertise	supplemental documentation	understand software system; maintain existing system
Software Reuse	software assets	software component (new context)	ROI, eliminate redundancy
Translation	existing source code or design language	new source or design code in alternate language	improve source code or design code

FIGURE 2-1. Comparison of Software Reengineering Activities.

2.2 Interfaces Between Reengineering Activities

The definitions above suggest strong relationships between terms which need clarification. These relationships are (1) reverse engineering and redocumentation and (2) restructuring and reengineering.

2.2.1 Reverse engineering and Redocumentation. Software reverse engineering and redocumentation have the goal to provide understanding of the existing software system. The differences between these capabilities are the end product and the underlying motivation for this activity. Reverse engineering generates a product that could potentially serve as a design or specification for the generation of new software. Redocumenting software produces supplementary reference material that explains a given representation of the software system, whether it be design or implementation. Reverse engineering most often results in a more comprehensive view of the software. Documentation usually explains a specific aspect of the software system in further detail.

2.2.2 Restructuring and Reengineering. Restructuring and reengineering both improve the software system, but restructuring is limited to the modification of a current representation of the system code which will replace the existing representation, without introducing new functionality, or implementation characteristics (i.e., programming language). Reengineering often includes functional alterations and often succeeds in generating a completely new implementation of the software in a new programming language.

2.3 Reengineering Process

The reengineering process often consists of several activities when it is applied to a software system (Figure 2-2). A data flow diagram is a useful format for representing the relationship between reengineering activities (Figure 2-3) relative to the data shared by each activity.

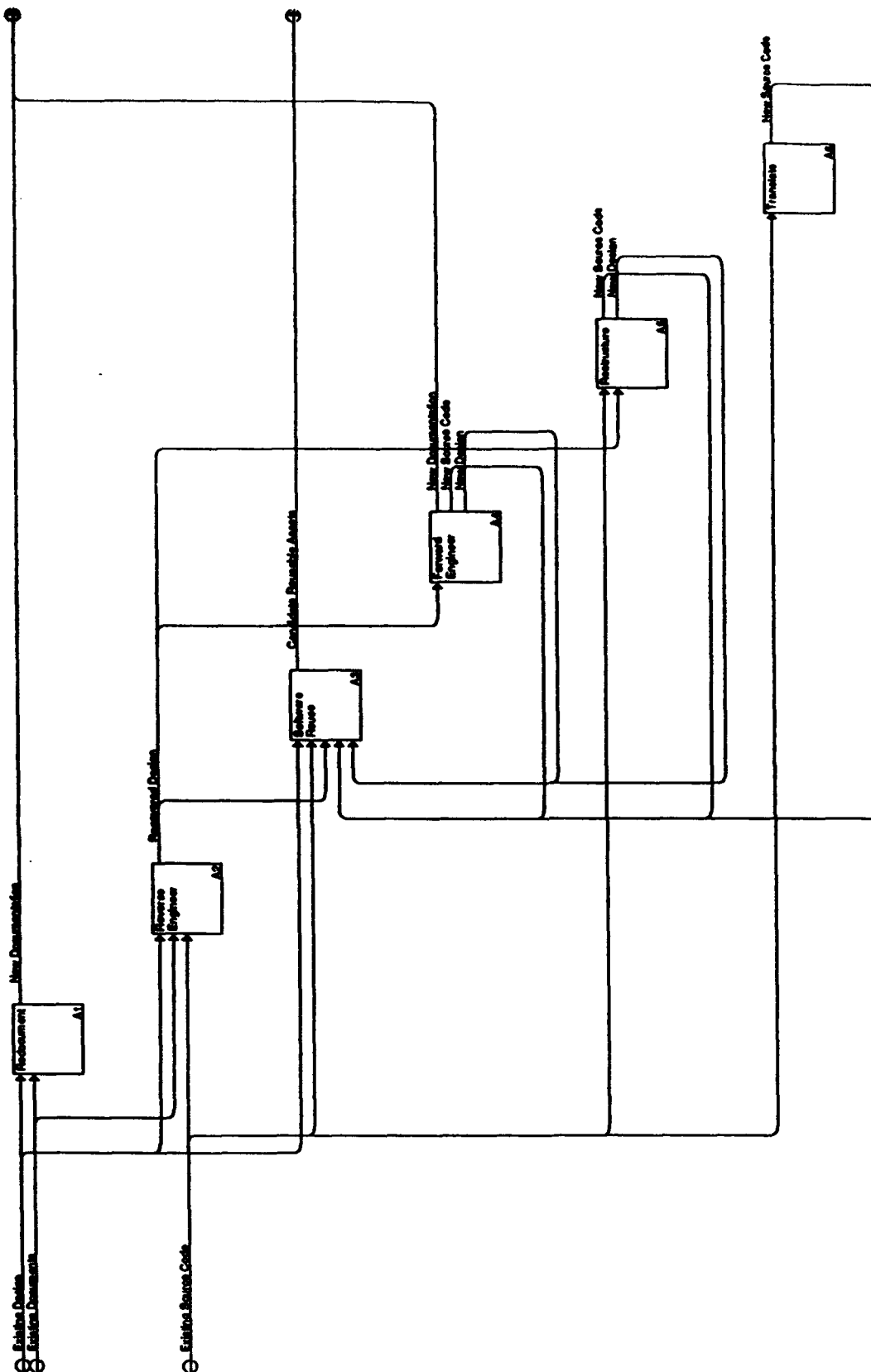


FIGURE 2-2. Software Reengineering Activities Relationship.

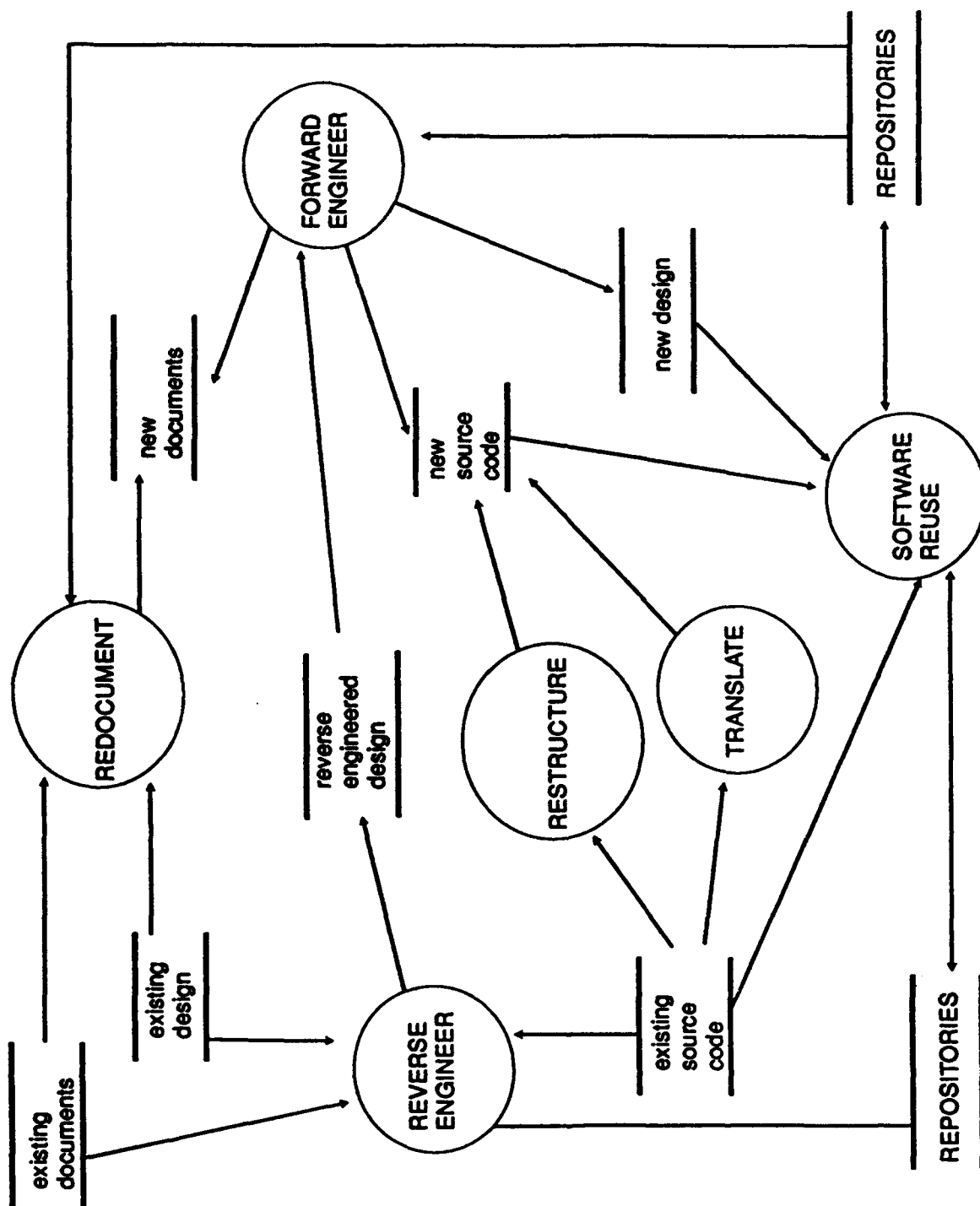


FIGURE 2-3. Software Reengineering Activities Data flow.

3. EXISTING GUIDANCE FOR REENGINEERING

Existing guidance for determining when to reengineer has been defined by the Software Technology Support Center (STSC) [Sitt92b], the Joint Logistic Commanders (JLC) Santa Barbara I Workshop [JLC92], the National Institute of Standards and Technology (NIST) [Ruhl91], and by others working in the commercial arena [Kerr91, Perr92]. This guidance lists important high-level issues to consider prior to reengineering and provides assistance in determining what type of reengineering should be performed on a candidate software system. It has also been suggested that there is a close link between software reuse and reengineering [Stev92]. This guidance does not discuss how the specific characteristics of a software system influence the decision to reengineer nor does it discuss the trade-offs for differing reengineering strategies. More detailed summaries of this guidance are discussed in Appendix B EXISTING REENGINEERING GUIDANCE SOURCES. A brief summary of each source is presented below.

3.1 STSC's Reengineering Candidate Selection Process

The STSC's Reengineering Candidate Selection Process is a common-sense approach to determining the most appropriate reengineering methodology to perform (e.g., restructure, reverse engineer, redocument). The process is based on a set of weighted questions that are dependent on three variables describing the software: complexity, importance, and longevity [Sitt92b, p24].

3.2 Joint Logistic Commanders Santa Barbara I Workshop

The JLC Joint Policy Coordinating Group on Computer Resources Management (CRM) sponsored the First Software Reengineering Workshop: Santa Barbara I on September 21-25, 1992 in Santa Barbara, California. Santa Barbara I brought together members of Government, Industry, and Academia to define reengineering technology and determine the impact of reengineering on software acquisition policies. The Proceedings of Santa Barbara I contains a list of the most critical criteria for determining when to reengineer (Appendix B) [JLC92]. Examples of this criteria include Technical Quality, Life-Cycle Remaining, and Maturity Level. This criteria includes important issues to consider prior to reengineering, but does not provide direction as to what reengineering strategy to follow.

A draft Reengineering Economics Handbook (MIL-HDBK-REH) was developed during Santa Barbara I [REH92]. This handbook defines a Reengineering Decision-Making Process that is based on the STSC's Reengineering Candidate Selection Process. The MIL-

HDBK-REH is in draft form and will evolve as part of the on-going Santa Barbara I effort. The objective of the handbook is to assist in deciding whether a software system should (1) be maintained at a similar level of effort, (2) reengineered to change, modify, or improve the software, or (3) replace the current software with a completely new system.

3.3 NIST Variables for Reengineering Effectiveness

A NIST study performed in 1991 concluded that the effectiveness of reengineering is dependent on three variables [Ruhl91, p14]: (1) corporate and system goals; (2) condition of the current application and documentation; and (3) available resources (tools and personnel). A few of the recommendations which resulted from this study include the following [Ruhl91, p16-23]:

When procuring equipment, require conformance to applicable standards (e.g., FIPS) to achieve flexibility and ease in future migrations.

While design recovery is difficult, time-consuming, and essentially a manual process, it is vital for recovering lost information and information transfer.

Provisions in terms of personnel and effort must be made to compensate for the lack of full support of the reengineering process by currently available off-the-shelf tools.

Adequate storage capacity and processor speed in equipment supporting the reengineering tools are essential to facilitate the reengineering process.

It is critical that the application system experts be involved throughout the reengineering process. They are essential for design recovery.

3.4 Other Guidance Sources

Other recommendations for determining when a software system is applicable for reengineering have been defined by industry representatives in recent literary articles [Kerr91, Perr92]. These recommendations provide general guidance that is useful when deciding when to reengineer, but are highly intuitive and do not provide any indications as to how the status of a candidate application impacts the success of the reengineering process.

Kerr and McGovern consider reengineering as part of the maintenance process [Kerr91]. Candidate applications are categorized into three types of systems based on levels of importance (this is similar to the STSC Process). High importance applications should be reverse engineered to improve quality and ease maintenance. Applications of medium

importance should be reengineered, during which other applications of similar importance are combined to justify resources necessary to perform this activity and consolidate systems, thus minimizing resources for more than one system. Applications that are not important should be left alone, assuming that the resources necessary for reengineering these systems are not worth the benefits.

Perry defines reengineering activities within the scope of downsizing from mainframe to microcomputer platforms. Since most object-oriented modeling, CASE, and reengineering tool support is available on workstations, downsizing is often intrinsic to the reengineering process selected. Downsizing begins with a three-step process, starting with the selection of the appropriate application, component analysis for partial-downsizing, and finally the determination of a technique for downsizing. These techniques include using a commercial packaged environment that enables the microcomputer to perform like a mainframe, in which case the application should be able to execute in its existing form. The other three techniques include restructuring, reengineering, and reverse engineering.

3.5 Conclusions

The STSC process provides useful guidelines for determining a reengineering option for a candidate software system. The NIST goal variables must also be considered in order to predict the success of the reengineering. The Software Reengineering Criteria considers the characteristics of the existing software system and the desired target system, the available reengineering technology, as well as the impact of such issues as those defined by the NIST goal variables.

4. SOFTWARE REENGINEERING CRITERIA

The Software Reengineering Criteria provides insight into the benefits of reengineering technology. The Criteria is composed of two categories: criteria that describe the existing software system(s) and criteria defined by reengineering technology (Figure 4-1). Existing Software System Criteria characterize the existing software system, the environment in which it operates, and the process by which it was developed and is currently maintained. Reengineered Software System Criteria characterize target software systems, new support environments, new operational environments, and the factors which influence the reengineering process.

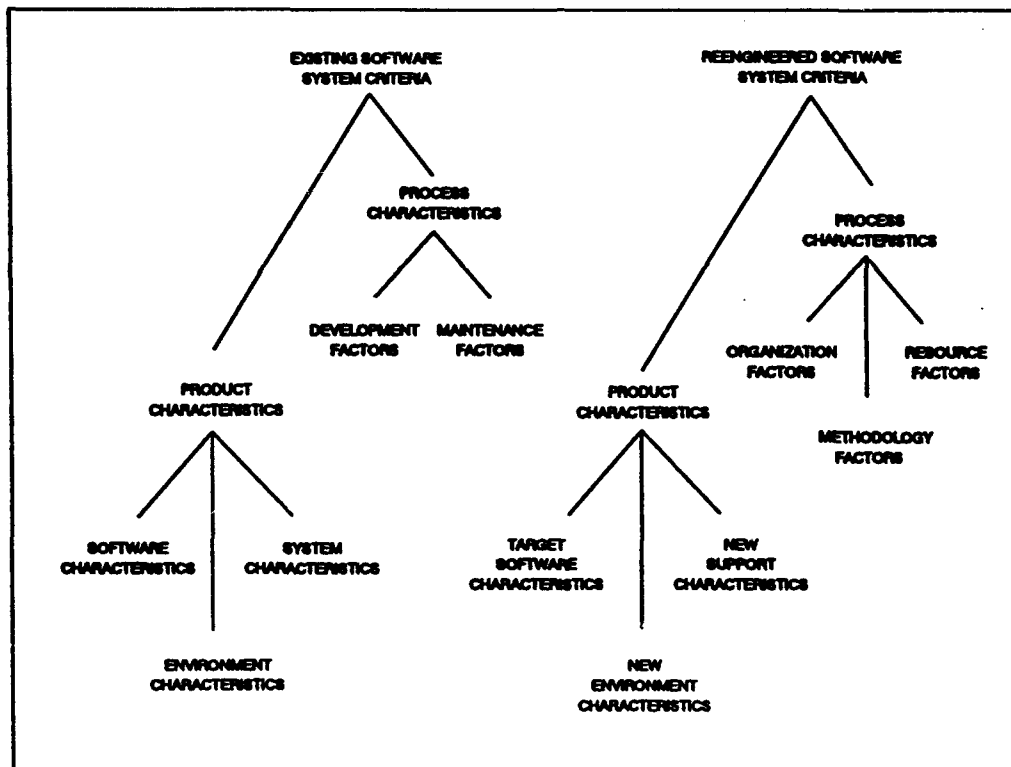


FIGURE 4-1. Software Reengineering Criteria Categories.

The Software Reengineering Criteria can be used in several ways. The Criteria can be applied to either a specific software system or an entire software engineering environment. An individual criteria can also be examined for its impact on applying reengineering

technology. For example, the impact of available design information when reengineering can be examined by looking up the criteria called Design.

The Criteria should be measured numerically where possible, otherwise documented in concise, easily understood terms. It is important to be consistent when applying the Criteria across multiple software systems and to be aware of the differences in units of measurements when comparing external data.

Each of the criteria is defined in terms of product and process². A tabular listing of the Criteria is located in Appendix D SOFTWARE REENGINEERING CRITERIA TABLES. The impact of each criteria on reengineering is discussed, including a description of the reengineering strategy that is recommended given that certain criteria is met.

4.1 Existing Software System Criteria

Existing Software System Criteria is composed of product characteristics that describe the existing software system, the environment in which it operates, and process factors which influenced the development and maintenance of the system.

4.1.1 Product Characteristics. Product characteristics describe the software system, including software characteristics and system characteristics, and the characteristics describing the environment in which this system operates.

4.1.1.1 Software Characteristics. Software characteristics describe the source code of the system. Several authors have defined software characteristics which impact reengineering (maintenance) [Boeh81, NIST, Sitt92a, Sitt92b]. The Software Characteristics Criteria are defined below.

<u>Software Characteristics</u>	<u>Description</u>
Complexity	software constructs (conditionals, iterations, statements), calls to external routines, and I/O
Data structure	variables, constants, complex structures (records, linked lists), user-defined constructs
Languages	number of implementation languages and defined grammars for each

² Product criteria are referred to as characteristics, since this criteria usually describes an attribute of the software or its environment. Process criteria are called factors since this criteria influence the activities of developing, maintaining, and reengineering.

Modularity	defined, unique functionality in each module
Size	SLOC, function points, number of modules/objects, bytes, number of files
Software change rate	average number of modifications, both corrective and perfective, over a given period of time
Software importance	Number of times this software is accessed in a given period of time; is the software functionality life-threatening; does the software perform some function(s) not performed anywhere else in the organization?
Structural quality	an analysis of the structure of the software system, i.e, missing code, code that is never executed; evidence of work-arounds.

Complexity describes the software constructs (conditionals, iterations, statements), calls to external routines, and I/O. There are several methods for calculating software complexity, including McCabe Cyclomatic Complexity³. Restructuring the existing software may lessen the complexity without altering the functionality of the software. Emphasis should be placed on eliminating "goto" statements and code that is never executed. The software can also be reverse engineered to a high-level design which is then restructured and used to generate a more concise and efficient implementation of the software. The reverse engineering process must be verified to insure that the recovered design accurately captures the existing software. This process is preferred when there are additional implementation changes desired, such as converting to the Ada programming language.

Data structure describes the variables, constants, complex structures (records, linked lists), and user-defined constructs in which data is stored in the software. It also includes the naming conventions utilized in the original software implementation. Data design is the key structural component in most information systems, requiring extensive database management. The largest percent of functionality in most information systems is performed within the context of data management. The data architecture is often the driving force behind the software control flow architecture. For this reason, emphasis should be placed on the data design because it will force modifications to the process design [Ruhl91, p17]. Redocumentation may provide insight into the current structure of the data, generating tables of variable names, and identifying within the software where the data is used and modified. Steps should be taken to rename data to more informative terms as permitted in the current software implementation [MITR92] and to adhere to DoD standards concerning data naming conventions. Reverse engineering the data model and integrating an efficient data management facility will provide the biggest payoff for most information systems [MITR92]. Improved naming conventions

³ McCabe, Thomas J, "A Complexity Measure," IEEE Transactions on Software Engineering, December 1976, pp. 308-320.

may only be achieved by reimplementing in a new programming language, such as a language that does not limit variable name length or characters that can be used. Migration to advanced hardware platforms also can improve data structure and naming conventions through increased memory capacity.

Languages describe the programming languages used to implement the software. For many information systems the COBOL programming language was used and sometimes a hardware-dependent assembly language. The functions of many information systems are limited in number and are repeated throughout all similar software systems. There are many automated translation tools which can be used to convert COBOL-based programs to new implementation languages. Translation is usually most successful in efforts where the goal is to solely generate a new version of the system in another language that is similar to the current language or a different version of the same language. A translation strategy is most successful with small programs where the software architecture and data structure will remain the same. Reverse engineering to a language-independent design is an alternative to translation which may be more time-consuming and expensive. This design can then be used to generate a more efficient representation of the software, taking advantage of the new language constructs. Translation does not incorporate alternative implementations which use the unique features of the target language. This must be accomplished through manual conversion or restructuring techniques.

Modularity refers to the ability to identify and isolate unique functionality within the software. An organization should examine all software systems that it maintains for functional commonality and consider eliminating redundancy and minimizing maintenance by combining functionality into single systems. Each application system should be evaluated with the intent of discovering what is worth retaining for future use and what is not [Ruhl91, p17]. Organizations need to identify the important functions within software systems that should be maintained as legacy. The designs and implementations of these requirements should be isolated and examined for potential software reuse. The implementations can be reverse engineered to capture the design for future use. Implementations that serve important functions may be restructured to achieve the optimal implementation for multiple use. All of these components should be well-documented.

Size defines the volume of the software in some standard unit of measure. There are several sources for measuring the size of software systems, including Albrecht and Gaffney⁴, and the Software Engineering Institute (SEI). The most common measurement is source lines of code (SLOC), although critics argue as to its usefulness in software planning and control [Keme93, p87]. Function points are another unit of measure that delineates size through specifying functions which the software performs as a way of measuring the magnitude of the software. Other units of measure that have been proposed include number of modules, objects, bytes, and number of files. Consistent measurements and the ability to adjust to other defined units

⁴ Albrecht, A., and J. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction," IEEE Transactions on Software Engineering, vol SE-9, no. 6, November 1983, pp. 639-648.

of measure is the most important issue when comparing data. Cost and schedule will also be impacted by the size. In general, reengineering processes that are easy to automate will be more successful on smaller software modules. More manual efforts will be required on larger modules.

Software change rate describes the average number of modifications, both corrective and perfective, performed on a software system during a given period of time. This rate is often indicative of the number of defects in the software, or may reflect the variety and volume of users. Assessments should be made to determine what the needs of the user are relative to the functionality of the software. An ever-changing software system should be quickly migrated to a more efficient support environment to better adapt to the needs of the user. Unnecessary functionality should be eliminated, and desired functionality improved. Reverse engineering to a Computer-aided software engineering (CASE) environment can provide automated support and is also useful generating a more modern system.

Software importance is determined by the number of people or organizations which utilize the software system. Users of the system also include external automated systems with which the candidate system interfaces. Examples of highly critical software systems include those that perform functions in no other software system, those which could not easily be replaced with another system, and those which would have a detrimental effect on the organization if they were to be eliminated. A payroll system could be considered a highly critical system, since its elimination would have an enormous impact on the organization should the employees experience a delay in their pay. If the system performs life-threatening functions or unique functions which no other system performs, then the system is also highly critical. Information dependencies between the candidate system and external systems, as well as other systems within a like domain should be identified [Ruhl91, p15]. The organization should make a determination as to why the system is important and consider redevelopment or reverse engineering, while the system remains in use. This system should probably be an example system to analyze from a functional viewpoint as a basis for determining new technology that will improve the overall current business practices of the organization [Ruhl91, p15]. The cost of reengineering these systems should be weighed heavily against the importance of its functions and ample support should be given to improve and secure this type of software system for extended use. Highly critical systems should be further examined for consolidation to minimize the number of overall systems which the organization must support.

Structural quality describes the structure or architecture⁵ of the software system. The software architecture may be well-structured and suitable for quickly converting to a modern programming language or hardware platform. In this case, translation can be an effective means for performing this conversion. Poorly structured software is a candidate for code restructuring or for reverse engineering to produce a design representation that can then be restructured [MITR92]. Poor structure may cause automated tools difficulty in analyzing the

⁵ Architecture includes the implementation design of the software. The term Design is used to define a criteria under Development Factors that refers to the existence of a high-level representation of the software.

source code. Time may be wasted on analyzing and reverse engineering dead code that is of no value to the system requirements. It may be advantageous for a team of programmers to examine the code manually, identifying such areas in the code and perhaps to manually discard or modify them. Redocumentation techniques can often report on the structural quality of source code. Automated redocumentation exists that generates structure charts⁶ which define the procedures used to implement the software, including the calling hierarchy, naming conventions, and input and output of data between these modules. This is useful in quickly identifying modules which are never executed, are not represented in an existing high-level design, or utilize global data [STSC92].

4.1.1.2 System Characteristics. System characteristics describe any commercial or external software packages that integrate with source code to form a complete system. These characteristics also include alternate configurations in which the system exists for different customers and external software applications with which the software interacts to meet high-level requirements. The overall requirements that this system performs that may be outdated and the importance of the system to the organization are also part of this criteria.

<u>System Characteristics</u>	<u>Description</u>
Alternate configurations	whether or not there are alternate configurations of this system; if so, a description of these configurations and scenarios for when each is an alternative
Commercial software interface	number of commercial software packages that interface; and a description of that interface
Requirement obsolescence	to what degree the system requirements are no longer viable; which requirements are viable

Alternate configurations identifies whether or not there are alternate configurations of this software system. The requirements justifying these versions should be identified and it should be determined whether the alternate configurations can be consolidated into a single software system. Restructuring the system may improve modularity for identifying specific functions and enable a system to incorporate functions previously performed in other systems. Reverse engineering permits the consolidation of these functions into a uniform design representation that can then be used to generate the new software system.

Commercial software interface describes the number of commercial software packages that interface the software system. It is important to identify all of these interfaces and understand the structure of each to insure that any reimplementations or modifications to the existing software does not alter this interface. Reengineering is considered for improving poorly integrated system components [MITR92]. Most modernization efforts integrate commercially

⁶ Structure charts depict the structure of the software as defined by Edward Yourdon and Larry Constantine in Structured Design, Englewood Cliffs, NJ: Prentice Hall, 1979.

available software components, including database management systems (DBMS) and other software packages to meet system requirements. The key issue in most of these integration efforts is the data interchange format. Commercial packages do not easily interact with older programming languages of hardware platforms, thus requiring conversion to new software and hardware implementations. Reverse engineering the software to a design representation provides a means for achieving this type of conversion. The design can be restructured to better integrate commercially available system components and reimplemented in modern programming languages for state of the art hardware.

Requirement obsolescence describes to what degree the system requirements are no longer viable. Current system functionality may not conform to user needs. Obsolete requirements should be eliminated from the existing system during reengineering. Source code which performs these requirements should be extracted from the software prior to restructuring if possible, else afterwards. Reverse engineering to the design level may more easily identify the portions of the software performing these outdated requirements and can easily be removed from the design prior to reimplementation.

4.1.1.3 Environment Characteristics. Environment characteristics describe the organization's primary functions with respect to how the software system operates. Domains of interest which the organization supports are defined within this information system(s), including personnel management, financial, and procurement. The software system executes on computer hardware and possibly interfaces with other hardware components. The high-level organizational goals which this system supports also reflect the operational environment. The number and type of external interfaces and how the system is used are also part of the operational environment.

Environment Characteristics

Description

Domain consistency

whether or not the system exists within a domain and the description of that domain; whether or not the system needs to fit within a domain and a description of that domain (e.g., personnel, financial)

Hardware interface

definition of the hardware upon which the system depends and a description of this interface; whether or not the system can reside on more than one hardware/software platform; identification and description of those platforms if currently residing on more than one

Organizational goals

describes the high-level goal(s) of the organization relative to this software system and the function this system plays in accomplishing this goal(s).

Usage

number and type of interactions with the system by individuals, other organizations, and external automated systems

Domain consistency describes whether or not the system exists within a domain and the description of that domain. Modifications to the system must conform within the constraints of this domain. Often reverse engineering is performed to migrate a software system to a domain that provides consistency across similar applications in an organization. Reimplementation of the new system should consider reuse options for achieving domain consistency as well as consolidation of functionality across multiple software systems.

Hardware interface describes the current hardware platform upon which the software system executes and a description of the dependencies of the software on this platform. The hardware capabilities often restrict software implementation options and these may change given an alternative platform. Antiquated hardware is often the reason for modernizing the software system [MITRE92]. Fewer dependencies may ease migration to a new hardware configuration, while strong ties may impact the technical approach taken to modernize the software. Reverse engineering is useful for generating a hardware-independent design of the software and determining implementation dependencies which resulted from that platform. The hardware-independent design can then be implemented for alternate hardware configurations.

Organizational goals describes the high-level goals of the organization relative to the software system and the function this system plays in accomplishing these goals. Room defines the high-level goals of most organizations to include the elimination of non-essential products and increase the value of those remaining. [Room92]. Reengineering should only be performed after careful analysis of how the software systems within an organization currently support these primary functions. Keyes quotes C. Finkelstein⁷, who said, "Legacy code contains the business rules of an organization at a particular time" [Keye92, p39]. Since the organization must adapt to the changing environment to stay competitive, the software system must allow the flexibility to adapt as well. Software systems which no longer support the organization's primary functions and goals should not be considered for reengineering; these systems are seldom used and should be discarded. Systems of minimal to moderate use that do not support these activities should be further examined for individual component importance. Relevant components should be identified, isolated, and stored for possible integration into other systems of importance in software reuse. Only those systems addressing the current primary functions and future goals should be considered for reengineering.

Usage describes the amount of use the software system undergoes, including the number and type of interactions with the system by individuals, external organizations and other automated systems. The needs of these entities must be maintained when modernizing the system. In many cases it is precisely these needs that are driving the reengineering effort. Modernizing the system promises to improve response times by the system as well as by maintainers responding to change requests from the users. Reverse engineering to an

⁷ Clive Finkelstein is the founder of Information Engineering Systems Corporation (IESC), Alexandria, VA. He writes, lectures, and espouses methodologies for information engineering, and has recently completed Information Engineering: Strategic Systems Development, Addison-Wesley Publishers, 1992.

automated support environment helps improve the modification process. Redocumentation provides supporting documentation to maintainers making decisions about modifications. Restructuring the software may enable the system to more readily accept modifications without negatively impacting other parts of the system.

4.1.2 Process Factors. Process factors describe the aspects of the original development process and the maintenance process which have impacted the current implementation of the software system.

4.1.2.1 Development Factors. Development factors define aspects of the development process which have affected the software system implementation. The development process for a software system is key to how well it can be maintained throughout its life-cycle. The age of the software can indicate the software engineering practices employed at the time of development and reflects the usefulness of the software. The availability of original developers is helpful for unraveling the mysteries of design and implementation decisions. The use of documented standards in the development process may also have supported the creation of a quality software system.

<u>Development Factors</u>	<u>Description</u>
Design	comprehensive high-level design of the software that is consistent and complete to the current implementation
Development methodology	whether or not a well-defined development procedure was used; whether it is documented in text available to the public; whether or not this methodology is supported by automation
Documentation	supplemental documents including reports, tables, users guides which describe aspects of the system not necessarily relating to the design
Standardization	use of well-documented development standards, continued throughout maintenance

Design is a comprehensive high-level representation of the software that is consistent and complete to the current implementation. If this representation exists, it may be used or restructured to reimplement the software in a standard development process. If the existing design is not consistent or complete, it may be used as supplemental documentation during a reverse engineering process which will produce a more comprehensive and current representation. Reverse engineering is used to generate designs for software systems that do not have an available design that is complete or consistent with the current implementation of the software system that is useable in a effective maintenance process. This may include designs that were not originally developed using structured modeling techniques or are not supported by an automated process (automation provides necessary efficient maintenance

support) [McCa92]. The design may serve as an end product that provides system understanding or may serve as a means for implementing a new system with added functionality or new programming language. Reverse engineering the data model for information systems often lends insight into how to restructure the data for an improved implementation.

Development methodology identifies whether or not a well-defined development procedure was used in developing this software. If a documented methodology was used, then it may be easier to understand the impact of this methodology on the implementation of the software and the design of the software, if it exists. It is also important to note if automation was used in developing the software system. If the previous development procedure is still a viable development methodology, then consideration should be given for utilizing this method in regenerating the software system.

Documentation describes any reports, tables, or design record that exists for the software system. Documentation is credited with being the key to adequate software maintenance, and yet it is still the most unqualified segment of the software engineering process.

There are useful suggestions that can be performed immediately to improve the documentation status of a software system [Hova92]. Obsolete documents, such as those that have not been used in more than a year, should be discarded since they are either outdated or unusable. Perform a documentation audit to determine what documents are needed and set about generating these documents or making additions to existing incomplete documents. This can be performed by examining trouble reports that were diagnosed as help requests, or the user help requests, if these are logged. Finally, new development, as well, as reengineering should establish documentation as a top priority in these processes. A lack of adequate documentation is the most common reason for considering reengineering [Ruhl91]. Redocumentation techniques produce various types of documents to supplement the software system implementations (specification, design, source code). Reports defining the variables, procedure calls, and procedure interfaces can be useful in gaining a better understanding of the software. Documentation supports reverse engineering by providing supplemental information that is useful in distinguishing design and requirements information from implementation idiosyncracies.

Standardization describes the utilization of standard development techniques during the original development process. These techniques may be proprietary standards, internal to the organization, or may be DoD standards (e.g., DoD-STD-2167 or the proposed MIL-STD-SDD). The standards used in the development of the existing software may provide insight into why design and implementation decisions were made, and to where modernization can be applied to the software. No utilization of documented standards indicates that the software should probably be converted to an implementation that adheres to current software engineering standards. This can be achieved through translation (for language conversion), or reverse engineering to a more open systems environment.

4.1.2.2 Maintenance Factors. Maintenance Factors define aspects of the maintenance process which have affected the software system implementation. A common reason for reengineering is often an inability to adequately maintain the existing software [MITRE92]. A good development process can easily be weakened by poor maintenance practices. Many systems have survived years of unmonitored modifications to the software, resulting in unstable implementations. The criteria to consider concerning maintenance are listed below.

<u>Maintenance Factors</u>	<u>Description</u>
Automation	whether or not an automated process is used for modifications
Configuration management	a management process for controlling and documenting modifications and versions of the software system
Improvement status	to what degree the system could be improved to meet current known requirements and a description of these improvements; to what degree the system could be improved to meet current known requirements efficiently.
Life expectancy	the number of years this system is expected to remain in operation
Modification effort	rate of modification over number of modifications multiplied by the average number of maintainers to complete
Modification rate	number of modifications per a given period of time relative to a desired number

Automation factor identifies whether or not an automated process is used for supporting the existing system configuration. Incorporating automation is usually a high-level goal of most organization since it can increase the efficiency of many processes, including development and maintenance [Room92]. Automated tools can assist the maintenance process, including language sensitive editors for performing source code modifications and automated configuration management to provide version control and document the changes made to the software. Preparation for maintenance can begin in the development process. Computer-aided software engineering (CASE) tools which can be used to design and develop software, should also be used to maintain it. Redocumentation is the most mature automation capability. Redocumenting the software using an automated tool can provide fast information about the source code architecture. Reverse engineering the software into a CASE tool serves to automate the maintenance process.

Configuration management identifies whether or not a formal process exists for controlling and documenting modifications to the software and alternate versions of the system. This process may document the information helpful in identifying problem areas in the system, those which are modified or corrected often. These functions may be isolated and implemented in a modern software system.

Improvement status identifies to what extent the system is operating according to the current system requirements. There are several methods for measuring faults in source code, including Gaffney⁸. If there are current defects in the software or modifications that have been requested which have not been implemented, either due to modification effort or cost, this may be a good time to consider reengineering options. An analysis of the system may identify isolated parts of the software which are not performing adequately. These parts may be reverse engineered, modified at the design level, and reimplemented. The corrected parts can then be integrated back into the remaining software. Most existing source code components are so tightly interweaved, that it is usually necessary to reverse engineer the entire software system and generate a more modular software system.

This criteria also identifies to what degree the system could be improved to meet current known requirements efficiently. If the system has many modifications that are necessary in order for it to reach an acceptable level of performance, then consideration should be given for whether this software should continue as a functioning element in the organization. It should be clearly understood what changes are necessary and incorporate those into a reengineering process. Restructuring can be applied to improve the performance of the software without introducing new functionality. Reverse engineering can be used to abstract a high-level design which could then be used to incorporate new functionality that is implemented in a new software system.

Life expectancy identifies the number of years this software system is expected to remain in operation. The system may not be operating much longer because the functions are becoming outdated or because the software system has been designated for replacement. If the software is not expected to be in use much longer due to functional obsolescence, only improvements to the current maintenance process should be considered. If the software has been designated for replacement, certain reengineering capabilities are feasible. Only available resources should be applied, including automated tools which currently exist in the organization. Minimal redocumentation and restructuring should be performed to provide quick benefits for improving maintenance and extending the life of the system to insure it remains operational until replacement can be made. Extensive redocumentation, restructuring, and reverse engineering of obsolete systems should be considered only if the option of replacing the system can be upheld by revitalizing the existing one through these efforts.

Modification effort (M_e) identifies the rate of modification (M_r) over number of modifications (n) multiplied by the average number of maintainers (m) to complete ($M_e = M_r/n*m$). If the existing software is very difficult to maintain, this may be a sign that it needs to be reengineered. The software maintainers should be interviewed to determine if the difficulties are due to a lack of understanding, complexity of the software, result in "rippling effect" that leads to more problems.

⁸ Gaffney, John E., "Estimating the Number of Faults in Code," IEEE Transactions on Software Engineering, vol SE-10, no. 4, July 1984, pp. 459-465.

Software understanding may be enhanced through the generation of supporting documentation. Redocumenting the software to identify interrelationships between the software components can improve maintenance procedures. This can be useful when modifying software for predicting effects of change on other parts of the software. Restructuring the software may reduce its complexity, and eliminate unnecessary code. Both of these techniques may reduce the chances that one modification to the software may have a disastrous effect on other parts of the software. Severe problems may only be solved through reverse engineering and a complete regeneration of the software system.

Modification rate (M_r) identifies the number of modifications per a given period of time relative to a desired number. A large number of requests to correct or modify the software system may be an indication that this system is not currently meeting the needs of the intended users, has hidden defects, or is struggling to perform too many requirements. An analysis should be performed to identify what the system actually does relative to what the users of the system expect it to do. Redocumentation techniques should be used to define the functions and performance of the existing software. Critical functions that the software performs should be identified, isolated and perhaps reused. Reverse engineering the software can also help provide a better understanding of what the software does, and the subsequent design can be used to implement a new system that better meets the needs of the intended users. This new system should integrate reusable modules. Testing should be performed on the existing software based on these requirements to insure that the system is performing accurately. If the system is performing too many requirements, steps should be taken to isolate and identify the functions of the software to determine whether a more cohesive system can be developed which better fulfills the needs of the users.

4.2 Reengineered Software System Criteria

Reengineered Software System Criteria include those criteria which describe the desired products of reengineering and the factors which influence the reengineering process.

4.2.1 Product Characteristics. The Product Characteristics describe the desired characteristics of the target software, the new support environment, and the new operation environment.

4.2.1.1 Target Software Characteristics. The characteristics of the target software describe the modifications that are desired between the existing software and the new software.

<u>Target Software Characteristics</u>	<u>Description</u>
Functional improvement	number and type of functional modifications which are desired in parallel to the reengineering of this system; test suite for verifying these

Language migration	identification and description of the language implementation requested for this system
Performance improvement	identification and description of the desired performance improvements for this system; test suite for validating these
Technology insertion	identification of specific technology advances that could be implemented to improve the software system, including new commercial packages

Functional improvement describes the number and type of functional modifications which are required for this system. If the number of lines of code changing is predicted to be more than twelve percent, it is often suggested that the reengineering effort is too great, and that consideration should be given for redeveloping the software system [Keye92]. The rationale behind Keye's percentage is not clear, however, the point is to weigh the level of effort for the reengineering with that of redevelopment. Minor functional enhancement can usually be performed through the maintenance process. Reengineering is useful when minor changes are not easily made to the existing system or when there are enough changes to warrant redeveloping the system, but there are no available resources to support a complete development project.

Language migration describes the identification and description of the language implementation requested for this system. There are several reasons for reimplementing the software in a different programming language, including implementation improvements offered by the language or the desire to implement all software in a common language within an organization to ease maintenance. Within DoD, the standard programming language selected is Ada. Most modernization efforts within DoD are required to use Ada. Reengineering the existing software to Ada is one way to adhere to this mandate. If the current language implementation is similar to Ada, translation can be performed to the new implementation and if necessary manual code generation for the software which does not easily convert. Reverse engineering to a high-level design and then regenerating the software in Ada is another option. Often the existing system implementation is used as the baseline, a design, or supporting documentation which is referenced when generating a new software system in the target language manually. This may prove to be the most time-consuming effort.

Performance improvement describes the identification and description of the desired performance improvements for this system. These improvements may be achievable in the existing software implementation, and this should be considered. Although, more often these performance requirements require improved hardware or new software implementation. Improving the performance of the software may be an opportunity to reengineer to a new system implementation. Translation to a new software language for execution on a new hardware platform is one of the fastest migration techniques, however the translated software may not utilize the capabilities of the target language or the new hardware. Reverse

engineering to a high-level design which can then be targeted for a new hardware suite is one way to achieve these performance improvements.

Technology insertion is the identification of specific technological advances that could improve the implementation of the software. This criteria does not include automated support for the development or maintenance of software, which is defined in New Support Characteristics. The awareness of new technology that may support the existing functionality of the current software system or new requirements that are to be incorporated into the existing system, may be performed through reengineering [MITRE92]. Reusable components may be incorporated into the existing system to replace outdated components and improve the performance of the software. Often improved data management facilities may be incorporated into an information system which previously performed its data management in an ad-hoc or hard-coded manner. Reverse engineering the data models from the software and incorporating a commercial database may improve data management. New programming languages may be used to implement a software system that contains improved library routines, functions and data structures that better support the system requirements. Reverse engineering can salvage an existing software design, which can then be restructured and reimplemented into a more modern software system that utilizes advanced software engineering technology.

4.2.1.2 New Support Characteristics. New support characteristics define the qualities desired in a new maintenance process. The transition to a new support environment is often the reason for considering reengineering. An organizational decision has been made to maintain software systems using automation, necessitating the move of existing systems to the tool's environment. This decision may be useful in providing more efficient support for software, but the overhead of altering the software to enable this support may out-weigh the move. Issues concerning this technology insertion should be considered carefully.

New Support Characteristics

Description

Adaptability

to what extent the new support environment will enable the software system to adapt to a changing environment, i.e., enhancements, alternate software/ hardware implementations, integration of new components

Automation insertion

desire to maintain the software system using automated techniques

Domain consistency

to what extent the new support environment supports other systems in the same domain

Maintenance improvement

identification and description of specific improvements to the overall maintenance process; to what degree the current maintenance practices are no longer useful; description of the problems

Adaptability describes to what extent the new support environment will enable the software system to adapt to a changing environment, i.e., enhancements, alternate software/hardware implementations, integration of new components. These characteristics will prepare the software system for future migrations. Reengineering is often used to modernize the software to a implementation that is better suited for future migrations [MITRE92]. Restructuring the data model and software architecture may better prepare the software for future modifications. Reverse engineering the software into a CASE tool may facilitate the generation of implementations in alternative languages. Eliminating hardware dependencies when using the recovered design in a new implementation may make the software more portable for future migrations to new or alternative hardware platforms.

Automation insertion describes the automated techniques that are needed to support the new support environment. This criteria does not include implementation techniques supplied by technological advances in software development, which was addressed in Target Software Characteristics. The desire to automate the maintenance environment is often the reason for modernizing a software system [MITRE92]. CASE tools which are used to develop software systems are useful throughout the life of the software by supporting modifications to the source code while updating designs and documentation.

Steps can be taken to integrate CASE into the existing software engineering environment within an organization [CIM92]. This is most often successful in more modern software systems and mature organizations. Many programming languages and older hardware platforms do not support modern CASE tools, thus forcing the modernization of the software system itself. Reverse engineering provides a means for extracting information and populating CASE repositories with data that can be used to support the current implementation. CASE often includes code generation support for reimplementing the software into a more modern programming language.

Domain consistency describes to what extent the new support environment supports other systems in the same domain. Systems that reside in a identifiable domain should be considered for reengineering for achieving commonality through software reuse and consolidation. Redocumentation techniques can be used to identify commonality between software systems. Software system should be consolidated where possible. Reverse engineering systems within a single domain can enable the generation of a consolidated system from separate software programs implemented in varying languages and executing on differing hardware platforms. Reuse modules should be used to implement common functionality when possible.

Maintainability describes to what extent the new support environment provides improvements to the overall maintenance process. This identifies specific technological advances that are a part of the new support environment that improve the process of supporting the system software and a description of how this is accomplished. The integration of CASE tools promises to enable faster modifications to the software and automatic updates to supporting documentation. Reverse engineering is used to produce a high-level design representation that

can be stored and automatically supported within a CASE tool. The ability to make changes and generate updates to this representation and subsequently the implementation, should improve response time to change requests.

4.2.1.3 New Environment Characteristics. New Environment Characteristics describe the new operational environment to which the existing system must migrate. New operational environment is the most common cause for considering reengineering. The new environment may include the requirement to achieve domain consistency, migration to new hardware platforms, as well as new system requirements.

<u>New Environment Characteristics</u>	<u>Description</u>
Domain integration	identification and description of the new target domain
New hardware platforms	number and type of hardware interfaces which are to be integrated to this system; identification and description of alternative or new hardware platforms
New usage	new people, organizations, or automated systems that must now interface with this software system
New organizational goals	new organizational goal that must be addressed by this software system and how

Domain integration is the identification and description of the new target domain. The domain may be the programming language, hardware platform, or functional. Domains from which the current system is far removed may be more difficult to reach through reengineering and may require a completed redevelopment. A well-structured and modular software system may incorporate replacement parts that are reuse software modules that immediately migrate the existing system to a desired domain.

New hardware platforms is the identification and description of alternative or new hardware platforms. An understanding of the new hardware platform and the current software interface is crucial to the success of the system migration. This describes the number and type of hardware interfaces which are to be integrated to this system. The new hardware platform often drives the modification of the software to a new or alternate version. It may be necessary to restructure or update the version of the current implementation language to execute the software on the new hardware. Restructuring may be necessary, and replacement of new library modules to accomplish the same functionality of the existing software. This is also a time to consider an alternate implementation of the software in another programming language that is more compatible with the hardware and operating system. Reverse engineering and regeneration of the software may be useful in generating a new software system for the new hardware.

New usage describes new people, organizations, or automated systems that must now interface with this software system. These new interfaces may require functional improvement, portability to new hardware platforms, new report generation format, and the ability to reply to increased change requests. Reengineering is being considered for several software systems within DoD that have been selected for use in multiple agencies to eliminate redundancy, but will be maintained at one central location [CIM92]. Supporting organizations faced with an inability to adequately address current maintenance issues are now considering reengineering the software system to an automated environment which will improve maintenance process.

New organizational goals describes any new organizational goal(s) that must be addressed by this software system. Room presented high-level goals of an organization which eliminate non-essential products and processes, increase the value of those remaining; and increase the efficiency of those processes through streamlining, simplification and/or automation [Room92]. The organization may consider the candidate software system as non-essential and in this case the best option is to eliminate this software system. More likely, this software offers some value and should be examined for potential consolidation with other similar systems for consolidation purposes that will streamline and simplify maintenance. Automation may also be incorporated as discussed in paragraph 5.2.1.2 New Support Characteristics and 5.2.2.2 Methodology Factors criteria below.

4.2.2 Process Factors. The process for performing reengineering is dependent on the high-level goals of the organization, reengineering methodology, and the available resources.

4.2.2.1 Organization Factors. Organizational support for reengineering is imperative to the success of the effort. Establish the context for reengineering by considering the corporate goals of the organization and how reengineering could be applied to achieve this mission [Ruhl91, p15]. Reengineering promises to provide a potentially large payoff for little risk. However, the fulfillment of this promise is often very application dependent. A large portion of the billions of dollars spent over the past decade on information technology was invested in the support and upgrade of information systems, however, there have been no notable improvements in information management [Sass91, p17].

The time and expense of reengineering is often not understood by management, resulting in a loss of interest and withdrawal of support. Although estimates of five years before benefits from reengineering emerge are not uncommon, management often wants to see results in one year [Cumm92]. Benefits from reengineering have been defined according to improvements in the number of software defects, time, and costs during the life of the software [Wild91]. These benefits are often seen in as early as one to two years. Realistic estimates for resources on individual applications should be established initially and management support secured for reengineering success. Resulting benefits from reengineering should be measured in terms of quality enhancement as well as cost impact [Vowl91]. The motivations and goal of the reengineering effort should be determined up front and then a

plan for achieving success developed that outlines steps to take and markers for calculating success along the way [Ruhl91, p16].

<u>Organization Factors</u>	<u>Description</u>
Budget constraints	budget limitations and predicted allocation of this amount to various resources and phases of the intended reengineering activity(s)
Effort estimations	projected estimations regarding manpower and time for performing the reengineering activity(s)
Management commitment	definition of management expectations and limitations; schedule for identification of management objectives and proof of progress throughout the reengineering activity(s)
Schedule	establish management approved schedule for performing the reengineering activity(s), including alternative plans based on trade-offs measured throughout the activity

Budget constraints identifies budget limitations to performing the desired modifications to the software. Reengineering is often considered an inexpensive process. In fact, reengineering can be quite expensive, particularly during design recovery. Design recovery may seem time-consuming and costly initially, but can greatly impact the success of the reengineering. Budget must also account for the training of reengineering methods and tools. The cost of continuing in the current mode must be weighed against the estimated costs of reengineering to determine if the process is worth the effort. Given that a completed redevelopment is not an affordable option in modernizing the software system, there are several options within reengineering that may improve the current status of the system. Restructuring the source code is one of the least expensive means for improving the source code. Restructuring for a minimum cost can be performed to eliminate "dead code" and gotos. More advanced restructuring reconstructs the data model or control flow of the software. Redocumentation can also be performed to varying degrees of effort, including generation of reports defining variables and data structures or generating a structure chart showing the software architecture. These documents provide supporting information that may be useful during maintenance. Most expensive effort is reverse engineering for full design recovery. If the budget permits, this is one of the most comprehensive processes within reengineering providing a design which can be used to reimplement the software.

Effort estimations identifies projected estimations regarding manpower and time for performing the reengineering activity(s). Although fewer people are likely to be needed to perform reengineering, the expertise of these individuals becomes more important.

Management commitment identifies definition of management expectations and limitations; schedule for identification of management objectives and proof of progress throughout the reengineering activity(s).

Schedule identifies the amount of time available for performing the modernization of the software system. Reengineering can often be accomplished in less time than a complete redevelopment effort. There are several reengineering capabilities which can be performed to varying degrees within respective time frames that may suit a hard deadline schedule. Many capabilities are supported by automation which also improves time.

4.2.2.2 Methodology Factors. Methodologies are proposed for many reengineering activities. The appropriateness of a methodology is based on the needs of the organization, characteristics of the application, funding, and personnel capabilities. The factors below should be considered prior to selecting an appropriate methodology.

<u>Methodology Factors</u>	<u>Description</u>
Automation support	automated support for the methodology selected to perform the reengineering
Forward engineering support	to what extent the reengineering tools support the forward engineering processes; identification and description of the interfaces to these processes
Methodology selection	identify and select a well-defined methodology; insure training is available and that the methodology meets the goals of the organization; insure that the methodology is usable across domains and in support throughout the maintenance process
Reuse commitment	investigate reuse options whenever possible

Automated support identifies the automated tools which support the reengineering methodology. There are several tools available which support various reengineering activities [STSC92]. It is important to be realistic in what these tools provide. Investigate their capabilities and be prepared to address inefficiencies through adequate personnel who are knowledgeable of these tools and can perform processes when the tools do not. Considering the focus of most CASE tools for a particular computing environment, one set of CASE tools should not be depended on for uniform applicability to all needs across an organization [Ruhl91, p21]. In addition, since several tools may be used during reengineering, it is important to consider not only the impact of these tools on the existing software engineering environment in the organization, but whether or not there is data interchange capability. Good business sense should be applied when considering the purchase of expensive tools.

Forward engineering support identifies to what extent the reengineering tools support the forward engineering processes; identification and description of the interfaces to these processes.

Methodology selected identifies a well-defined methodology; insures training is available and that the methodology meets the goals of the organization. A methodology should be chosen that is compatible to the requirements of the organization and are supported by automation where possible [Ruhl91, p21]. It is cost-effective if this methodology is usable across domains and transitions the system to a better maintenance process.

Reuse commitment identifies that there is a commitment to investigate reuse options whenever possible. This is primarily a concern in forward engineering where reuse components should be utilized whenever possible to supply required system functionality.

4.2.2.3 Resource Factors. Resources for performing reengineering are composed of automated support, including hardware platforms and tools, and personnel. Reduced resources due to budget cuts are often the reason behind considering reengineering as opposed to standard development. Often the cost of reengineering is assumed to be very inexpensive and when it is not, the process is often not completed or considered a failure. Reengineering should ultimately be less expensive than redevelopment or estimates of continued maintenance practices, but realistic estimates with regards to resources are needed for successful reengineering.

4.2.2.3.1 Automation Factors. Automation can be essential to ensuring the efficient, effective renewal of software systems. Adequate storage capacity and processor speed in equipment supporting the reengineering tools are essential to facilitate the reengineering process [Ruhl91, p21]. The environment suggested to adequately support the reengineering of COBOL-based programs utilizes both personal computers and a mainframe [Dyso92].

Using equipment and tools that conform to applicable standards (e.g., POSIX) will assist in eliminating incompatibility problems and will support future migrations [Ruhl91, p16]. The primary factors that are considered in automated support are listed below.

<u>Automation Factors</u>	<u>Description</u>
Capability limitations	limitations on use of tool, i.e., maximum number of users, limitations on methodology support
Platform support	insure that hardware/operating systems and integration to existing or selected reengineering platform environment; identification of platforms supporting tools and the limitations of this support
Target hardware support	relationship of tool to hardware platform of reengineered product (does the target hardware itself support the tool, does the tool support the development of software for the target hardware)
Tool availability	does the tool currently exist in the organization, is the organization capable of acquiring this tool, understanding the cost and time for procurement

Capability limitations identifies the limitations of available tools. The limitations of the methods and tools must be understood up-front in order to accommodate for automation inefficiencies by qualified personnel.

Platform support identifies the hardware/operating systems and integration to existing or selected reengineering platform environment; the platforms supporting tools and the limitations of this support.

Target hardware support identifies the relationship of tool to hardware platform of reengineered product (does the target hardware itself support the tool, does the tool support the development of software for the target hardware).

Tool availability identifies whether the tool currently exists in the organization, is the organization capable of acquiring this tool, understanding the cost and time for procurement.

4.2.2.3.2 Personnel Factors. Key to the success of reengineering is the availability of persons knowledgeable in both the existing system, target system, and the reengineering process. It is critical that the existing system experts be involved throughout the reengineering process for accomplishing correct design recovery [Ruhl91, p23]. There must be commitment from these experienced personnel to assist throughout the reengineering effort.

As with any technology, true productivity improvements can only be realized through a balance of both enhanced technology and appropriate staffing of individuals with the specialized knowledge to use this technology [Sass91, p20]. Reengineering requires a highly trained staff that has experience in the automated tools and the specific programming languages to be used in the reengineering effort [Ruhl91, p22]. The inefficiencies of available tools can often be compensated by experienced personnel. The Criteria listed below addresses the availability of experienced personnel who take part in the reengineering effort.

<u>Personnel Factors</u>	<u>Description</u>
Existing system expertise	does the personnel exist with the expertise on the existing system and are they committed to the reengineering effort
Target system expertise	does the personnel exist with the expertise on the desired target system and are they committed to the reengineering effort
Reengineering expertise	does the personnel exist with the expertise on the methodology and automated tools for performing the reengineering activity(s) and are they committed to the reengineering effort

Existing system expertise identifies the personnel with the expertise on the existing system. During reverse engineering and redocumentation, these experts should verify that the

recovered design and support documentation accurately portray the software system. They should also be consulted to clarify complex or confusing aspects of the software system throughout the reengineering process. During restructuring these individuals should be used to verify that this process is not altering the functionality of the software system (during restructuring, existing and target system expertise are usually the same individuals since only the performance aspects of the system are altered and functionality remains the same).

Target system expertise identifies personnel who possess the expertise on the desired target system. These individuals should verify new requirements for achieving the target system and should be consulted to clarify any confusing aspect of the proposed target system. During forward engineering these individuals should be consulted to insure the new system meets all requirements, including functional and performance. During restructuring, these individuals insure that performance has been improved.

Reengineering expertise whether the personnel exist with the expertise on the methodologies and automated tools for performing the reengineering activity(s). These individuals should work with the appropriate experts to match the technical approach for reengineering with the characteristics of the software system and the desired target system. If these experts are not available, the reengineering strategy must supply appropriate training for the methods and tools associated with the reengineering strategy.

5. APPLICATION AREAS FOR SELECTION CRITERIA

A selection criteria can be used when (1) selecting a system from a set of functionally homogeneous systems and (2) selecting a system from a set of heterogeneous systems.

5.1 Homogeneous Application

Homogeneous systems are identified by their similarities in functionality through domain analysis. Examples of these domains include Personnel, Finance, and Payroll. The selection among homogeneous systems may support consolidation efforts or elimination of multiple configurations. The criteria which are important in this selection include, but are not limited to the following:

- portability of system to alternate hardware platforms
- existence of alternate configurations for this system
- ability of system to meet domain requirements
- how well system performs
- potential new users of system

5.2 Heterogeneous Application

A single organization may be responsible for many systems that are functionally very different (heterogeneous). This organization may have decreasing resources to adequately support these systems, including budgets, personnel, and computer resources. Selecting a system for reengineering among heterogeneous systems is often dependent on some of the following criteria:

- Systems ability to meet current organizational goals
- Amount of use this system experiences
- Life expectancy of this system (from a functional view)
- Effort to modify system
- Number of modifications system has experienced
- Impact on organization's maintenance costs

5.3 Results of Effort to Define Selection Criteria Method

The results of CIM/XE's effort to identify a selection criteria provided insight into the way systems are currently chosen as candidates for reengineering. Two important conclusions include:

- (1) the current process for identifying information systems as candidates for software reengineering is tool-dependent and ad-hoc; and
- (2) data from reengineering efforts is needed to validate a selection criteria for identifying information systems as candidates for software reengineering.

The current process for identifying automated information systems for software reengineering is constrained by the capabilities of available reengineering tools. The criteria currently used to select information systems insures that available reengineering tools will work effectively on these information systems. Many information systems in DoD, however, typically have very different characteristics than those often selected for reengineering pilot projects. Table 1 depicts a comparison of these characteristics.

Characteristics of Systems in Reengineering Pilot Projects	Characteristics of Typical Information Systems Within DoD
small in size Cobol-based no external software interfaces well-defined functionality no additional functionality is required no interface with operating system	very large (100,000+) may include assembly or proprietary languages interfaces with external software unknown functionality/lacks documentation additional functionality is needed calls to operating system routines

These characteristics indicate that software reengineering is limited by the capabilities of automated tools. The technology is still immature and not well-defined. The proof that current software reengineering technology is capable of fully supporting large-scale conversion efforts is not shown.

Project data from reengineering efforts should help define a selection criteria for identifying information systems that benefit from reengineering. Currently, there is not enough data or the type of data to support a formal selection criteria for selecting information systems for software reengineering. As a result, the CIM/XE work succeeded in identifying the type of data which appears to impact the software reengineering process. This data forms a proposed selection criteria. We are identifying proposed reengineering efforts that could be used to collect the data needed to validate and evolve the proposed selection criteria.

6. CONCLUSION

Ultimately, the process of reengineering must support the high-level goals of an organization, including (1) elimination of non-essential products and processes, (2) increase the value of those remaining; and (3) increase the efficiency of those processes through streamlining, simplification and/or automation [Room92]. Two broad concepts serve as the guide for reengineering technology development, including the prevention of unnecessary duplication by joint use of personnel, information systems, facilities, and services across DoD and the conformance to new regulations, policies, standards, and guidelines for software acquisition and support. The Selection Criteria provides insight into which information systems might benefit from reengineering technology and how.

The Software Reengineering Criteria was developed using data gathered from completed reengineering projects. To date there is little documented experience estimating the effort involved in reengineering. Future plans are to gather data from reengineering projects to validate the Criteria as a means for identifying software reengineering candidates. The Criteria will decrease in number, while the understanding of its impact on software engineering becomes more concrete. The application of the Criteria will provide information useful in defining metrics and support for cost/benefit analysis for software reengineering. As experience builds, so will the understanding of how reengineering should be performed. This document serves as the basis for such guidance and provides information that will benefit organizations considering reengineering technology.

List of References

- [Boeh81] B.W. Boehm, Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ, 1981.
- [Chik90] E.J. Chikofsky and J.H. Cross, "Reverse Engineering and Design Recovery: A Taxonomy," IEEE Software, pp. 13-17, January 1990.
- [CIM92] I-CASE Readiness for I-CASE Pilot Projects (Draft Plan), Defense Information Systems Agency, Center for Information Management, Arlington, VA, November 1992.
- [Cumm92] J. Cummings, "Reengineering is High on List But Little Understood...," Network World, July 1992, v9, n30, p27.
- [DoD92] Functional Management Process for Implementing the Information Management Program of the Department of Defense, DoD 8020.1-M (Draft), August 1992, Director of Defense Information, Office of the Secretary of Defense.
- [Dyso92] N. Dyson-Hudson, "Taming the COBOL Maintenance Monster," Computer Language, September 1992, v9, n9, p40(4).
- [Hobb91] R.L. Hobbs, J.R. Mitchell, and G.E. Racine, System Re-engineering Project Executive Summary, ASQB-GI-92-003, Nov 1991.
- [Hova92] H. Hovaness, "It's Here Somewhere," Corporate Computing, Sept 1992, v1, n3, p45(2).
- [JLC92] Department of Defense Joint Logistic Commanders (JLC) Joint Policy Coordinating Group on Computer Resources Management (CRM), Proceedings of the Santa Barbara I Workshop, Santa Barbara, CA, September 21-25 1992.
- [Kerr91] J. Kerr and T. McGovern, "The Three R's of IS: Demystifying the Reverse Engineering Revolution," Database Programming & Design, Oct 1991, v4, n10, p19(3).
- [Keme93] C.F. Kemerer, "Reliability of Function Points Measurement," Communications of the ACM, February 1993, Vol. 36, No. 2, p85(12).
- [Keye92] J. Keyes, "Code Trapped Between Legacy, Object Worlds," Software Magazine, June 1992, v12, n8, p39(5).

- [McCa92] T.J. McCabe and E.S. Williamson, "Tips on Reengineering Redundant Software," DATAMATION, April 15, 1992, v38, n9, p71(4).
- [MITR92] MITRE Corporation, "Lessons Learned: Re-engineering the Weighted Airman Promotion System for the CIM Environment," Draft Report under contract number DAAB07-91-C-N751, Software Engineering Center, MITRE Corporation, dated 30 September 1992.
- [NIST] National Institute of Standards and Technology, "Guidance on Software Maintenance," NIST Special Publication Number 500-106.
- [Perr92] W.E. Perry, "Follow These Three Steps for Downsizing Applications," Government Computer News, July 6, 1992, v11, n14, p20.
- [REH92] MIL-HDBK-REH (DRAFT), Department of Defense Reengineering Economics Handbook, Sept 25, 1992.
- [Room92] L. Roomets, "Integrating CASE with Business Process Re-Engineering," Proceedings CASE WORLD, Sep 30 - Oct 2, 1992, pD9-1 to D9-13.
- [Ruhl91] M.K. Ruhl and M.T. Gunn, "Software Reengineering: A Case Study and Lessons Learned," NIST Special Publication 500-193, National Institute of Standards and Technology, September 1991.
- [Sass91] P.G. Sassone, Office Productivity: The Impacts of Staffing, Intellectual Specialization and Technology, Sept 1991 (rev. Nov 1991).
- [Sitt92a] C. Sittenauer and M. Olsem, "Time to Reengineer," Crosstalk, Software Technology Support Center, Hill AFB, Utah, March 1992.
- [Sitt92b] C. Sittenauer and M. Olsem, "Reengineering Options Analysis," Proceedings of the Third Annual Systems Reengineering Technology Workshop, Naval Surface Warfare Center, Silver Spring, MD, August 11-13, 1992, pp. 23-31.
- [STSC92] Software Technology Support Center, Reengineering Tools Report, Hill Air Force Base, Utah, 1992.
- [Stev92] B.J. Stevens, "Linking Software Re-engineering and Reuse: An Economic Motivation," Crosstalk, August 1992, pp. 13-20.
- [Vowl91] J. Vowler, "A Reengineering Union," Computer Weekly, Nov 7, 1992, p20.
- [Wild91] C. Wilder, "Measuring the Payoff From Reengineering," Computerworld, Nov 18, 1991, v25, n46, p65.

Other References

DoD 8020.1-M	Functional Process Improvement: Functional Management Process for Implementing the Information Management Program of the Department of Defense.
DoD-STD-2167A	Defense System Software Development
DoD-STD-2168	Defense System Software Quality Program
MIL-STD-1815A	Ada Programming Language
MIL-STD-7935A	DoD AIS Documentation Standards
MIL-STD-SDD (DRAFT)	Software Development and Documents

APPENDIX A

GLOSSARY OF TERMS

GLOSSARY

Forward Engineering. Within the context of reengineering, forward engineering is the software engineering activities that consume the products of reengineering activities (primarily reverse engineering), reuse, and new requirements to produce a target system.

Redocumentation. The creation or revision of a semantically equivalent representation within the same relative abstraction level.

Reengineering. The examination and alteration of an information system to reconstitute it in a new form. The process encompasses a combination of other processes such as reverse engineering, restructuring, forward engineering, redocumentation, and translation.

Restructuring. The transformation from one representation form to another at the same relative abstraction level, while preserving the information system's external behavior (functionality and semantics).

Reverse Engineering. The process of examining an information system by analyzing its documentation, application software, and data structures within the environment in which the information system operates.

software. within the text of this document refers to the custom programs implementing a given information system.

Software Reuse. The process by which existing software work products (which may include not only source code, but also products such as documentation, designs, test data, tools, and specifications) are carried over and used in another new development efforts, preferably with minimal modification.

software system. within the text of this document refers to the software, as well as any database or other Commercial (COTS) package with which this software interfaces to perform the requirements of the information system, independent of the hardware platform on which these components execute.

Translation. Transformation of source code from one language to another or from one version of a language to another version of the same language.

APPENDIX B

EXISTING SOFTWARE REENGINEERING GUIDANCE SOURCES

Existing software reengineering guidance sources are the Software Technology Support Center [Sitt92], the Joint Logistics Commanders Santa Barbara I Workshop [JLC92], and the National Institute of Standards and Technology [Ruhl91].

STSC's Reengineering Candidate Selection Process

The STSC's Reengineering Candidate Selection Process is based on a set of weighted questions [Sitt92b, p24] that are dependent on three variables: complexity, importance, and longevity. Complexity and importance of the software system are calculated in terms of low, medium, and high. The longevity of the system is estimated in terms of short (less than six months), medium (greater than six months and less than 3 years), or long (greater than 3 years). The following section summarizes the STSC process.

Regardless of the complexity and importance, if the remaining life expectancy is less than 6 months, then the system should not be reengineered. If the complexity and importance are low and the system's life expectancy is less than 3 years, then the system is also not suitable for reengineering. The remaining combinations of these three variables suggests performing activities associated with reengineering that extend from simple reformatting to a more complex restructuring of the software architecture, possibly including redocumentation, to reverse engineering followed by forward engineering, and finally a complete redevelopment.

Reformatting is suggested for systems whose life expectancy is greater than 6 months and less than three years (medium life expectancy) and when the complexity is medium in a system of little importance, or when the complexity is low in a system of medium importance. Systems with life expectancies greater than 3 years (long life expectancy) which are not complex and are of little importance are also candidates for reformatting. Reformatting primarily serves to improve maintenance by cleaning up the appearance of the source code. It does not alter the basic software architecture and requires little effort. There are several reformatting tools available, including language sensitive editors and pretty printers.

The software architecture and data structures are optimized during the more complicated process of restructuring. Systems of medium life expectancy which are highly complex and of low to medium importance are good candidates for restructuring. Systems of long life expectancy that are not complex and are of medium importance should also be restructured. In addition to restructuring, redocumenting is also a helpful reengineering process that extends the life of a software system. Candidate software systems for restructuring in conjunction with redocumenting include those with long life expectancy that are medium to highly complex with low importance.

Reverse engineering followed by forward engineering is the most time-consuming, but potentially the most beneficial activity within reengineering. This activity is usually limited to systems of great importance to an organization, since it is expensive, requiring commitment

of resources (tools, personnel) to be successful. Among very important systems with medium life expectancy, regardless of complexity, reverse engineering is the suggested choice of reengineering processes. Long life systems of medium importance that are medium to highly complex and long life systems of great importance that exhibit low to medium complexity are also likely candidates for reverse engineering. In conjunction with reverse engineering, redocumentation is always suggested for highly complex systems. Not all candidates are appropriate for reengineering. A complete redevelopment may be necessary for the highly complex, very important system which is expected to have a long life.

The JLC Santa Barbara I Workshop

The JLC Santa Barbara I Workshop generated a list of the most critical criteria determining when to reengineering [DOD92] (Figure A-1). A candidate application is rated a value 1-4 as to how well the application meets each criteria.

THE MOST CRITICAL CRITERIA DETERMINING WHEN TO REENGINEER

CRITERIA	1	2	3	4
Technical Quality				
Cost				
Complexity				
Mission Requirements				
Importance				
Political				
Life-Cycle Remaining				
Change of Platform				
Too Hard to Maintain				
Budget Availability				
User Satisfaction				
Maturity Level				
Documentation out of Date/Missing				
Reliability				
Maintenance Change Rate				
Design Change				

FIGURE A-1. SB-I Reengineering Economics Critical Criteria.

The draft Reengineering Economics Handbook (MIL-HDBK-REH) defines a

Reengineering Decision-Making Process that is based on the STSC's Reengineering Candidate Selection Process. There are two noteworthy differences between the original STSC Process and the Process as it appears in the draft MIL-HDBK-REH. First, the variable called "importance" was renamed "environmental risk." The weighted questions for this variable are essentially the same. Secondly, the appropriate reengineering strategy(s) to follow after applying this Process are presented in two charts: one for medium life expectancy (Figure 3-1) and one for long life expectancy (Figure 3-2). The Process suggests that software with a remaining life expectancy of less than 6 months should not be reengineered.

Complexity

High Medium Low	Restructure	Reverse then Forward	Reverse then Forward	Reverse then Forward
	Restructure	Restructure	Restructure	Reverse then Forward
	Reformat	Reformat	Restructure	Restructure
	Leave Alone	Leave Alone	Leave Alone	Restructure
Low Medium High Environmental Risk				

FIGURE A-2. Santa Barbara I: Medium Lifetime Remaining.

Complexity

High Medium Low	Restructure Redocument	Reverse then Forward (P)	Redevelop	Redevelop
	Restructure Redocument	Restructure	Restructure	Reverse then Forward
	Reformat	Reformat	Restructure	Restructure
	Leave Alone	Leave Alone	Leave Alone	Restructure
Low Medium High Environmental Risk				

(P) - prototype

FIGURE A-3. Santa Barbara I: Long Lifetime Remaining.

National Institute of Standards and Technology Case Study

The recommendations which resulted from the NIST Case Study [Ruhl91, p15-23] include the following:

Establish the context for reengineering by considering the corporate goals of the organization and how reengineering could be applied to achieve this mission. Information dependencies between application systems must be identified [p15].

Analyze system requirements from a functional viewpoint and consider new technology to improve current business practices [p15].

When procuring equipment, require conformance to applicable standards (e.g., FIPS) to achieve flexibility and ease in future migrations [p16].

Identify motivations and what is to be achieved by reengineering [p16].

Evaluate application system with the intent of discovering what is worth retaining for future use and what is not [p17].

Stress data design because it will force modifications to the process design [p17].

Evaluated code, documentation, maintenance history, and appropriate metrics to determine the current condition of the system [p19].

While design recovery is difficult, time-consuming, and essentially a manual process, it is vital for recovering lost information and information transfer [p19].

Identify critical system information. Do not be tightly tied to a certain set of documentation forms; focus on information content and usage [p20].

Provisions in terms of personnel and effort must be made to compensate for the lack of full support of the reengineering process by currently available off-the-shelf tools [p21].

Considering the focus of most CASE tools for a particular computing environment, one set of CASE tools should not be depended on for uniform applicability to all needs across an organization [p21].

Adequate storage capacity and processor speed in equipment supporting the reengineering tools are essential to facilitate the reengineering process [p21].

Consider CASE reengineering tools that provide methodologies which are compatible to the requirements of the particular enterprise [p21].

Additional features that merit consideration include a data interchange facility and appropriate metric analysis utility [p22].

Reengineering requires a highly trained staff that has experience in the current and target system, the automated tools, and the specific programming languages [p22].

It is critical that the application system experts be involved throughout the reengineering process. They are essential for design recovery [p23].

APPENDIX C

CASE STUDY DATA

Perhaps the best way to determine what factors influence the reengineering process is to look at actual reengineering efforts as case studies. Only recently have there been reengineering efforts of the magnitude to produce data that is useful in predicting the success of reengineering similar applications. Some of these efforts have been documented [Hobb91, Ruhl91, MITRE92]. The documented reasons for selecting these software systems for reengineering varied, but the expected benefits listed below were the same. A summary of the issues involved in these case studies is presented below.

Reasons for Considering Reengineering

The reasons for considering reengineering vary from anticipated support for continued maintenance to support for development of modernized systems. Existing systems are also being examined for consolidation, by commonality among functions and redundancy. Another reason for considering reengineering is to analyze the technology of reengineering for determining its potential benefits.

Maintenance support. Improved maintenance was one reason for reengineering in many of the case studies examined. Manual processes were often used to support existing systems. It was important to provide the ability to respond quickly and correctly to change requests. Existing systems were being modified more to keep pace with the changing needs of the users since fewer new systems were being developed. Reengineering the system to an implementation which is more maintainable or to a CASE environment for automated support promised to extend the life of the system.

Consolidation of Functionality. Consolidating functionality across software systems reduces maintenance costs by decreasing the number of systems to support and eliminates redundancy across domains. Many of the reengineering efforts examined were performed to consolidate software systems within a single organization, as well as consolidate in business practice by selecting a single system to be integrated in multiple organizations where duplicate systems were performing the same function in like domains.

In an effort to minimize redundancy across agencies, DOD has begun initiatives to select specific software systems for use by multiple organizations for performing the same functionality. Traditionally, each agency maintained independent software systems that performed functions in similar domains such as personnel, billing, and contracts. A single software system is being selected for use in multiple organizations that would be supported at one central location. The existing duplicate systems are to be reengineered for the development of a single system for easy transition within various agencies. These systems need to execute on a variety of hardware platforms, including personal computers and mainframes. Support for these systems needs to handle a greater volume of change requests since they will be handled by one central location servicing a larger number of organizations.

Conformance to Standards. New standards in software engineering provide for the development of improved software systems. Reengineering was being performed in many cases to enable existing systems to conform to these new standards as if they had been developed using them. These standards include implementation using the Ada programming language and open systems environments (POSIX). Introducing new standards to the existing system promised to extend the life of these systems and prepare for future migration.

New Software System Development. Many of the reengineering efforts examined were performed to achieve new system requirements. Previous common practice would have developed a brand new software system; reengineering would utilize as much existing software as possible, while improving the performance and enhancing the functionality to meet these new requirements. The inability to develop new software systems due to decreasing budgets was overcome by reverse engineering existing systems to CASE and regenerating more modern software systems that would adhere to modern standards.

Technology analysis. Technology analysis can be facilitated by actually applying interim technological advances to determine the benefits and examine alternative solutions. This analysis is often too costly for many organizations, although it would ease technology transition by providing proof-of-concept. Research organizations often sponsor pilot projects or case studies using mock applications to provide proof-of-concept and to assess the benefits of a maturing technology. Several of the reengineering efforts examined as case studies for developing the Criteria were analyzing the technology of reengineering to determine its potential benefits. Software applications were selected to reengineer which were representative of other projects in the organization; these applications had similar functionality, were written in the same programming languages, and executed on the hardware platforms from which many systems were migrating. The candidate applications were often much smaller than their representative systems, and although this facilitated the reengineering effort, it was unclear as to whether the resulting data would scale up for larger software systems [Hobbs92]. The size of a software system is often detrimental to the success of any software engineering effort by impacting cost, scheduling, and the utilization of resources. Reengineering is no different.

Case Study Data Summary

Case study data showed that most of the software systems had similar characteristics. The existing systems were primarily written in the Common Business Oriented Language (COBOL), with small amounts of hardware-dependent assembly language. There was usually some in-house method for performing data management. The hardware platforms ranged from personal computers to mainframe computers. The sizes overall ranged from ten thousand to over one million lines of source code, although in the cases of the larger systems, it was often only a portion of the software that was to be reengineered.

The target software was the Ada programming language, and the desire to incorporate COTS wherever possible and to integrate with a commercial database management system. The target platform was often a workstation environment, although more often there was a need to support multiple or unknown platforms. This was to be achieved through an open systems environment with POSIX. The target system needed to be adaptable to future modifications with additional or alternative hardware platforms, while reliably performing its required functionality.

Suggested Data to be Collected in Future Efforts

Ideally, an organization that is about to begin a reengineering effort should gather data as the effort progresses. This data should be sent to the authors of this report for further refinement of the Criteria and should be used in their own organizations to predict the success of reengineering efforts on similar software systems.

APPENDIX D

SOFTWARE REENGINEERING SELECTION CRITERIA TABLES

TABLE I. SOFTWARE CHARACTERISTICS

Software Characteristics	Description
Complexity	software constructs (conditionals, iterations, statements), calls to external routines, and I/O
Data structure	variables, constants, complex structures (records, linked lists), user-defined constructs
Languages	number of implementation languages and defined grammars for each
Modularity	defined, unique functionality in each module
Size	SLOC, function points, number of modules/objects, bytes, number of files
Software change rate	average number of modifications, both corrective and perfective, over a given period of time
Software importance	number of people/organizations serving, number of external systems interfacing, times accessed per day/month, life-threatening; function not offered by any other system in parent organization, or other known organization
Structural quality	an analysis of the structure of the software system, i.e, missing code, work-arounds.

TABLE II. SYSTEM CHARACTERISTICS

System Characteristics	Description
Alternate configurations	whether or not there are alternate configurations of this system; if so, a description of these configurations and scenarios for when each is an alternative
Commercial software interface	number of commercial software packages that interface; and a description of that interface
Requirement obsolescence	to what degree the system requirements are no longer viable; which requirements are viable

TABLE III. ENVIRONMENT CHARACTERISTICS

Environment Characteristics	Description
Domain consistency	whether or not the system exists within a domain and the description of that domain; whether or not the system needs to fit within a domain and a description of that domain
Hardware interface	definition of the hardware upon which the system depends and a description of this interface; whether or not the system can reside on more than one hardware/software platform; identification and description of those platforms if currently residing on more than one
Organizational goals	describes the high level goal(s) of the organization relative to this software system and the function this system plays in accomplishing this goal(s).
Usage	number and type of interactions with the system by individuals, other organizations, and external automated systems

TABLE IV. DEVELOPMENT FACTORS

Development Factors	Description
Design	whether or not the design is complete to source code; if not, identification of missing parts or extent of incompleteness; whether or not the design is consistent with the source code; if not, identification of inconsistency
Development methodology	whether or not a well-defined development procedure was used; whether it is documented in text available to the public; whether or not this methodology is supported by automation
Documentation	supplemental documents including reports, tables, user guides which describe aspects of the system not necessarily relating to design
Standardization	use of well-documented development standards, continued throughout maintenance

TABLE V. MAINTENANCE FACTORS

Maintenance Factors	Description
Automation	whether or not an automated process is used for modifications
Configuration management	a management process for controlling and documenting modifications and versions of the software system
Improvement status	to what degree the system could be improved to meet the organizational goals; description of these improvements; to what degree the system could be improved to meet current known requirements efficiently
Life expectancy	amount of time this system is expected to be in existence
Modification effort (M_e)	rate of modification over number of modifications times the average number of maintainers to complete ($M_e = M/n \cdot m$).
Modification rate (M_r)	number of modifications per a given period of time relative to a desired number

TABLE VI. TARGET SOFTWARE CHARACTERISTICS

Target Software Characteristics	Description
Functional improvement	number and type of functional modifications which desired in parallel to the reengineering of this system; test suite for verifying these
Language migration	identification and description of the language implementation requested for this system
Performance improvement	identification and description of the desired performance improvements for this system; test suite for validating these
Technology insertion	identification of specific technological advances that could be implemented to improve the software system, including new commercial packages

TABLE VII. NEW SUPPORT CHARACTERISTICS

New Support Characteristics	Description
Adaptability	to what extent the new support environment will enable the software system to adapt to a changing environment, i.e., enhancements, alternate software/hardware implementations, integration of new components
Automation insertion	desire to maintain the software system using automated techniques
Domain consistency	to what extent the new support environment supports other systems in the same domain
Maintenance improvement	identification and description of specific improvements to the overall maintenance process; to what degree the current maintenance practices are no longer useful; description of the problems

TABLE VIII. NEW ENVIRONMENT CHARACTERISTICS

New Environment Characteristics	Description
Domain insertion	identification and description of the new target domain
New hardware platforms	number and type of hardware interfaces which are to be integrated to this system; identification and description of alternative or new hardware platforms
New Usage	new people, organizations, or automated systems that must now interface with this software system
New organizational goals	new organizational goal(s) that must be addressed by this software system and how

TABLE IX. ORGANIZATION FACTORS

Organization Factors	Description
Budget constraints	budget limitation and predicted allocation of this amount to various resources and phases of the intended reengineering activity(s)
Effort estimations	projected estimations regarding manpower and time for performing the reengineering activity(s)
Management commitment	definition of management expectations and limitations; schedule for identification of management objectives and proof of progress throughout the reengineering activity(s)
Schedule	establish management approved schedule for performing the reengineering activity(s), including alternative plans based on trade-offs measured throughout the activity

TABLE X. METHODOLOGY FACTORS

Methodology Factors	Description
Automation support	automated support for the methodology selected to perform the reengineering
Forward engineering support	to what extent the reengineering tools support the forward engineering processes; identification and description of the interfaces to these processes
Methodology selection	identify and select a well-defined methodology; insure training is available and that the methodology meets the goals of the organization; insure that the methodology is usable across domains and in support throughout the maintenance process
Reuse commitment	investigate reuse options whenever possible

TABLE XI. AUTOMATION FACTORS

Automation Factors	Description
Capability limitations	limitations on use of tool, i.e., maximum number of users, limitations on methodology support
Platform support	insure that hardware/operating systems and integration to existing or selected reengineering platform environment; identification of platforms supporting tools and the limitations of this support
Target hardware support	relationship of tool to hardware platform of reengineered product (does the target hardware itself support the tool, does the tool support the development of software for the target hardware)
Tool availability	does the tool currently exist in the organization, is the organization capable of acquiring this tool, understanding the cost and time for procurement

TABLE XII. PERSONNEL FACTORS

Personnel Factors	Description
Existing system expertise	personnel with the expertise on the functionality, operations, and performance of the existing system and are committed to the reengineering effort
Target system expertise	personnel with the expertise on the desired functionality, operations, and performance of the target software system and are committed to the reengineering effort
Reengineering expertise	personnel with the expertise on the methodology and automated tools for performing the reengineering activity(s) and are committed to the reengineering effort

(This page was intentionally left blank.)

APPENDIX E

SOFTWARE REENGINEERING CRITERIA QUESTIONNAIRE

SOFTWARE REENGINEERING CRITERIA QUESTIONNAIRE

The following questionnaire documents data that is useful in identifying candidate information systems for software reengineering. Please answer the following questions.

I. Existing Software System Criteria *(Existing Software System Criteria is composed of product characteristics that describe the existing software system, the environment in which it operates, and process factors which influenced the development and maintenance of the system.)*

A. Product Characteristics *(Product characteristics describe the software system, including software characteristics and system characteristics, and the characteristics describing the environment in which each system operates.)*

1. Software Characteristics *(Software characteristics describe the source code of the system.)*

- What is the *complexity* of the software? (identify those modules with highest complexity for additional examination)
- What is the *data structure* used in the software?
- What programming *languages* are used to implement the software?
- Is there *modularity* in the software? Do modules perform identifiable functions or display specific behavior?
- What is the *size* of the software? (How many lines of source code/function points?)
- What is the *software change rate*? (The number of modifications made over a given period in time)
- What is the *software importance*? (Number of times each software is accessed in a given period of time; is the software functionality life-threatening; does the software perform some function(s) not performed anywhere else in the organization?)
- What is the *structural quality* of the software? (Is there missing code or "dead" code that is never executed; is there evidence of work-arounds that modify the software)

2. System Characteristics *(System characteristics describe any commercial or external software packages that integrate with source code to meet system requirements.)*

- Are there *alternate configurations* of each software and what are the conditions for which these exist?
- What *commercial software interface* does the system contain? (e.g., DBMS and X-Windows)
- Are existing *requirements obsolete* or need change?

3. Environment Characteristics (*Environment characteristics describe the organization's primary functions with respect to how the software system operates.*)

- Are there *domain consistencies* which must be maintained by each system?
- What is the *hardware interface* for each software system? (Is there a strong relationship between hardware and software? Is the current software *portable* to other hardware platforms?)
- What *organizational goals* do these software systems address and how?
- What is the *usage* of the software? How many users are there for each system? How many individuals, organizations, or automated systems use each system?)

B. Process Factors (*Process factors describe the aspects of the original development process and the maintenance process which have impacted the current implementation of the software system.*)

1. Development Factors (*Development factors define aspects of the development process which have affected the software system implementation.*)

- Does a high level *design* of the software exist? (Is the available design complete with respect to the current software? Is the available design consistent with respect to the current software?)
- What *development methodology* was followed? (Were structured *methodologies* or some type of well-defined and documented methodology utilized?)
- What supporting *documentation* exists?
- What *standardization* principles were used in the development of the software?

2. Maintenance Factors (*Maintenance Factors define aspects of the development process which have affected the software system implementation.*)

- Is the maintenance process supported by *automation*?
- Is any form of *configuration management* employed in maintenance process?
- What is the *improvement status* of the software? (Is the system status regarded as one of improvement or decline? Is the current system defective?)
- What is the *life expectancy* of each system? How long is each system expected to operate?
- What is the *modification effort*? (How many labor hours are spent modifying the system?)
- What is the *modification rate*? (How often is the system modified?)

II. Reengineered Software System Criteria (*Reengineered Software System Criteria include those criteria which describe the desired products of the reengineering process and the factors which influence the reengineering process.*)

A. Reengineering Product Characteristics (*Reengineering Product Characteristics describe the desired characteristics of the new target software, the new support environment and the environment in which the software will be operating.*)

1. Target Software Characteristics (*Target Software Characteristics describe the modifications that are desired between the existing software and the new software.*)

- Will the system be *functionally improved*? (Are new functional requirements being added?)
- Is there a desire to *migrate* to another language?
- Will the *performance* of the system be *improved*? (Will new performance requirements be introduced?)
- Can *technology insertion* improve the software system? Are new technologies available that would greatly improve system implementation?

2. New Support Characteristics (*New support characteristics define the qualities desired in a new maintenance process.*)

- Is the goal to make the system *adaptable*?
- Is there a desire to *automate* the maintenance process?
- Is the goal to make the system *domain consistent*?
- Is there a desire to *improve* the *maintenance* process or change to a new or different automated support environment?

3. New Environment Characteristics (*New Environment Characteristics describe the new operational environment to which the existing system must migrate.*)

- Is there a desire to *insert* systems into a new *domain*?
- Initially, is there a desire to move to another/multiple *new hardware platforms*?
- Are new *users* of the system?
- Is there a new *organizational goal* to which each software system must adhere?

B. Reengineering Process Factors (*Reengineering Process Factors define the influence of the organizational goals, reengineering methodology, and the available resources.*)

1. Organization Factors

- Are there *budget constraints* in which to perform reengineering?
- Have *effort estimations* been made as to the schedule and effort of the reengineering? (including personnel, tools, training, operating, post-implementation costs, reengineering)
- Is *management committed* to renewing these systems? Which ones?
- Are there *schedules* to adhere to?

2. Methodology Factors

- Is there *automation support* for the methodology selected?
- Does the methodology selected link to the *forward engineering* environment?
- Has a *methodology* for reengineering already been *selected*?
- Is there a *reuse commitment* to introduce reusable components into the system wherever possible?

3. Resource Factors

a. Automation Factors

- Are there *capability limitations* in the available tools for reengineering?
- Is there reengineering hardware *platform* available to *support* tools, system data, and the reengineering process?
- Are there reengineering tools that *support* the development of software systems for the *target hardware(s)* of the reengineered system (if applicable)?
- Are these *tools available* in the organization currently or will they have to be procured?

b. Personnel Factors

- Are there personnel available who have *existing system expertise* in the operations, functions, and performance of the software system and are available to work on the reengineering effort?
- Are there personnel available who have *target system expertise* in the operations, functions, and performance of the target software system and are available to work on the reengineering effort?
- Are there personnel who have *reengineering expertise* and are familiar with the selected tools?

(This page was intentionally left blank.)

APPENDIX F

Example Software Reengineering Criteria Application

The following example application of the Software Reengineering Criteria uses data from a typical software engineering environment (SEE) for information systems within DOD. The example begins by documenting the candidate SEE using the Software Reengineering Criteria Questionnaire (Appendix E SOFTWARE REENGINEERING CRITERIA QUESTIONNAIRE), followed by the application of the criteria resulting in a reengineering strategy, and a comparison of the application results to an actual reengineering effort within a similar SEE as proof-of-concept for the Criteria.

5.1 Answering the Software Reengineering Criteria Questionnaire. The Software Reengineering Criteria Questionnaire was answered below using the available data concerning typical information systems agencies within DOD.

I. Existing Software System Criteria

A. Product Characteristics

1. Software Characteristics

- What is the *complexity* of the software? (Identify those modules with highest complexity for additional examination) **average system is estimated between 10 and 20 using Cyclomatic complexity method³, between 5 and 10 for Essential complexity³**
- What is the *data structure* used in the software? **data design based on physical components of the computer system**
- What programming *languages* are used to implement the software? **COBOL-74**
- Is there *modularity* in the software? Do modules perform identifiable functions or display specific behavior? **unknown**
- What is the *size* of the software? (How many lines of source code/function points?) **100-150 778KSLOC per software system across multiple programs**
- What is the *software change rate*? (The number of modifications made over a given period in time) **average of 3 per month per system**
- What is the *software importance*? (Number of times software is accessed in a given period of time; is the software functionality life-threatening; does the software perform some function(s) not performed anywhere else in the organization?) **there are several systems that perform unique functions, loss of one of these system would be very damaging to the organization**
- What is the *structural quality* of the software? (Is there missing code or "dead" code that is never executed; is there evidence of work-arounds that modify the software?) **evidence of patches in many systems, poorly structured in most cases**

2. System Characteristics

- Are there *alternate configurations* of software and what are the conditions for which these exist? **no**
- What *commercial software interface* does the system contain? (e.g., DBMS and X-Windows) **none**
- Are existing *requirements obsolete* or need change? **not at this time**

3. Environment Characteristics

- Are there *domain consistencies* which must be maintained by each system? **no**
- What is the *hardware interface* for each software system? (Is there a strong relationship between hardware and software? Is the current software *portable* to other hardware

platforms?) **Honeywell DPS-8000 computer, software is not portable**

- What *organizational goals* do these software system address and how?
- What is the *usage* of the software? How many users are there for each system? How many individuals, organizations, or automated systems use each system?) **tape interface with several external software systems; tape interface with several devices, including HP 1000-A minicomputer and a NCS Westinghouse Scanner; system is on-line**

B. Process Factors

1. Development Factors

- Does a high level *design* of the software exist? (Is the available design complete with respect to the current software? Is the available design consistent with respect to the current software?) **data requirements for each system existed when it was built, but were never documented.**
- What *development methodology* was followed? (Were structured *methodologies* or some type of well-defined and documented methodology utilized?) **no standard procedure**
- What supporting *documentation* exists? **very little**
- What *standardization* principles were used in the development of software? **unknown**

2. Maintenance Factors

- Is the maintenance process supported by *automation*? **no automation, manual data management facilities**
- Is any form of *configuration management* employed in maintenance process? **unknown**
- What is the *improvement status* of the software? (Is the system status regarded as one of improvement or decline? Is the current system defective?) **maintenance back-log which is increasing for at least one system, ever-increasing user demands mean there are several systems that are becoming outdated, system quality is relatively remaining the same**
- What is the *life expectancy* of each system? How long is each system expected to operate? **indefinite, the functionality of these systems will be continuously required**
- What is the *modification effort*? (How many labor hours are spent modifying the system?) **more than 32 hours per month average, maintainers have expressed general difficulty in making software changes**
- What is the *modification rate*? (How often is the system modified?) **unknown**

II. Reengineered Software System Criteria

A. Product Characteristics

1. Target Software Characteristics

- Will the system be *functionally improved*? (Are new functional requirements being added?) **not at this time**
- Is there a desire to *migrate* to another *language*? **Ada and 4GL**
- Will the *performance* of the system be *improved*? (Will new performance requirements be introduced?) **better response to user requests is needed**
- Can *technology insertion* improve the software system? Are new technologies available that would greatly improve system implementation? **yes, DBMS**

2. New Support Characteristics

- Is the goal to make the system *adaptable*? **yes (future migrations are anticipated for each system)**
- Is there a desire to *automate* the maintenance process? **yes**
- Is the goal to make the system *domain consistent*? **migrate these systems towards**

DISA/CIM information systems domain

- Is there a desire to *improve* the *maintenance* process or change to a new or different automated support environment? **yes, current maintenance organization is declining: there is a need to reduce current maintenance costs, improve data management, and ease modification effort**

3. New Environment Characteristics

- Is there a desire to *insert* systems into a new *domain*? **none at this time**
- Initially, is there a desire to move to another/multiple *new hardware platforms*? **yes, AT&T B32 processor and others, all running Unix**
- Are there new *users* of the system? **none at this time**
- Is there a new *organizational goal* to which each software system must adhere? **prepare for future integration with DISA/CIM information systems, need to better express the data requirements of the business environment**

B. Process Factors

1. Organization Factors

- Are there *budget constraints* in which to perform reengineering? **unknown**
- Have *effort estimations* been made as to the schedule and effort of the reengineering? (including personnel, tools, training, operating, post-implementation costs, reengineering) **unknown**
- Is *management committed* to renewing these systems? **unknown** Which ones?
- Are there *schedules* to adhere to? **none at this time**

2. Methodology Factors

- Is there *automation support* for the methodology selected? **Bachman COBOL Capture and Data Analyst tools**
- Does the methodology selected link to the *forward engineering* environment? **DOD-STD-2167A will be followed**
- Has a *methodology* for reengineering already been *selected*? **reverse engineering and forward engineering**
- Is there a *reuse commitment* to introduce reusable components into the system wherever possible? **unknown**

3. Resource Factors

a. Automation Factors

- Are there *capability limitations* in the available tools for reengineering? **none at this time**
- Is there reengineering hardware *platform* available to *support* tools, system data, and the reengineering process? **Dec 5100/Ultrix, Personal Computers**
- Are there reengineering tools that *support* the development of software systems for the *target hardware(s)* of the reengineered system (if applicable)? **unknown**
- Are these *tools available* in the organization currently or will they have to be procured? **some currently available, but were procured for this effort**

b. Personnel Factors

- Are there personnel available who have *existing system expertise* in the operations, functions, and performance of the software system and are available to work on the reengineering effort? **yes: It is understood that expert domain knowledge is imperative to capturing the domain knowledge and at some of the systems have experts available to form redesign groups**

- Are there personnel available who have *target system expertise* in the operations, functions, and performance of the target software system and are available to work on the reengineering effort? **yes: Redesign groups, contractors, and DISA/CIM**
 - Are there personnel who have *reengineering expertise* and are familiar with the selected tools? **some: training is required for specific tools; contractors and DISA/CIM have reengineering experience**
-

5.2 Suggested Reengineering Strategy Based on Criteria. The suggested reengineering strategy based on the criteria is derived through the following process. Each question in the questionnaire relates to a specific criteria that is highlighted in the text of the question. The definition of each criteria is obtained from Section 4 and examined using the answer to the question. This examination extracts the relevant text from the definition based on the answer to the question. From the relevant text references to the application of the specific reengineering capabilities as defined in Section 2 are extracted and then combined across criteria and finally consolidated to eliminate any redundant text. The reengineering strategy is then presented based on these capabilities. It is important that specific methodologies [for these capabilities] be chosen that are compatible with the requirements of the organization and are supported by automation where possible [Ruhl91, p21]. The reengineering strategy for the example utilizes redocumentation, restructuring, and reverse engineering with forward engineering techniques.

Since several software systems in this SEE are likely to benefit from varying reengineering techniques, it is probable that one system should be chosen that is representative of the others for a first reengineering effort. The system should also exhibit some of the unique qualities of the systems documented in the answers to the questions above as possible. For example, it was stated that some of the software systems interfaced with external software systems and hardware devices. It was also stated that a limited number of systems had personnel available with system expertise. A candidate system should be selected which has at least one interface to an external software system and one to a hardware device. Personnel with the expertise on this system should also be available to confer during the reengineering process.

Redocumentation should be used to provide insight into the current structure of the data, generating tables of variable names, and usage. The *data structure* criteria determined that the software systems in the example SEE contained data design that was based on the physical components of the computer system. Steps should be taken to rename data to more informative terms. Documentation supports reverse engineering by providing supplemental information that is useful in distinguishing design and requirements information from implementation idiosyncracies for improving portability. Redocumenting the software using an automated tool can provide fast information about the software architecture that can be maintained along with the system. The *design* and *documentation* criteria determined that there was little supplemental representations of the software systems.

Restructuring the data model and software architecture may better prepare the software for future modifications. The *structural quality* criteria suggests that these can be improved. However, the longer the system is expected to be in service, the more likely it is that the entire system should probably be reverse engineered. The *life expectancy* criteria states that this system is expected to remain in use for an extended period of time. Older systems that are still relied upon for their functionality will probably continue to serve a useful function in the future and should be moved to a more maintainable, survivable status. During restructuring existing system experts should be used to verify that this process is not altering the functionality of the software system (during restructuring, existing and target system expertise are usually the same individuals since only the performance aspects of the system are altered and functionality remains the same). During restructuring, target software experts insure that performance has been improved. The *existing system expertise* and *target system expertise* criteria suggest that personnel possessing this expertise are available to perform these tasks, thus improving the likelihood of success in reengineering.

Reverse engineering the data model and integrating an efficient data management facility will provide the biggest payoff for this information system. Reverse engineering can salvage an existing software design, which can then be restructured and reimplemented into a more modern software system that utilizes advanced software engineering technology. The lack of current designs as suggested by the *design* criteria supports the generation of such representations. Reverse engineering the data models from the software and incorporating a commercial database may improve data management as suggested by the *technology insertion* criteria. Improved naming conventions can be achieved when reimplementing to a new programming language that does not limit the designation of variables by length or characters, and permits extended structures. Migration to advanced hardware platforms also can affect data naming and structure conventions, often enabling greater memory capacity. Poorly structured software is candidate for reverse engineering to a design representation that can then be restructured. Time may be wasted on analyzing and reverse engineering dead code that is of no value to the software. Steps should be taken to manually parse through the code to eliminate unnecessary code. During forward engineering target software experts are consulted to insure the new system meets all requirements, including functional and performance.

Reverse engineering the software into a CASE tool will help integrate automation into the maintenance process. Modernizing the system promises to improve system performance as well as the response time of maintainers to change requests from the users. Reverse engineering and regeneration of the software may be useful in generating a new software system for the new hardware. Reengineering is often used to modernize the software to a implementation that is better suited for future migrations [MITRE92]. Eliminating hardware dependencies when using the recovered design in a new implementation may make the software more portable for future migrations to new or alternative hardware platforms. Reusable components may be incorporated into the existing system to replace outdated components and improve the performance of the software. Reverse engineering the software into a CASE tool may facilitate the generation of implementations in alternative languages.

Reverse engineering to a high level design and then regenerating the software in Ada is another option. During reverse engineering and redocumentation, these experts should verify that the recovered design and support documentation accurately portray the software system. They should also be consulted to clarify complex or confusing aspects of the software system throughout the reengineering process.

Automated tools, if available, may be useful in supporting this reengineering strategy. The tool performance will be impacted by several issues, including size and complexity. In general, reengineering processes that are easy to automate will be more successful on smaller software modules. More manual efforts will be required on larger modules. Poor structure may cause automated tools difficulty in analyzing the source code. It may be advantageous for a team of programmers to go through the code manually to identify such areas in the code and perhaps manually discard or modify them. Considering the focus of most CASE tools for a particular computing environment, one set of CASE tools should not be depended on for uniform applicability to all software systems across this organization [Ruhl91, p21]. In addition, since several tools may be used during reengineering, it is important to consider not only the impact of these tools on the existing software engineering environment in the organization, but whether or not there is data interchange capability between these tools. Good business sense should be applied when considering the purchase of expensive tools. The personnel with the expertise on the methodologies and automated tools for performing the reengineering activity(s) should work with the appropriate experts to match the technical approach for reengineering with the characteristics of the software system and the desired target system. If these experts do not readily exist, the reengineering strategy must accommodate appropriate training for the methods and tools associated with the reengineering strategy selected.

5.3 Comparison of Strategy to Actual Reengineering Effort. The strategy proposed by the Software Reengineering Criteria was similar to one performed in an actual reengineering effort [MITR92]. This effort involved a software system similar to the one suggested for a first time reengineering effort in the example organization above. The software system was typical of those in the SEE according to the *size*, *languages*, and *usage* criteria. This system was 120K source lines of COBOL and included tape interfaces.

The actual effort primarily consisted of reverse engineering the logical data model from the COBOL source code and available documentation. The forward engineering process has yet to be completed, but consists of generating a new logical data model, physical data model, and new database design.

The Bachman COBOL Capture and Data Analyst tools were used to capture the system's data requirements. The *data structure* and *documentation* criteria suggest that redocumentation be used to understand the current structure of the data and using an automated tool would speed the process as suggested by the Methodology factors. Reverse engineering was used to capture the data model and the Bachman toolset automatically

provided some of the suggested information on the data requirements during this process. The *technology insertion* criteria did suggest that reverse engineering the data model and integrating an efficient data management facility would provide the biggest payoff for this information system and this was the approach taken in the actual project.

The key issues with respect to the existing system were data management and software characteristics. Data management was not very efficient, utilizing manual means. Improving the data management process was key to improving the system, since a large percentage of the existing system functions centered on data management (70-80%). Data names were not very informative, and better naming conventions were desired in the target system. The COBOL source code had been modified so often that it was poorly structured as suggested by the *structural quality* criteria.

Technology support for reengineering to efficiently support data management and maintenance was determined immature during this reengineering effort. Reverse engineering of COBOL and translation tools from COBOL to Ada were also viewed as limited. There were delays in installing software due to incomplete documentation and the limitations of the tools which had to be compensated for manually. These limitations are addressed by the *capability limitations* criteria which was unknown prior to the reengineering effort.

The *new hardware platforms* criteria determined that the decision had been made to move to an advanced software and hardware platforms in anticipation that this would also improve the system. The hardware platform would also improve data management and support the integration of a commercial DBMS. The target system selected was a Unix-based AT&T 3B2 computer and the implementation languages of Ada and a fourth generation language (4GL), integrated with the Oracle relational database. These implementation decisions also supported the *new organizational goal* criteria.

The Criteria recommended that automated tools should be considered, and that appropriate training for the methods and tools must be incorporated into the schedule. If these tools did not exist then adequate compensation for their limitations should be made by experienced personnel. The actual effort found that available technology was immature and processes had to be performed manually. The *schedule and effort estimations* criteria refer to the issues which must be considered when establishing milestones for a reengineering project, which include training and tool installation. This criteria warns that delays should be expected in utilizing automated tools with respect to training and that adequate hardware platforms are necessary to support these products. It was not clear in this reengineering effort as to whether reasonable estimates and schedules were made, however these issues resulted in delays and execution problems for the actual effort.

5.4 How Results Are Derived. The following explains how the answers to the Software Reengineering Criteria Questionnaire in Section 5.1 determined the reengineering strategy presented in Section 5.2. Each question in the questionnaire relates to a specific criteria that

is highlighted in the text of the question. The definition of each criteria is obtained from Section 4 and examined. This examination extracts the relevant text from the definition based on the answer to the question. From the relevant text references to the application of the specific reengineering capabilities as defined in Section 2 are extracted, then combined with other criteria and finally consolidated to eliminate any redundant text. This text was then consolidated by combining text which applied to like reengineering capabilities. For example, all of the criteria which implied reverse engineering strategies was combined to form the reverse engineering strategy presented in Section 5.2. Each question and answer is listed below followed by the appropriate text for each criteria from Section 4.

What is the *complexity* of the software? (identify those modules with highest complexity for additional examination) *average system is estimated between 10 and 20 using Cyclomatic complexity method³, between 5 and 10 for Essential complexity³*

The complexity criteria states that "Restructuring the existing software may lessen the complexity without altering the functionality of the software. Emphasis should be placed on eliminating goto statements and code that is never executed. The software can also be reverse engineered to a high level design which is then restructured and used to generate a more concise and efficient implementation of the software. The reverse engineering process must be verified to insure that the recovered design accurately captures the existing software. This process is preferred when there are additional implementation changes desired, such as converting to the Ada programming language."

What is the *data structure* used in the software? *data design based on physical components of the computer system*

The criteria states that "Data design is the key structural component in most information systems, requiring extensive database management. The largest percent of functionality in most information systems is performed within the context of data management. The data architecture is often the driving force behind the software control flow architecture. For this reason, emphasis should be placed on the data design because it will force modifications to the process design [Ruhl91, p17]. Redocumentation may provide insight into the current structure of the data, generating tables of variable names, and identifying within the software where the data is used and modified. Steps should be taken to rename data to more informative terms as permitted in the current software implementation [MITR92] and to adhere to DOD standards concerning data naming conventions. Reverse engineering the data model and integrating an efficient data management facility will provide the biggest payoff for most information systems [MITR92]. Improved naming conventions may only be achieved by reimplementing in a new programming language, such as a language that does not limit variable name length or characters that can be used. Migration to advanced hardware platforms also can improve data structure and naming conventions through increased memory capacity."

What programming *languages* are used to implement the software? **COBOL-74**

The criteria states that "Translation is usually most successful in efforts where the goal is to solely generate a new version of the system in another language that is similar to the current language or a different version of the same language. This effort is also under time constraints and is minimally funded. This strategy is most successful with small programs where the software architecture and data structure will remain the same. Reverse engineering to a language-independent design is an alternative to translation which may be more time-consuming and expensive. This design can then be used to generate a more efficient representation of the software which is more efficient, taking advantage of the new language constructs. Translation does not incorporate alternative implementations which use the unique features of the target language. This must be accomplished through manual conversion or restructuring techniques."

What is the *size* of the software? (How many lines of source code/function points?) **100-150 KSLOC per software system across multiple programs**

The criteria states that "Cost and schedule will also be impacted by the size. In general, reengineering processes that are easy to automate will be more successful on smaller software modules. More manual efforts will be required on larger modules."

What is the *software change rate*? (The number of modifications made over a given period in time) **average of 3 per month per system**

The criteria states that "Assessments should be made to determine what the needs of the user are relative to the functionality of the software. An ever-changing software system should be quickly migrated to a more efficient support environment to better adapt to the needs of the user. Unnecessary functionality should be eliminated, and desired functionality improved. Reverse engineering to a CASE environment can provide automated support and is also useful for generating a more modern system."

What is the *software importance*? (Number of times the software is accessed in a given period of time; is the software functionality life-threatening; does the software perform some function(s) not performed anywhere else in the organization?) **there are several systems that perform unique functions, loss of one of these system would be very damaging to the organization**

The criteria states that "Software importance is determined by the number of people or organizations which utilize the software system. Users of the system also include external automated systems with which the candidate system interfaces. Examples of highly critical software systems includes those that perform functions in no other software system, those

which could not easily be replaced with another system, and those which would have a detrimental effect on the organization if they were to be eliminated. A payroll system could be considered a highly critical system, since its elimination would have an enormous impact on the organization should the employees experience a delay in their pay. If the system performs life-threatening functions or unique functions which no other system performs, then the system is also highly critical. Information dependencies between the candidate system and external systems, as well as other systems within a like domain should be identified [Ruhl91, p15]. The organization should make a determination as to why the system is important and consider redevelopment or reverse engineering, while the system remains in use. This system should probably be an example system to analyze from a functional viewpoint as a basis for determining new technology that will improve the overall current business practices of the organization [Ruhl91, p15]. The cost of reengineering these systems should be weighed heavily against the importance of its functions and ample support should be given to improve and secure this type of software system for extended use. Highly critical systems should be further examined for consolidation to minimize the number of overall systems which the organization must support."

What is the *structural quality* of the software? (Is there missing code or "dead" code that is never executed; is there evidence of work-arounds that modify the software?) **evidence of patches, poorly structured in most cases**

The criteria states that "Structural quality describes the structure or architecture¹ of the software system. The software architecture may be well-structured and suitable for quickly converting to a modern programming language or hardware platform. In this case, translation can be an effective means for performing this conversion. Poorly structured software is a candidate for code restructuring or for reverse engineering to produce a design representation that can then be restructured [MITR92]. Poor structure may cause automated tools difficulty in analyzing the source code. Time may be wasted on analyzing and reverse engineering dead code that is of no value to the system requirements. It may be advantageous for a team of programmers examine the code manually to identify such areas in the code and perhaps manually discard or modify them. Redocumentation techniques can often report on the structural quality of source code. Automated redocumentation exists that generates structure charts² which define the procedures used to implement the software, including the calling hierarchy, naming conventions, and input and output of data between these modules. This is useful in quickly identifying modules which are never executed, are not represented in an existing high level design, or utilize global data [STSC92]."

¹Architecture includes the implementation design of the software. The term Design is used to define a criteria under Development Factors that refers to the existence of a high-level representation of the software.

²Structure charts depict the structure of the software as defined by Edward Yourdon and Larry Constantine in Structured Design, Englewood Cliffs, NJ: Prentice Hall, 1979.

Are there *alternate configurations* of the software and what are the conditions for which these exist? **no**

The criteria states that "Alternate configurations identifies whether or not there are alternate configurations of the software system. The requirements justifying these versions should be identified and it should be determined whether the alternate configurations can be consolidated into a single software system. Restructuring the system may improve modularity for identifying specific functions and enable a system to incorporate functions previously performed in other systems. Reverse engineering permits the consolidation of these functions into a uniform design representation that can then be used to generate the new software system."

What *commercial software interface* does the system contain? (e.g., DBMS and X-Windows)
none

The criteria states that "Commercial software interface describes the number of commercial software packages that interface the software system. It is important to identify all of these interfaces and understand the structure of this interface to insure that any reimplementation or modification to the existing software maintains this intersection. Poorly integrated system components is a reason to consider reengineering [MITR92]. Most modernization efforts integrate commercially available software components, including database management systems (DBMS) and other software packages to meet system requirements. The key issue in most of these integration efforts is the data interchange format. Commercial packages do not easily interact with older programming languages of hardware platforms, thus requiring conversion to new software and hardware implementations. Reverse engineering the software to a design representation provides a means for achieving this type of conversion. The design can be restructured to better integrate commercially available system components and reimplemented in modern programming languages for state of the art hardware."

Are existing *requirements obsolete* or need change? **not at this time**

The criteria states that "Requirement obsolescence describes to what degree the system requirements are no longer viable. Current system functionality may not conform to user needs. Obsolete requirements should be eliminated from the existing system during reengineering. Source code which performs these requirements should be extracted from the software prior to restructuring if possible, else afterwards. Reverse engineering to the design level may more easily identify the portions of the software performing these outdated requirements and can easily be removed from the design prior to reimplementation."

Are there *domain consistencies* which must be maintained by this system? **no**

The criteria states that "Domain consistency describes whether or not the system exists within a domain and the description of that domain. Modifications to the system must conform within the constraints of this domain. Often reverse engineering is performed to migrate a software system to a domain that provides consistency across similar applications in an organization. Reimplementation of the new system should consider reuse options for achieving domain consistency as well as consolidation of functionality across multiple software systems."

What is the *hardware interface* for this software system? (Is there a strong relationship between hardware and software? Is the current software *portable* to other hardware platforms?) **Honeywell DPS-8000 computer, software is not portable**

The criteria states that "Hardware interface describes the current hardware platform upon which the software system executes and a description of the dependencies of the software on this platform. The hardware capabilities often restrict software implementation options and these may change given an alternative platform. Antiquated hardware is often the reason for modernizing the software system [MITRE92]. Fewer dependencies may ease migration to a new hardware configuration, while strong ties may impact the technical approach taken to modernize the software. Reverse engineering is useful for generating a hardware-independent design of the software and determining implementation dependencies which resulted from that platform. The hardware-independent design can then be implemented for alternate hardware configurations."

What is the *usage* of this software? How many users are there for this system? How many individuals, organizations, or automated systems use this system?) **tape interface with several external software systems; tape interface with several devices, including HP 1000-A minicomputer and a NCS Westinghouse Scanner; system is on-line**

The criteria states that "Usage describes the amount of use the software system undergoes, including the number and type of interactions with the system by individuals, external organizations and other automated systems. The needs of these entities must be maintained when modernizing the system. In many cases it is precisely these needs that are driving the reengineering effort. Modernizing the system promises to improve response times by the system as well as by maintainers responding to change requests from the users. Reverse engineering to an automated support environment helps improve the modification process. Redocumentation provides supporting documentation to maintainers making decisions about modifications. Restructuring the software may enable the system to more readily accept modifications without negatively impacting other parts of the system."

Does a high level *design* of this software exist? (Is the available design complete with respect to the current software? Is the available design consistent with respect to the current software?) **data requirements for each system existed when it was built, but were never documented.**

The criteria states that "Design is a comprehensive high level representation of the software that is consistent and complete to the current implementation. If this representation exists, it may be used or restructured to reimplement the software in a standard development process. If the existing design is not consistent or complete, it may be used as supplemental documentation during a reverse engineering process which will produce a more comprehensive and current representation. Reverse engineering is used to generate designs for software systems that do not have an available design that is complete or consistent with the current implementation of the software system that is useable in a effective maintenance process. This may include designs that were not originally developed using structured modeling techniques or are not supported by an automated process (automation provides necessary efficient maintenance support) [McCa92]. The design may serve as an end product that provides system understanding or may serve as a means for implementing a new system with added functionality or new programming language. Reverse engineering the data model for information systems often lends insight into how to restructure the data for an improved implementation."

What *development methodology* was followed? (Were structured *methodologies* or some type of well-defined and documented methodology utilized?) **no standard procedure**

The criteria states that "Development methodology identifies whether or not a well-defined development procedure was used in developing this software. If a documented methodology was used, then it may be easier to understand the impact of this methodology on the implementation of the software and the design of the software, if it exists. It is also important to note if automation was used in developing the software system. If the previous development procedure is still a viable development methodology, then consideration should be given for utilizing this method in regenerating the software system."

What supporting *documentation* exists? **very little**

The criteria states that "Documentation describes any reports, tables, or design record that exists for the software system. Documentation is credited with being the key to adequate software maintenance, and yet it is still the most unqualified segment of the software engineering process. As the languages with which software systems are implemented move to lower abstraction, the importance of creating adequate documentation becomes more critical.

There are useful suggestions that can be performed immediately to improve the documentation status of a software system [Hova92]. Obsolete documents, such as those that

have not been used in more than a year, should be discarded since they are either outdated or unusable. Perform a documentation audit to determine what documents are needed and set about generating these documents or making additions to existing incomplete documents. This can be performed by examining trouble reports that were diagnosed as help requests, or the user help requests, if these are logged. Finally, new development, as well, as reengineering should establish documentation as a top priority in these processes. A lack of adequate documentation is the most common reason for considering reengineering [Ruhl91]. Redocumentation techniques produce various types of documents to supplement the software system implementations (specification, design, source code). Reports defining the variables, procedure calls, and procedure interfaces can be useful in gaining a better understanding of the software. Documentation supports reverse engineering by providing supplemental information that is useful in distinguishing design and requirements information from implementation idiosyncracies."

Is the maintenance process supported by *automation*? no automation, manual data management facilities

The criteria states that "Automation factor identifies whether or not an automated process is used for supporting the existing system configuration. Incorporating automation is usually a high level goal of most organization since it can increase the efficiency of many processes, including development and maintenance [Room92]. Automated tools can assist the maintenance process, including language sensitive editors for performing source code modifications and automated configuration management to provide version control and document the changes made to the software. Preparation for maintenance can begin in the development processing Computer-aided software engineering (CASE) tools which can be used to design and develop software, and then maintain it. Redocumentation is the most mature automation capability. Redocumenting the software using an automated tool can provide fast information about the source code architecture. Reverse engineering the software into a CASE tool can integrate automation into the maintenance process."

What is the *improvement status* of the software? (Is the system status regarded as one of improvement or decline? Is the current system defective?) maintenance back-log which is increasing for at least one system, ever-increasing user demands mean there are several systems that are becoming outdated, system quality is relatively remaining the same

The criteria states that "Improvement status identifies to what extent the system is operating according to the current system requirements. There are several methods for measuring faults in source code, including Gaffney³. If there are current defects in the software or

³Gaffney, John E., "Estimating the Number of Faults in Code," IEEE Transactions on Software Engineering, vol SE-10, no. 4, July 1984, pp. 459-465.

modifications that have been requested which have not been implemented, either due to modification effort or cost, this may be a good time to consider reengineering options. An analysis of the system may identify isolated parts of the software which are not performing adequately. These parts may be reverse engineered, modified at the design level, and reimplemented. The corrected parts can then be integrated back into the remaining software. Most existing source code components are so tightly interweaved, that it is usually necessary to reverse engineer the entire software system and generate a more modular software system.

This criteria also identifies to what degree the system could be improved to meet current known requirements efficiently. If the system has many modifications that are necessary in order for it to reach an acceptable level of performance, then consideration should be given for whether this software should continue as a functioning element in the organization. It should be clearly understood what changes are necessary and incorporate those into a reengineering process. Restructuring can be applied to improve the performance of the software without introducing new functionality. Reverse engineering can be used to abstract a high level design which could then be used to incorporate new functionality that is implemented in a new software system."

What is the *life expectancy* of this system? How long is this system expected to operate? indefinite, the functionality of these systems will be continuously required

The criteria states that "Life expectancy identifies the number of years this software system is expected to remain in operation. The system may not be operating much longer because the functions are becoming outdated or because the software system has been designated for replacement. If the software is not expected to be in use much longer due to functional obsolescence, only improvements to the current maintenance process should be considered. If the software has been designated for replacement, certain reengineering capabilities are feasible. Only available resources should be applied, including automated tools which currently exist in the organization. Minimal redocumentation and restructuring should be performed to provide quick benefits for improving maintenance and extending the life of the system to insure it remains operational until replacement can be made. Advanced processes within these capabilities and reverse engineering should be considered only if the option of replacing the system can be upheld by revitalizing the existing one through reengineering."

What is the *modification effort*? (How many labor hours are spent modifying the system?) more than 32 hours per month average, maintainers have expressed general difficulty in making software changes

The criteria states that "Modification effort identifies the rate of modification over number of modifications times the average number of maintainers to complete. If the existing software is very difficult to maintain, this may be a sign that it needs to be reengineered. The software maintainers should be interviewed to determine if the difficulties are due to a lack of

understanding, complexity of the software, result in "rippling effect" that leads to more problems.

A lack of software understanding may be minimized through improved supporting documentation. Redocumenting the software may identify interrelationships between the software components that are impacted during maintenance procedures. This can be useful when modifying software for predicting effects of change on other parts of the software. Restructuring the software may reduce its complexity, and eliminate unnecessary code. Both of these techniques may reduce the chances that one modification to the software may have a disastrous effect on other parts of the software. Severe problems may only be solved through reverse engineering and a complete regeneration of the software system."

Is there a desire to *migrate* to another *language*? **Ada and 4GL**

The criteria states that "Language migration describes the identification and description of the language implementation requested for this system. There are several reasons for reimplementing the software in a different programming language, including implementation improvements offered by the language or the desire to implement all software in a common language within an organization to ease maintenance. Within DOD, the standard programming language selected is Ada. Most modernization efforts within DOD are required to use Ada. Reengineering the existing software to Ada is one way to adhere to this mandate. If the current language implementation is similar to Ada, translation can be performed to the new implementation and if necessary manual code generation for the software which does not easily convert. Reverse engineering to a high level design and then regenerating the software in Ada is another option. Often the existing system implementation is used as the baseline, a design, or supporting documentation which is referenced when generating a new software system in the target language manually. This may prove to be the most time-consuming effort."

Will the *performance* of the system be *improved*? (Will new performance requirements be introduced?) **better response to user requests is needed**

The criteria states that "Performance improvement describes the identification and description of the desired performance improvements for this system. These improvements may be achievable in the existing software implementation, and this should be considered. Although, more often these performance requirements require improved hardware or new software implementation. Improving the performance of the software may be an opportunity to reengineer to a new system implementation. Translation to a new software language for execution on a new hardware platform is one of the fastest migration techniques, however the translated software may not utilize the capabilities of the target language or the new hardware. Reverse engineering to a high level design which can then be targeted for a new hardware suite is one way to achieve these performance improvements."

Can *technology insertion* improve the software system? Are new technologies available that would greatly improve system implementation? yes, DBMS

The criteria states that "Technology insertion is the identification of specific technological advances that could improve the implementation of the software. This criteria does not include automated support for the development or maintenance of software, which is defined in New Support Characteristics. The awareness of new technology that may support the existing functionality of the current software system or new requirements that are to be incorporated into the existing system, may be performed through reengineering [MITRE92]. Reusable components may be incorporated into the existing system to replace outdated components and improve the performance of the software. Often improved data management facilities may be incorporated into an information system which previously performed its data management in an ad-hoc or hard-coded manner. Reverse engineering the data models from the software and incorporating a commercial database may improve data management. New programming languages may be used to implement a software system that contains improved library routines, functions and data structures that better support the system requirements. Reverse engineering can salvage an existing software design, which can then be restructured and reimplemented into a more modern software system that utilizes advanced software engineering technology."

Is the goal to make the system *adaptable*? yes (future migrations are anticipated for each system)

The criteria states that "Adaptability describes to what extent the new support environment will enable the software system to adapt to a changing environment, i.e., enhancements, alternate software/hardware implementations, integration of new components. These characteristics will prepare the software system for future migrations. Reengineering is often used to modernize the software to a implementation that is better suited for future migrations [MITRE92]. Restructuring the data model and software architecture may better prepare the software for future modifications. Reverse engineering the software into a CASE tool may facilitate the generation of implementations in alternative languages. Eliminating hardware dependencies when using the recovered design in a new implementation may make the software more portable for future migrations to new or alternative hardware platforms."

Is there a desire to *automate* the maintenance process? yes

The criteria states that "Automation insertion describes the automated techniques that are needed to support the new support environment. This criteria does not include implementation techniques supplied by technological advances software development, which was addressed in Target Software Characteristics. The desire to automate the maintenance environment is often the reason for modernizing a software system [MITRE92]. CASE tools

which are used to develop software systems are useful throughout the life of the software by supporting modifications to the source code while updating designs and documentation.

Steps can be taken to integrate CASE into the existing software engineering environment within an organization [CIM92]. This is most often successful in more modern software systems and mature organizations. Many programming languages and older hardware platforms do not support modern CASE tools, thus forcing the modernization of the software system itself. Reverse engineering provides a means for extracting information and populating CASE repositories with data that can be used to support the current implementation. CASE often includes code generation support for reimplementing the software into a more modern programming language."

Is the goal to make the system *domain consistent*? migrate these systems towards DISA/CIM information systems domain

The criteria states that "Domain consistency describes to what extent the new support environment supports other systems in the same domain. Systems that reside in a identifiable domain should be considered for reengineering for achieving commonality through software reuse and consolidation. Redocumentation techniques can be used to identify commonality between software systems. Software system should be consolidated where possible. Reverse engineering systems within a single domain can enable the generation of a consolidated system from separate software programs implemented in varying languages and executing on differing hardware platforms. Reuse modules should be used to implement common functionality when possible."

Is there a desire to *improve* the *maintenance* process or change to a new or different automated support environment? yes, current maintenance organization is declining: there is a need to reduce current maintenance costs, improve data management, and ease modification effort

The criteria states that "Maintainability describes to what extent the new support environment provides improvements to the overall maintenance process. This identifies specific technological advances that are a part of the new support environment that improve the process of supporting the system software and a description of how this is accomplished. The integration of CASE tools promises to enable faster modifications to the software and automatic updates to supporting documentation. Reverse engineering is used to produce a high level design representation that can be stored and automatically supported within a CASE tool. The ability to make changes and generate updates to this representation and subsequently the implementation, should improve response time to change requests."

Are there *domain consistencies* which must be maintained by this system? **none at this time**

The criteria states that "Domain integration is the identification and description of the new target domain. The domain may be the programming language, hardware platform, or functional. Domains from which the current system is far removed may be more difficult to reach through reengineering and may require a completed redevelopment. A well-structured and modular software system may incorporate replacement parts that are reuse software modules that immediately migrate the existing system to a desired domain."

Initially, is there a desire to move to another/multiple *new hardware platforms*? **yes, AT&T B32 processor and others, all running Unix**

The criteria states that "New hardware platforms is the identification and description of alternative or new hardware platforms. An understanding of the new hardware platform and how the current software interfaces with is crucial to the success of the system migration. This describes the number and type of hardware interfaces which are to be integrated to this system. The new hardware platform often drives the modification of the software to a new or alternate version. It may be necessary to restructure or update the version of the current implementation language to execute the software on the new hardware. Restructuring may be necessary, and replacement of new library modules to accomplish the same functionality of the existing software. This is also a time to consider an alternate implementation of the software in another programming language that is more compatible with the hardware and operating system. Reverse engineering and regeneration of the software may be useful in generating a new software system for the new hardware."

Is there a new *organizational goal* to which this software system must adhere? **prepare for future integration with DISA/CIM information systems, need to better express the data requirements of the business environment**

The criteria states that "New organizational goals describes any new organizational goal(s) that must be addressed by this software system. Room presented high level goals of an organization which eliminate non-essential products and processes, increase the value of those remaining; and increase the efficiency of those processes through streamlining, simplification and/or automation [Room92]. The organization may consider the candidate software system as non-essential and in this case the best option is to eliminate this software system. More likely, this software offers some value and should be examined for potential consolidation with other similar systems for consolidation purposes that will streamline and simplify maintenance. Automation may also be incorporated as discussed in paragraph 5.2.1.2 New Support Characteristics and 5.2.2.2 Methodology Factors criteria below."

Is there *automation support* for the methodology selected? Bachman COBOL Capture and Data Analyst tools

The criteria states that "Automated support identifies the automated tools which support the reengineering methodology. There are several tools available which support various reengineering activities [STSC92]. It is important to be realistic in what these tools provide. Investigate their capabilities and be prepared to address inefficiencies through adequate personnel who are knowledgeable of these tools and can perform processes when the tools do not. Considering the focus of most CASE tools for a particular computing environment, one set of CASE tools should not be depended on for uniform applicability to all needs across an organization [Ruhl91, p21]. In addition, since several tools may be used during reengineering, it is important to consider not only the impact of these tools on the existing software engineering environment in the organization, but whether or not there is data interchange capability. Good business sense should be applied when considering the purchase of expensive tools."

Does the methodology selected link to the *forward engineering* environment? DOD-STD-2167A will be followed

The criteria states that "Forward engineering support identifies to what extent the reengineering tools support the forward engineering processes; identification and description of the interfaces to these processes."

Has a *methodology* for reengineering this system already been *selected*? reverse engineering and forward engineering

The criteria states that "Methodology selected identifies a well-defined methodology; insures training is available and that the methodology meets the goals of the organization. A methodology should be chosen that is compatible to the requirements of the organization and are supported by automation where possible [Ruhl91, p21]. It is cost-effective if this methodology is usable across domains and transitions the system to a better maintenance process."

Is there reengineering hardware *platform* available to *support* tools, system data, and the reengineering process? Dec 5100/Ultrix, Personal Computers

The criteria states that "Platform support identifies the hardware/operating systems and integration to existing or selected reengineering platform environment; the platforms supporting tools and the limitations of this support."

Are these *tools available* in the organization currently or will they have to be procured? **some currently available, but were procured for this effort**

The criteria states that "Tool availability identifies whether the tool currently exists in the organization, is the organization capable of acquiring this tool, understanding the cost and time for procurement."

Are there personnel available who have *existing system expertise* in the operations, functions, and performance of the software system and are available to work on the reengineering effort? **yes: it is understood that expert domain knowledge is imperative to capturing the domain knowledge and at some of the systems have experts available to form redesign groups**

The criteria states that "Existing system expertise identifies the personnel with the expertise on the existing system. During reverse engineering and redocumentation, these experts should verify that the recovered design and support documentation accurately portray the software system. They should also be consulted to clarify complex or confusing aspects of the software system throughout the reengineering process. During restructuring these individuals should be used to verify that this process is not altering the functionality of the software system (during restructuring, existing and target system expertise are usually the same individuals since only the performance aspects of the system are altered and functionality remains the same)."

Are there personnel available who have *target system expertise* in the operations, functions, and performance of the target software system and are available to work on the reengineering effort? **yes: Redesign groups, contractors, and DISA/CIM**

The criteria states that "Target system expertise identifies personnel who possess the expertise on the desired target system. These individuals should verify new requirements for achieving the target system and should be consulted to clarify any confusing aspect of the proposed target system. During forward engineering these individuals should be consulted to insure the new system meets all requirements, including functional and performance. During restructuring, these individuals insure that performance has been improved."

Are there personnel who have *reengineering expertise* and are familiar with the selected tools? **some: training is required for specific tools; contractors and DISA/CIM have reengineering experience**

The criteria states that "Reengineering expertise identifies whether the personnel exist with the expertise on the methodologies and automated tools for performing the reengineering activity(s). These individuals should work with the appropriate experts to match the technical

approach for reengineering with the characteristics of the software system and the desired target system. If these experts are not available, the reengineering strategy must supply appropriate training for the methods and tools associated with the reengineering strategy. "