

DTIC FILE COPY

(2)

AD-A226 613

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DTIC
ELECTE
SEP 24 1990
S D CS D



THESIS

THE CLASSIFICATION AND EVALUATION
OF
COMPUTER-AIDED SOFTWARE ENGINEERING
TOOLS

by

GARY WAYNE MANLEY

September, 1990

Thesis Advisor:
Co-Advisor:

Luqi
Bernd J. Krämer

Approved for public release; distribution is unlimited.

90 09 20 026

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				
1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b RESTRICTIVE MARKINGS	
2a SECURITY CLASSIFICATION AUTHORITY			3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.	
2b DECLASSIFICATION/DOWNGRADING SCHEDULE				
4 PERFORMING ORGANIZATION REPORT NUMBER(S)			5 MONITORING ORGANIZATION REPORT NUMBER(S)	
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School		6b OFFICE SYMBOL (If applicable) 37		7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000			7b ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	
8a NAME OF FUNDING/SPONSORING ORGANIZATION National Science Foundation		8b OFFICE SYMBOL (If applicable)		9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER NSF CCR-8710737
8c ADDRESS (City, State, and ZIP Code) Washington, D.C. 20550			10 SOURCE OF FUNDING NUMBERS Program Element No Project No Task No Work Unit Accession Number	
11 TITLE (Include Security Classification) The Classification and Evaluation of Computer-Aided Software Engineering Tools				
12 PERSONAL AUTHOR(S) Manley, Gary Wayne				
13a TYPE OF REPORT Master's Thesis		13b TIME COVERED From To		14 DATE OF REPORT (year, month, day) 1990, September
15 PAGE COUNT 197				
16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
17 COSATI CODES FIELD GROUP SUBGROUP			18 SUBJECT TERMS (continue on reverse if necessary and identify by block number) Computer-Aided Software Engineering; Systems Development Lifecycle; DoD STD-2167A; CASE Environment; Framework; Repository; Tool Taxonomy; Tool Evaluation Process	
19 ABSTRACT (continue on reverse if necessary and identify by block number) The use of Computer-Aided Software Engineering (CASE) tools has been viewed as a remedy for the software development crisis by achieving improved productivity and system quality via the automation of all or part of the software engineering process. The proliferation and tremendous variety of tools available have stretched the understanding of experienced practitioners and has had a profound impact on the software engineering process itself. To understand what a tool does and compare it to similar tools is a formidable task given the existing diversity of functionality. This thesis investigates what tools are available, proposes a general classification scheme to assist those investigating tools to decide where a tool falls within the software engineering process and identifies a tool's capabilities and limitations. This thesis also provides guidance for the evaluation of a tool and evaluates three commercially available tools.				
20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/DUPLICATE ☐ SAME AS REPORT ☐ OTHER USERS			21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a NAME OF RESPONSIBLE INDIVIDUAL Professor Luqi			22b TELEPHONE (Include Area code) (408) 646-2735	
			22c OFFICE SYMBOL Code 52Lq	

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsoleteSECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Approved for public release; distribution is unlimited.

The Classification and Evaluation of
Computer-Aided Software Engineering Tools

by

Gary W. Manley
Captain, United States Marine Corps
B.B.A., Texas A&M University, 1981

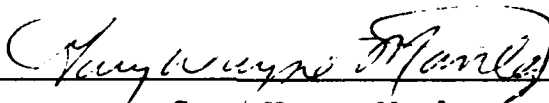
Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

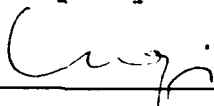
NAVAL POSTGRADUATE SCHOOL
September 1990

Author:

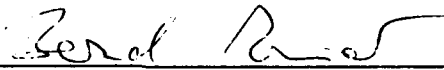


Gary Wayne Manley

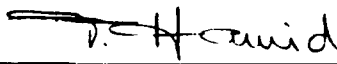
Approved by:



Luigi, Thesis Advisor



Bernd J. Krämer, Co-Advisor



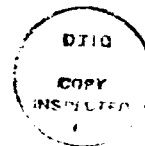
Tarek Abdel-Hamid, Second Reader



David R. Whipple, Chairman
Department of Administrative Sciences

ABSTRACT

The use of Computer-Aided Software Engineering (CASE) tools has been viewed as a remedy for the software development crisis by achieving improved productivity and system quality via the automation of all or part of the software engineering process. The proliferation and tremendous variety of tools available have stretched the understanding of experienced practitioners and has had a profound impact on the software engineering process itself. To understand what a tool does and compare it to similar tools is a formidable task given the existing diversity of functionality. This thesis investigates what tools are available, proposes a general classification scheme to assist those investigating tools to decide where a tool falls within the software engineering process and identifies a tool's capabilities and limitations. This thesis also provides guidance for the evaluation of a tool and evaluates three commercially available tools.



ACQUISITION	
PROJECT	
DATE	
UNIT	
JUST	
By	
DATE	
APPROVED	
DATE	
Dist	
A-1	

Thesis Disclaimer

The following trademarks are used throughout this thesis:

3COM+	is a Registered Trademark of 3COM Corporation
Ada	is a Registered Trademark of the U.S. Government (Ada Joint Program Office)
AD/Cycle	is a Registered Trademark of International Business Machines Corporation
Analysis/Design Workbench	is a Registered Trademark of KnowledgeWare, Inc
Apollo	is a Registered Trademark of Apollo Computer Incorporated
AT&T 6300/AT&T StarLAN	are Registered Trademarks of AT&T Bell Laboratories
AutoCAD	is a Registered Trademark of Autodesk Incorporated
BAT	is a Registered Trademark of McCabe & Associates, Inc
CCC	is a Registered Trademark of Softool Corporation
Compaq III/Compaq Plus/Compaq Portable 286	are Registered Trademarks of Compaq Computer Corporation
Cross Systems Product	is a Registered Trademark of International Business Machines Corporation
Customizer	is a Registered Trademark of Index Technology Corporation
Data Analyst	is a Registered Trademark of Bachman Information Systems, Inc
Developmate	is a Registered Trademark of International Business Machines Corporation
DSEE	is a Registered Trademark of Hewlett-Packard Company
EPOS	is a Registered Trademark of Software Product & Services Incorporated
Epson/FX100/LQ1500	are Registered Trademarks of Epson America Incorporated
Expert Systems Environment	is a Registered Trademark of International Business Machines Corporation
Excelerator/IS	is a Registered Trademark of Index Technology Corporation

FrameMaker	is a Registered Trademark of Frame Technology Corp
HP 9000/Laserjet/ HP/Vectra/HP7475A	are Registered Trademarks of Hewlett-Packard Company
HP SoftBench/ HP Encapsulator	is a Registered Trademark of Hewlett-Packard Company
IBM	is a Registered Trademark of International Business Machines Corporation
IBM PC-DOS/Proprinters	are Registered Trademarks of International Business Machines Corporation
IBM PC LAN	is a Registered Trademark of International Business Machines Corporation
IBM Token Ring	is a Registered Trademark of International Business Machines Corporation
IBM XT/AT/PS/2	are Registered Trademarks of International Business Machines Corporation
IEF/TI 855	are Registered Trademarks of Texas Instruments Inc
InterLeaf	is a Registered Trademark of InterLeaf, Inc
KEE	is a Registered Trademark of International Business Machines Corporation
KeyOne	is a Registered Trademark of LPS s.r.l.
Knowledge Tool	is a Registered Trademark of International Business Machines Corporation
MicroSTEP	is a Registered Trademark of SysCorp International
MS-DOS	is a Registered Trademark of Microsoft Corporation
Novell Advanced Netware	is a Registered Trademark of Novell Inc
Novell ELS Netware 286	is a Registered Trademark of Novell Inc
PC Prism	is a Registered Trademark of Index Technology Corporation
Planning Workbench	is a Registered Trademark of KnowledgeWare, Inc
RCS	is a Registered Trademark of Hewlett-Packard Company
Refine	is a Registered Trademark of Reasoning Systems, Inc.
Repository Manager	is a Registered Trademark of International Business Machines Corporation
Saber-C	is a Registered Trademark of Saber Software, Inc

SES/workbench	is a Registered Trademark of Scientific & Engineering Software, Inc
Software BackPlane	is a Registered Trademark of Atherton Technology
START	is a Registered Trademark of McCabe & Associates, Inc
StP	is a Registered Trademark of Interactive Development Environments
StP/INGRES Interface	is a Registered Trademark of Softool Corporation
StP/TESTBED Interface	is a Registered Trademark of IGL Technology
SYBASE SQL Server	is a Registered Trademark of Sybase Incorporated
Sun/Sparcstation	are Registered Trademarks of Sun Microsystems Incorporated
TIRS	is a Registered Trademark of International Business Machines Corporation
Toshiba P1350/P1351/P351	are Registered Trademarks of Toshiba America Inc
UNIFACE	is a Registered Trademark of Uniface B. V.
Unix	is a Registered Trademark of AT&T Bellaboratories
VADS	is a Registered Trademark of Verdix Corporation
VAX/MicroVAX/VAXstation/ VAXset/DECstation	are Registered Trademarks of Digital Equipment Corporation
XL/Doc	is a Registered Trademark of Index Technology Corporation
XL/Interface	is a Registered Trademark of Index Technology Corporation
XL/Programmer Interface	is a Registered Trademark of Index Technology Corporation

GLOSSARY

Data Item Descriptions (DID's): DID's describe the set of documents for recording information required by the DoD STD-2167A.

Encyclopedia: A database that stores information created by an integrated set of CASE tools.

Framework: An architecture for the integration of a collection of CASE tools designed to form a single integrated environment with a consistent user interface.

Product Baseline: The software as designed, tested, and implemented prior to installation.

Project Management: All the tasks associated with the role of the project manager including planning, estimating and monitoring the progress of a software development project. Support for project management includes a set of well-known tools and procedures such as cost and size modeling, critical path methods, schedule charts, (Gantt charts or timelines), resource loading, spreadsheets, work breakdown structure, status reporting, electronic mail, milestone definition, budgeting, expense tracking, capital allocation, problem tracking and change authorization.

Rapid Prototyping: Quick and inexpensive construction of high fidelity simulation of an interactive system for whatever purpose (i.e., requirements definition). Used to convey the look and feel of a system. Depends heavily on automated tool support like data dictionaries, screen formatters and painters, report generators and very high level languages like fourth generation languages and functional languages.

Repository: The database management facility of the CASE environment which provides data integration services among all the tools in the environment. It saves design information in an abstract form like an Encyclopedia, but also captures project and enterprise information.

Software Development Plan (SDP): A single document outlining the steps for conducting the activities required by the standard.

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	BACKGROUND	2
1.	Rising Software Costs	2
2.	Software Development Crisis	2
B.	WHY CASE	4
C.	CASE OBJECTIVES	4
D.	TOOL EXPLOSION	5
E.	RESEARCH FOCUS	5
F.	THESIS ORGANIZATION	6
II.	THE FULL CASE ENVIRONMENT	7
A.	WHAT IS CASE	7
B.	EVOLUTION OF CASE	8
1.	Origin of Case	8
2.	CASE Arrives	9
3.	The CASE Environment	9
C.	ENVIRONMENTAL ELEMENTS	10
1.	Toolset	10
2.	Integration Architecture	11
3.	Toolkits (CASE) vs Workbenches (ICASE)	13
a.	Toolkits (CASE)	13
b.	Workbenches (ICASE)	15
c.	CASE vs ICASE	16

4.	Repository	16
5.	Methodology	19
D.	THE FULL CASE ENVIRONMENT	20
E.	CASE TRENDS	23
1.	Integration Architectures	23
2.	Specification Compilers	26
F.	SUMMARY	27
III.	IMPACT OF DoD STD-2167A ON CASE	28
A.	BACKGROUND	28
B.	APPLICABILITY OF DoD STD-2167A	29
C.	SOFTWARE DEVELOPMENT PROCESS	30
D.	IMPACT ON CASE	31
1.	Documentation Requirements	33
2.	Traceability of Requirements	35
3.	First Generation Support Tools	36
4.	Second Generation Support Tools	36
E.	SUMMARY	37
IV.	TCOL TAXONOMY	38
A.	CLASSIFICATION GOAL	38
B.	CLASSIFICATION STRATEGY	39
1.	Framework	39
2.	Categories	40
a.	Lifecycle Coverage	40
b.	Integration Level	41
c.	Application Areas	43
3.	Attributes	45

4. Employment	45
C. SURVEYS	47
D. SUMMARY	48
V. TOOL EVALUATION PROCESS	49
A. PREFACE	50
B. EVALUATION CRITERIA	50
C. ASSESSMENT PROCESS	51
1. Needs Analysis	52
2. Environment Analysis	53
3. Develop Candidate List	53
4. Apply Criteria and Select	56
a. Establish Evaluative Criteria	57
b. Define a Specific Experiment	57
c. Execute the Experiment	58
d. Analyze the Results	58
D. SUMMARY	59
VI. Tool Evaluations	62
A. EXCELERATOR/IS 1.9 OF INDEX TECHNOLOGY CORPORATION	63
1. Hardware/Operating System Evaluated On	63
2. Tool Description	63
3. Methodology Supported	64
4. Hardware/Operating Systems Requirements	64
5. Installation	65
6. Documentation	66
7. Interface to Other Products	68

8.	Multi-user Support	70
9.	Network Support	70
10.	DoD STD-2167A Support	71
11.	User-Interface	72
12.	Traceability of Requirements	73
13.	Dictionary/Repository	74
14.	Prototyping	77
15.	Consistency/Completeness Checking	78
16.	Training Support	81
17.	Diagramming/Graphic Facilities	82
B.	StP 4.2A (SUN) OF INTERACTIVE DEVELOPMENT ENVIRONMENTS	86
1.	Hardware/Operating System Evaluated On	86
2.	Tool Description	86
3.	Methodology Supported	87
4.	Hardware/Operating Systems Supported	89
5.	Installation	90
6.	Documentation	90
7.	Interface to Other Products	94
8.	Multi-user Support	96
9.	Network Support	98
10.	DoD STD-2167A Support	100
11.	User-Interface	102
12.	Traceability of Requirements	104
13.	Dictionary/Repository	105
14.	Prototyping	107

15. Consistency/Completeness Checking	109
16. Training Support	111
17. Diagramming/Graphic Facilities	112
C. EPOS 4.0 (PC-Vers) OF SOFTWARE PRODUCTS & SERVICES, INC	115
1. Hardware/Operating System Evaluated On	115
2. Tool Description	116
3. Methodology Supported	116
4. Hardware/Operating Systems Requirements	117
5. Installation	118
6. Documentation	120
7. Interface to Other Products	123
8. Multi-user Support	124
9. Network Support	125
10. DoD STD-2167A Support	125
11. User-Interface	126
12. Traceability of Requirements	128
13. Dictionary/Repository	131
14. Prototyping	134
15. Consistency/Completeness Checking	135
16. Training Support	137
17. Diagramming/Graphic Facilities	138
D. OVERALL EVALUATION	139
E. SUMMARY	140
VII. CONCLUSIONS	142
APPENDIX A - SAMPLE TOOL TAXONOMY FORM	145

APPENDIX B - TOOL TAXONOMY	147
APPENDIX C - BLANK TOOL TAXONOMY FORM	155
APPENDIX D - TOOL EVALUATION CHECKLIST	157
LIST OF REFERENCES	174
INITIAL DISTRIBUTION LIST	177

I. INTRODUCTION

Cost overruns, delivery postponements and the production of ineffective or inadequate systems has characterized the software development process within the software engineering industry and the Department of Defense (DoD). Both industry and DoD have explored many options to address these shortfalls and reduce both software costs and application backlogs. The application of software engineering and the use of software development methodologies helped, but did not provide the desired impact. Computer Aided Software Engineering (CASE) tools were then advertised as a remedy for the software development crisis by automating analysis, design, and coding, but met with little initial success due to the immature technology and limited tool availability. However, the recent revolutions in CASE technology have caused an explosion in tool capabilities and availability. The assorted features and capabilities now available have greatly increased the complexity of their evaluation. This thesis will examine the tools available and their range of capabilities and evaluate a discrete sample of tools.

A. BACKGROUND

1. Rising Software Costs

Both the Department of Defense (DoD) and industry are expending enormous amounts of time and money developing and maintaining software systems with costs continuing to rise. Figure 1 reflects the trends of software costs noted by Boehm [Ref. 1:pp. 32-33].

Software Costs (Billions)		
	<u>1985</u>	<u>1990</u>
World	140	250
U.S.	70	125
	<u>1985</u>	<u>1995</u>
DoD	11	36

Figure 1-1 Rising Software Costs

The sheer magnitude of these costs and the pending budget reductions necessitate serious considerations by DoD to understand and control software costs.

2. Software Development Crisis

DoD and other federal software development efforts have been plagued by cost overruns, postponements and the delivery of ineffective or inadequate systems. The extent of the problem is evidenced by the following statistics:

A U.S. Army study of several federal projects found that:

- 47 percent were delivered, but not used
- 29 percent were paid for, but not delivered
- 19 percent abandoned or reworked
- 3 percent were used after changes were made
- only 2 percent were used as delivered.

For a U.S. Air Force command and control system:

- the initial estimate was \$1.5 million
- the winner's bid was \$400,000
- the actual project cost was \$3.7 million [Ref. 2: p.51].

In addition to the problems noted above, DoD faces another pending reality of the software crisis regarding the backlog of systems needed and the maintenance requirements of existing software systems: The lack of personnel to perform such efforts. This issue is best reflected in the following statements:

The backlog for software development in both the DP/MIS and the Aerospace, Defense, Engineering (ADE) sectors is large and growing at an accelerating pace, and the supply of professionals to address this backlog is severely limited. [Ref. 3: p. viii].

...the national demand for software is rising at least 12 percent per year, while the supply of people who produce software is increasing about four percent per year; this leaves a cumulative four percent gap [Ref. 4].

...25 percent of the draft age population will be required to maintain DoD software by the year 2000 [Ref. 5].

The quality and productivity issues cited above are compounded by the increasing complexity of software systems as well. DoD cannot ignore these issues. Software Engineering has mitigated some of the impact of these issues, but in order to achieve the quality and productivity required DoD must rely on the automation of all aspects of software development.

B. WHY CASE

The fundamental purpose of CASE is to allow developers to produce higher quality software more quickly with less effort [Ref. 3:p. viii]. CASE focuses on automating the activities of software developers. Automating these activities increases quality and productivity at the same time [Ref. 2:p. 49].

C. CASE OBJECTIVES

CASE is not just confined to quality and productivity. McClure cited the following objectives for CASE based on the potential it offers:

- Improve productivity
- Improve software quality and reduce errors
- Speed up the software development process
- Reduce software costs
- Automate software development and maintenance
- Automate generation of software documentation
- Automate generation of code
- Automate error checking
- Automate project management
- Formalize and standardize software documentation
- Promote greater control of the software development process
- Integrate tools and methodologies of software engineering
- Promote software reusability
- Improve software portability [Ref. 6].

The potential benefits promised by achieving these objectives are compelling. They require significant capabilities in order to achieve them. The lure of and need for the potential benefits have fueled an intense effort by CASE vendors. In the past several years, the capabilities of CASE have increased to a point where CASE has evolved from a concept to an industry. "Major computer and workstation companies and many of the 'Big Eight'¹ accounting firms now have dedicated CASE product or service groups." [Ref. 3:p. vii]

D. TOOL EXPLOSION

In 1988, there were over 100 CASE vendors, each marketing one or more CASE products [Ref. 3:p. vii]. By 1989, the number of CASE vendors had doubled to 200 [Ref. 7:p. 1]. Currently, there are over 350 vendors and in excess of 500 tools on the market [Ref. 8].

E. RESEARCH FOCUS

This thesis will investigate what tools exist, examine the 2167A impact on tool requirements, identify a general classification and evaluation scheme and evaluation checklist for tools. Specifically, this thesis will survey several vendors and institutions for CASE tools currently available for developing software systems and evaluate three tools

¹ The "Big Eight" are now the "Super Six".

currently available in the commercial market. The intended target audience is Project Managers/Planners, Systems Engineers and Systems Analysts within the DoD.

F. THESIS ORGANIZATION

Chapter II presents an overview of the CASE environment and its composition. A synopsis of the major CASE toolsets is provided along with some future trends in CASE development.

Chapter III provides an overview of DoD STD-2167A and the comprehensive framework it details. It identifies the major areas suitable for CASE application and the evolution of tools for supporting the documentation requirements imposed by the standard.

Chapter IV provides general categories and capabilities of CASE tools currently available and identifies a general classification scheme for several commercial tools surveyed within the framework detailed by DoD STD-2167A.

Chapter V describes the current state of CASE evaluation efforts and introduces a tool evaluation process. It also identifies several governmental organizations available for supplying information on CASE tools.

Chapter VI contains the personal evaluations of three commercial tools currently available within the commercial market.

Chapter VII summarizes the contents of this work.

II. THE FULL CASE ENVIRONMENT

CASE is no longer just individual tools targeted for specific activities within the software development process it is a vast collection of tools that contribute to a total CASE environment. This chapter describes the evolution of CASE and contrasts CASE as toolkits and workbenches. It also identifies the crucial role of integration and other critical elements of the full CASE environment. The chapter ends by providing an overview of future trends in CASE development.

A. WHAT IS CASE

Computer Aided Software Engineering involves the use of computers to aid the software development process. This simplistic view has characterized CASE since its development in the early 1970's. However, CASE has become much more than automated tool support for the software engineering process. Today CASE has evolved into a total systems approach to the design and production of software, as evidenced by the wide variety of tools available which contribute to the CASE environment. This changing view is reflected in the following definitions of the 11 definitions of CASE recently published by the CASE Studies Consortium:

CASE (PROCESS)

CASE (software engineering): "the establishment and use of sound engineering principles in order to obtain economically the software that is reliable and works efficiently on real machines." It encompasses a set of three key elements -- methods, tools, and procedures -- that enable the manager to control the process of software development and provide the practitioner with a foundation for building high quality software in a productive manner.

Pressman [Ref. 9:p.277]

CASE METHOD

An interlocking set of formal techniques in which enterprise models, data models, and process models are built up in a comprehensive knowledge base and are used to create and maintain data processing systems. Or, an enterprise-wide set of automated disciplines for getting the right information to the right people at the right time.

James Martin [Ref. 9:p. 276]

CASE (BEHAVIORAL)

CASE is the rigorous implementation of well-integrated methods, procedures and tools optimizing human behavior and technology to improve the productivity of software development.

Bartner Group [Ref. 9:p. 279]

B. EVOLUTION OF CASE

1. Origin of Case

The concept of CASE grew out of early efforts of using computers to assist with systems analysis and design in the early 1970's. One product called Problem Statement Language/Problem Analyzer (PSL/PSA) is recognized by some as the original CASE tool. It was developed by Dr. Daniel Teichrowe at the University of Michigan and designed to run on

large mainframe computers. User requirements were specified in PSL and analyzed by the PSA. PSL/PSA's goal was to eventually generate code from the requirements statement. Its only problem was that it required too much computer resources to function. Few companies could afford dedicated PSL/PSA computers nor could they release access time from their own production machines. This product and others like it were the early forerunners of CASE. [Ref. 10:p. 4]

2. CASE Arrives

In the late 1970's and early 1980's, graphical modeling techniques of structured analysis (along with fourth generation languages [Ref. 7:p. 1]) began to spread throughout systems development organizations fueling the dependence on automated resources. But, even these efforts were limited by the lack of affordable automated support. The advent of powerful graphic workstations in the mid 1980's, however, gave rise to the industry known as Computer Aided Software Engineering. [Ref. 11:pp. 126-128]

3. The CASE Environment

In the past few years, a rapid series of new approaches have been adopted including: information engineering, entity-relationship modeling, automatic code generation, real time design, object-oriented techniques, rapid prototyping, software simulation, visual programming and reverse engineering, among others. The distinction between

CASE and its support environment has blurred since CASE has incorporated most of the aspects of software development. CASE has become a general term encompassing:

- Planning and estimating
- Requirements analysis
- Architectural design
- Detailed design
- Prototyping
- Programming
- Maintenance
- Documentation
- Reverse engineering
- Project management
- Testing

Configuration management [Ref. 7:p. 1]

Indeed, the CASE environment provides "... support to the entire engineering team (i.e., managers, analysts, designers, programmers, maintainers, etc...) for overall product development" [Ref. 12:p. 20].

C. ENVIRONMENTAL ELEMENTS

1. Toolset

From a systems view, CASE includes any computerized tool that automates a portion of the software development

lifecycle. This view is shared by many in the industry.

According to one consulting group:

"There is no reason, for example, that the many high-productivity applications development systems on microcomputers, such as screen generators, cross-tabulation systems, fourth-generation languages, and so on, cannot be included in the range of CASE tools as long as they can be integrated with the existing CASE tools and are controlled in this own use in an engineering sense. [Ref. 2:pp. 24-25]

Therefore, any computerized tool that automates a portion of the software development lifecycle should be included:

"Even traditional software tools, such as editors and compilers, must now be considered part of the CASE toolset in the sense that they will eventually share data with the central design database used by all other tools." [Ref. 3:p. vii]

Toolsets that integrate traditional tools for documentation, design, source code generation, compilation and testing already exist while new tools which combine "many of the traditional feature sets with new capabilities such as graphical program design and reverse engineering, all operating from a single design database" are emerging [Ref. 3:p. vii].

2. Integration Architecture

The CASE environment requires a tightly-coupled toolset. The key to a tightly-coupled, consistent CASE environment is the integration capability provided. There are

three distinct aspects of full integration in the CASE environment: presentation integration, data integration and control integration.

Presentation integration is concerned with providing a common user interface (i.e. standard menu interface) for accessing a toolset and a common look and feel (i.e., similar menu characteristics, iconic behavior, etc.). Facilities such as X-Windows along with look and feel guidelines Motif (the OSF standard) and Open Look (from Unix International) support presentation efforts. [Ref. 13:p. 11]

"Data integration involves mechanisms enabling CASE tools to share and manage information" [Ref. 13:p. 11] which is primarily a function of the database of the CASE toolset. It relies on a database management facility with typical capabilities such as data access, security and recovery capabilities. However, the full CASE environment imposes special requirements which distinguish the environment repository from traditional commercial databases. The environment repository must be able to define both a schematic and semantic description of the contents of the database to provide standardized information to support true information sharing among tools and automated consistency checking. Moreover, the repository must record and manage the relationships and dependencies among data elements to support configuration management and other features. [Ref. 13:p. 11]

Communication between individual tools is accomplished via mechanisms provided by control integration. Tools must be able to communicate with one another in order to synchronize activities and perform user defined task sequences. This function is partly accomplished by the repository and partly by an additional layer associated with the repository, but separate from it. Special requirements such as rule enforcements involving certain changes in the data which invoke certain actions (i.e., integrity checks) are generally accomplished by a trigger facility within the repository. However, a control integration layer is normally used to provide generalized message passing, tool invocation, methodology guidance and process control. [Ref. 13:p. 11]

The high degree of functionality required by the CASE environment and the lack of standardization among tools makes full integration a challenging effort. As a result, vendors are approaching it in different ways.

3. Toolkits (CASE) vs Workbenches (ICASE)

The degree of integration and scope of a toolset has become a major delineation between CASE products. The two major toolset distinctions noted by this author are: Toolkits (CASE) vs Workbenches (ICASE).

a. Toolkits (CASE)

The first distinctive type of toolset noted by the author are toolkits which Loh and Nelson refer to as:

... a set of integrated CASE tools designed to work together to automate, or partially automate, a particular development job or a single phase of the systems development cycle. [Ref. 14:p. 31]

Toolkits can vary because vendors bundle various tools together to target particular user problems. Examples include analysis and design toolkits, data design toolkits, programmer's toolkits, code generator toolkits, maintenance toolkits and project management toolkits. The toolkits normally provide a user shell specifically prepared for the target user. Figure 2-1 contains the basic components normally found in a toolkit according to Hanner.

-
1. Window, screen, report, graph and other output formatting editors
 2. Program flow editors including data flow diagrams, traditional flow charts, and ERD's
 3. Schema design and data dictionary managers to build and maintain the CASE Data Dictionary
 4. Code management systems for version control and code maintenance
 5. Program development tools including fourth generation languages, prototyping tools and application generators
 6. Bug reporting and tracking to allow automated program maintenance
 7. Network management tools
-

Figure 2-1 CASE Toolkit Components

In addition to the components of a toolkit, Hanner also described "...several characteristics of CASE

tools that bridge all user types". He cited the following common CASE features:

Data Dictionary (Single most vital part of tool) allows easy cross-referencing and access to all objects known to the tool

Visual/graphic exposition of programs and data (i.e., Dataflow Diagrams, Entity-Relationship Diagrams)

Automated consistency checking of data and program elements

Multi-user data access (concurrent data access by multiple users)

Prototyping [Ref. 14:p. 40]

b. Workbenches (ICASE)

Workbenches are the second distinctive type of toolset noted by the author which Loh and Nelson define as:

...integrated CASE tools that assist across all phases of the systems development cycle--planning, analysis and design, implementation and maintenance. [Ref. 14:p. 31]

The major differences between these toolsets are the integration and coverage of the development cycle provided. Workbenches provide seamless integration between tools to provide full coverage and support all activities within the development cycle. However, individual workbenches alone may not be sufficient, "since most do not include robust cross life cycle tools such as project management and configuration management, and testing and other quality assurance tools are typically primitive or missing entirely"[Ref. 15:p. 11]. In addition, they tend to focus on

particular application areas (i.e., business or engineering) and incorporate a single methodology.

c. CASE vs ICASE

Martin refers to the major difference between toolsets as CASE vs ICASE. He considers CASE to be "power tools" which focus on particular aspects of development and ICASE (Integrated CASE) as toolkits which contain tools "for all aspects of software development" that are integrated via a repository he calls an Encyclopedia. Once again, coverage of the development process is a distinguishing characteristic although he also emphasizes the generation of executable programs as a critical characteristic. [Ref. 16:pp. 5-6]

Consequently, the ideal environment can be accomplished in two ways: combining various toolkits via a **framework** or using a workbench if limited to one particular application area (provided a workbench supporting the application exists). A completely integrated **full lifecycle** toolkit has not yet been achieved, but is fastly approaching. [Ref. 15:p. 11]

4. Repository

The critical element of any CASE system is the repository or centralized database used to accumulate the information related to an application. It does not just store the data, but **the meaning of the data as well**. For example, it may employ rule processing routines to determine how

processes on a dataflow diagram are to be linked or data elements are to be referred to. These routines can be used to help achieve "accuracy, integrity and completeness of the plans, models and designs"[Ref. 16:p. 23] thus becoming a knowledge base for not only storing information, but controlling its validity and accuracy.

Storage is not the only function of the repository. As noted above, CASE tools can only achieve full integration by **sharing** a common database allowing multiple tools to share the same object. Therefore, one tool such as dataflow diagramming tool can share information with entity-relationship modeling tool to construct an application, further enhancing consistency and completeness of an application. [Ref. 15:p. 39]

Ideally, the repository should:

Enable one tool to use information derived from input to other tools

Provide analysis and consistency checking across all phases

Increase the level and feedback from the detail specification in the back-end phases to the more abstract front-end specifications

Support project-wide configuration management and requirements tracking
[Ref. 3:p. xvi]

Figure 2-2 illustrates the role of the repository in the CASE environment [Ref. 3:p. xvi].² Although mature repositories are not yet widely available, several are under

² CASE tools designed to assist in the System Planning, Analysis and Logical Design phases are referred to as Front-end tools while tools supporting Physical Design and Construction are referred to as Back-end tools in trade publications.

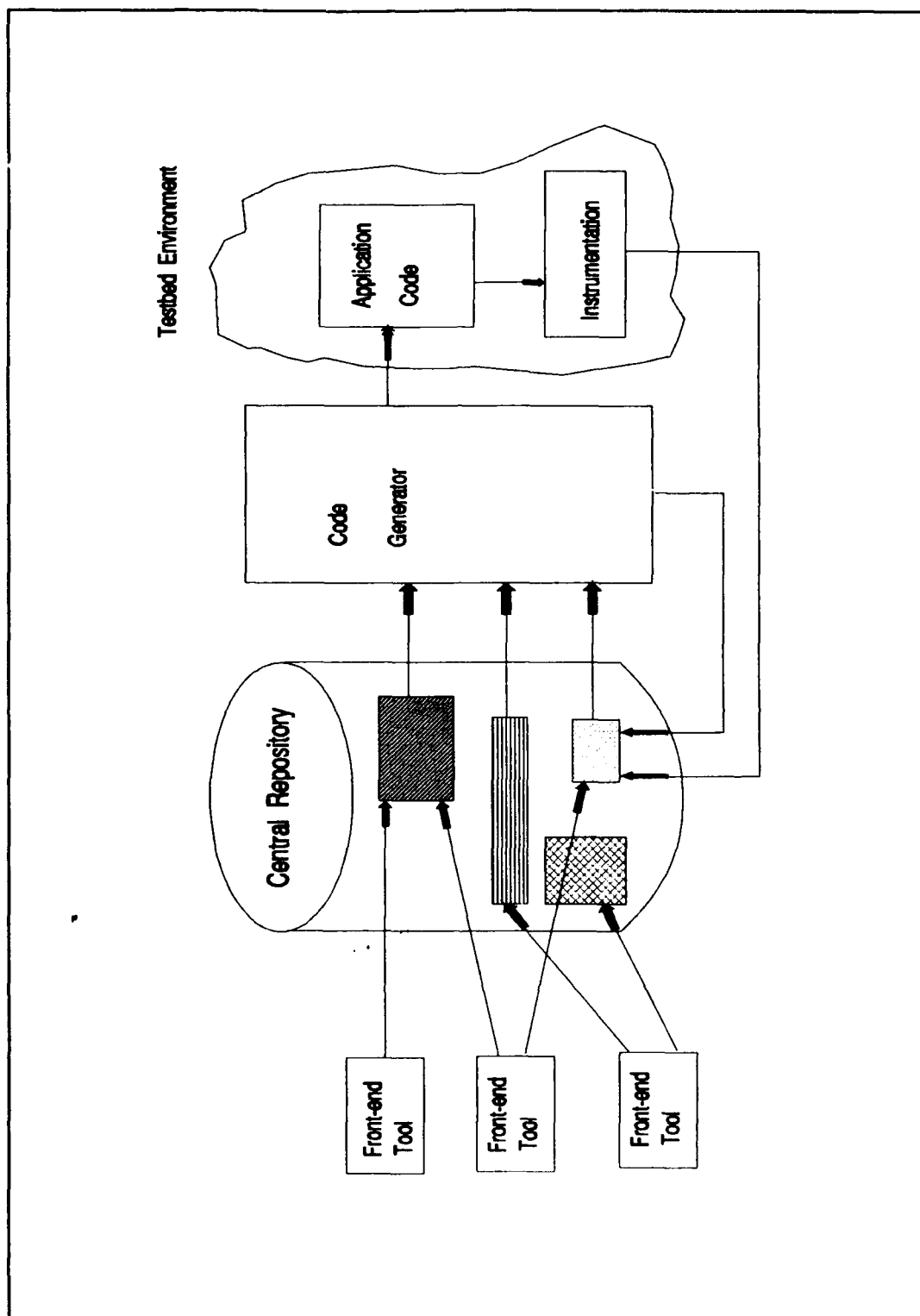


Figure 2-2 The CASE Repository

development. The most notable is International Business Machines (IBM's) recently announced "Repository Manager" [Ref. 17:p. 3].

5. Methodology

Tools in and of themselves are not enough. For CASE to be successful, the organization must thoroughly understand the software development process and how to apply the tools at the points of greatest leverage which implies an organization must adopt a systems view of the development cycle for its particular environment. Experience indicates that unless tools operate within the constraints of an overall design discipline (i.e., methodology) they cannot be effective. [Ref. 3:p. x]

However, Wallace points out that many CASE implementations are unsuccessful because organizations misuse or abuse the methodologies employed by confusing the techniques used with the method itself [Ref. 18:p. 17]. This point is best illuminated by distinguishing between a technique and a methodology.

A **technique** describes the rules and notations for representing the requirements and design of a system in commonly understood terms. Most tools today rely on graphics-based techniques, such as dataflow diagrams and system structure charts, to communicate between the developer and the end user.

A methodology "is a system of methods, rules, and the set of procedural steps to be followed in order to achieve a desired end" [Ref. 2:p. 17]. It describes the required process and deliverables at each phase of the development lifecycle by answering the questions regarding which work products to produce, when to produce them and who does the work. Techniques resolve "how" work products are to be produced. [Ref. 18:p. 17]

Tools automate tasks and techniques. Some "impose a standard technique and methodology, some adapt to user notations and methods, many can do both" [Ref. 3:p. x]. However, it is essential that a design discipline be fully understood and accepted by software developers and endorsed by management for a tool to be truly effective. Hence, methodology plays a crucial role in the CASE environment by providing an infrastructure for controlling CASE techniques.

D. THE FULL CASE ENVIRONMENT

The full CASE environment provides a wide assortment of tools for specific phases of the software development lifecycle (vertical tools), as well as tools that span the entire development process (horizontal tools). Figure 2-3 depicts the full CASE environment and the various layers of integration supporting it. Figure 2-4 contains several definitions of particular tools depicted in the full CASE

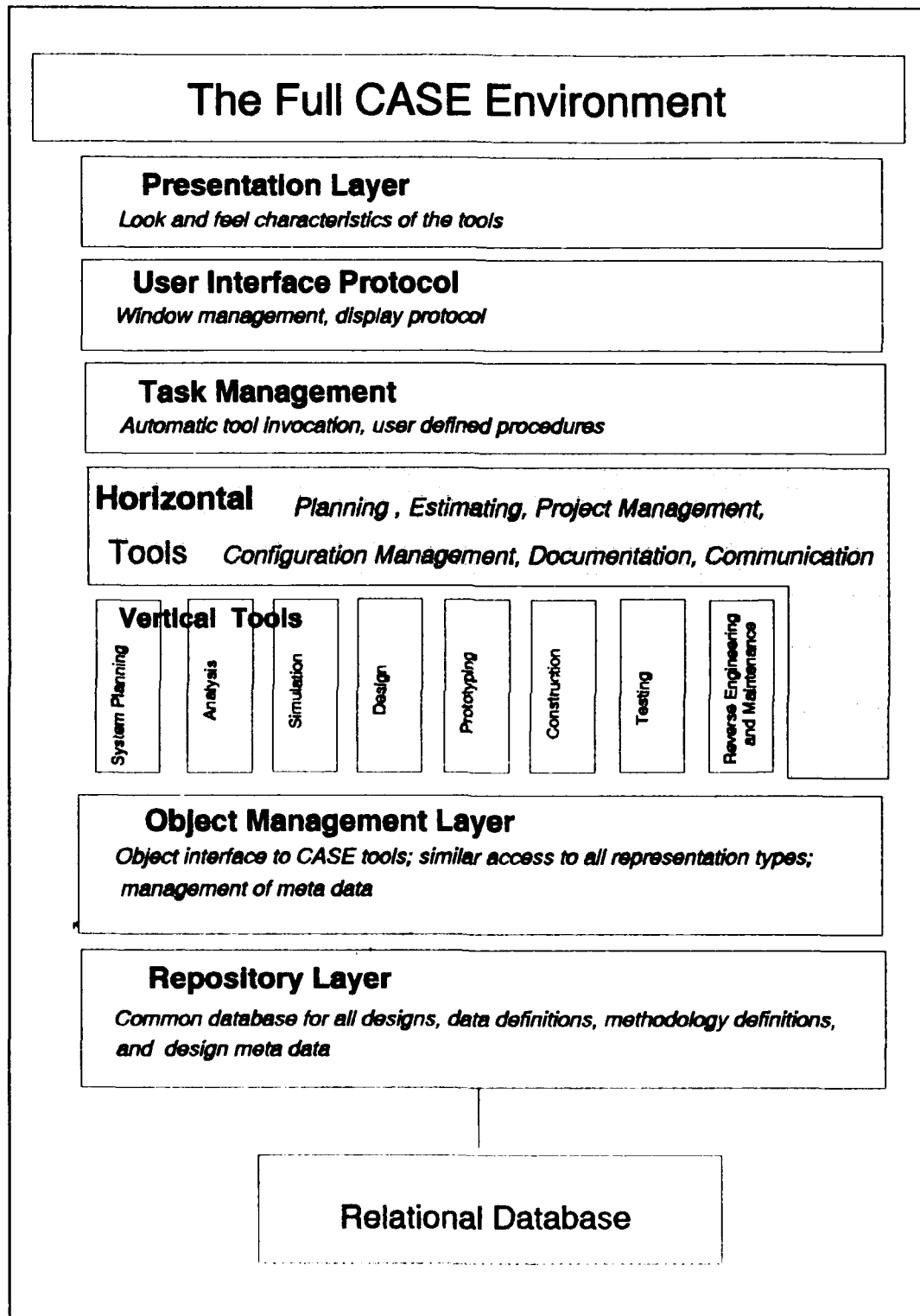


Figure 2-3 The Full CASE Environment

Code Generation: Tool can generate some programming language from analysis and design representation.

Configuration Management: Tool maintains histories of document versions and configurations of documents.

Design: Tool depicts the module structure of a program being designed either in text (structured English, program design language) or graphically in structure charts or modular block diagrams.

Documentation Support: Tools that provide for the extraction and formatting of the contents of the project database. Others go further to provide standard reports, report generators and templates to meet certain standards (i.e., DoD STD-2167A) with interfaces to technical publishing systems from Interleaf, Framemaker, etc...).

Performance Analysis: Tools that measure the complexity software, generate static or dynamic statistics of a program's performance, or analyzes the structure of a program.

Project Management: Tool provides or reports project management information including number of processes, allocation of work, completion status and, in some cases, schedules, budgets and project dependencies.

Prototyping: Tool provides ability to develop screen or report prototypes and generate appropriate code, or provides capability to rapidly develop algorithms and test the code.

Real Time: Tool provides design representations for real time systems (i.e., control flow diagrams, state transition diagrams, process activation tables, state event matrices or equivalent).

Requirements: Tools providing either text or graphic capability to generate or analyze requirements. If graphic, a popular structured analysis technique is used (i.e., Yourdan/Demarco).

Reverse Engineering: Tool is capable of reading source code or database schema and creating the documentation and design representations (structure charts, entity relationship diagrams, module block diagrams, calling trees, etc...) necessary for enhancing and maintaining the code at the analysis and design level. Some tools allow for new code to be generated from the modified designs.

Simulation: Same as prototyping except that the ability to simulate the behavior of the prototype system is also provided.

Strategic Planning: Tool is capable of creating an enterprise model or is used to generate a strategic systems plan.

Testing: Tool provides the capability to generate test beds or test suites from the source code. Also includes capability to assist in system integration testing in the target hardware environment.

Testing & Maintenance: Tool provides the capability to generate test plans and test data and manage the test data.

Traceability of requirements: Tool can track and report the impact of change between documents or trace the development of a requirement throughout the system so compliance and completeness checks are possible.

Figure 2-4 Tool Definitions

environment [Ref. 7:p. 3]. It provides a central repository which is used to accumulate and maintain all application information as well as providing communication among the

various tools. The environment not only incorporates the tools, but the methods and procedures utilized by an organization. T. Capers Jones suggests an ideal CASE environment might contain up to 110 separate software tools [Ref. 19: page xiii].

E. CASE TRENDS

1. Integration Architectures

Full CASE integration is required if the ideal CASE environment is to be achieved. Workbench (ICASE) tools offer a limited environment since a package deal from one vendor may not be able to offer the best tools available for certain activities within the development cycle (i.e., testing). Therefore, the framework approach appears to be the major trend for integration architectures.

IBM's recent announcement of AD/Cycle, IBM's proposed CASE framework architecture for integrating CASE toolkits, represents a ringing endorsement of the framework solution to CASE integration. According to a leading CASE industry publication:

AD/Cycle is an integration architecture or framework for a full life cycle CASE environment. It comprises several layers addressing presentation, data, and control integration. AD/Cycle includes a repository, tool integration services, vertical tools and a common user interface. [Ref. 20:p. 54]

Figure 2-5 provides an overview of the AD/Cycle environment.

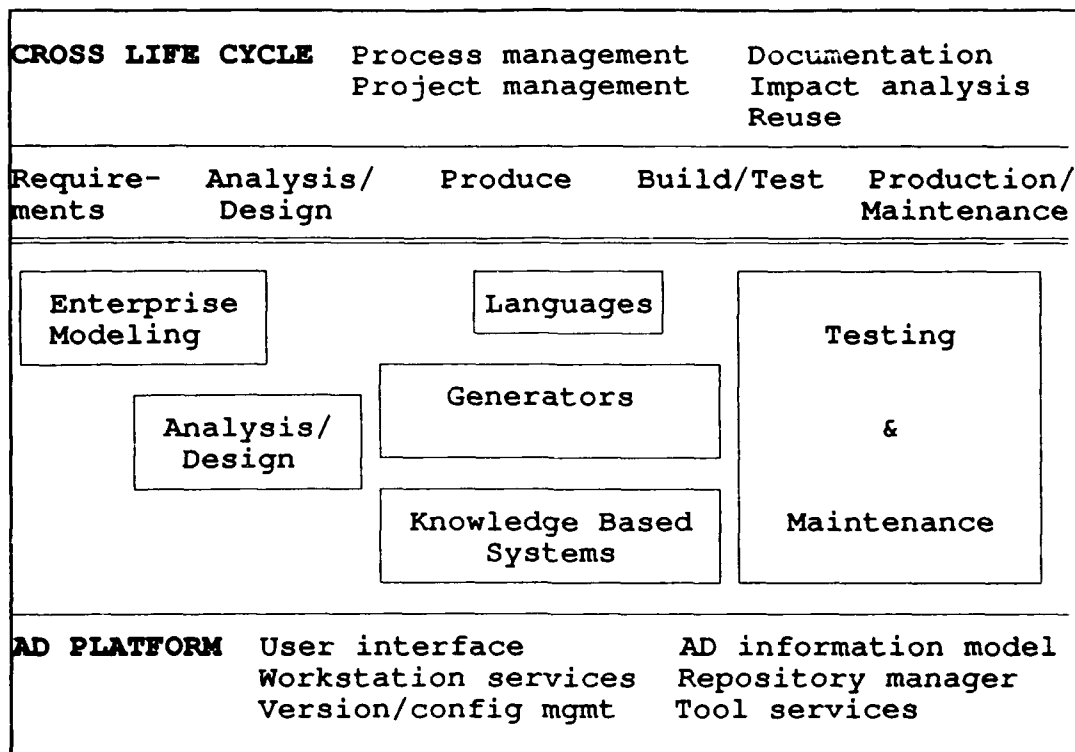


Figure 2-5 AD/Cycle Architecture Chart

AD/Cycle's tool support arsenal includes an impressive array of third party vendors as well as several tools of its own. System planning is supported by IBM's Developmate, Index Technology's PC Prism and KnowledgeWare's Planning Workbench. System analysis and design tasks are supported by Bachman's Data Analyst, Index Technology's Excelerator³ and KnowledgeWare's Analysis/Design Workbench. IBM also bundles their own knowledge engineering products into AD/Cycle to provide artificial intelligence (AI) and expert system capability within AD/Cycle. These tools include KEE, Knowledge Tool, Expert Systems Environment and TIRS. The

³ Excelerator/IS is one of the tools evaluated in chapter VI.

tools identified represent several of the leading tools in the industry which demonstrates the considerable support and interest generated by AD/Cycle.

The key to AD/Cycle's support strategy is the Cross Systems Product (CSP) code generator. Under IBM's approach, all front end CASE tools will target their output to be compatible with the CSP. Moreover, IBM intends for the CSP not only to function as a code generator, but as a mechanism "allowing developers to target application specifications to any desired platform (theoretically) with no additional effort" [Ref. 20:p. 52]. IBM seems to imply that if a vendor can meet CSP specifications AD/Cycle will take care of the unique target details. Initially, compatibility between the tools and the CSP is to be accomplished via an External Source Format (ESF) data transfer interface and will eventually be provided via the AD/Cycle Repository (Repository Manager). [Ref. 20:p. 52]

The announcement of IBM's Repository Manager signals that the complete integration of a toolset is on the near horizon. However, IBM's AD/Cycle is not the only framework architecture. There are several other vendors offering similar products. One example is the Visible Connections open software architecture adopted by Interactive Development

Environment's Software through Pictures⁴ which is described in chapter VI.

2. Specification Compilers

Current code generation tools rely on high-level language compiler technology developed over 25 years ago. A relatively new CASE tool named MicroSTEP⁵ (STEP: Specification to Executable Programs) developed by Dr. Raymond Yeh, represents the possible next step for CASE. The tool allows systems analysts to use personal computers (PC's) and graphical design tools to develop specifications that are machine interpretable. The tool contains a specification compiler which is used to create executable programs directly from the design specifications. By working from a higher level of abstraction, developers can ignore the implementation-specific details of coding and concentrate their attention on the system and its desired behavior.

In addition, STEP facilitates changes and documentation during development and throughout maintenance efforts. Rather than changing the code and then updating the design, developers simply modify the specification and regenerate the new program. Moreover, STEP provides 100% code

⁴ Software through Pictures is one of the tools evaluated in chapter VI.

⁵ MicroStep is one of the tools surveyed in chapter IV.

generation capability, whereas most traditional code generators produce 80%-85% of a programs code.

Specification compilers are based on the assumption that code can be synthesized automatically given a precise specification which implies high-level language compilers are no longer needed. Given the tremendous improvement in quality and productivity resulting when high-level language compilers were introduced, "advances in 'specification compilers' might produce another quantum leap in software productivity" [Ref. 21:pp. 30-32]. Therefore, specification compilers may represent the next step for CASE.

F. SUMMARY

CASE has evolved from a tool or set of tools for software development to a systems approach to software development. The systems perspective implies a CASE environment encompassing the organization as well as all aspects of software development. The CASE environment is a dynamic entity constantly changing to obtain an optimum tool mix or approach depending on the application requirement. A key feature of the environment in the future will be the linkage between tools, systems and management controls to yield the optimum set of tools for a particular application design. Currently, CASE is poised on the threshold of the Full CASE Environment.

III. IMPACT OF DoD STD-2167A ON CASE⁶

This chapter provides an overview of DoD STD-2167A, Defense System Software Development, the comprehensive framework it details and describes its applicability to DoD software projects. The chapter identifies the major areas suitable for the application of CASE and the evolution of tools for supporting the documentation requirements imposed by DoD STD-2167A.

A. BACKGROUND

DoD-STD-2167, the precursor to DoD STD-2167A, was developed out of the recognition by DoD of the need for a standard mechanism for developing requirements specifications. Moreover, the Military contracting community dictated the DoD have a mechanism for specifying detailed defense system requirements that encouraged fair and open bidding by all interested contractors. The need to accurately and completely specify a contract and its set of deliverables necessitated a straightforward well-understood requirements standard such as DoD-STD-2167. [Ref. 23:p. 237]

⁶ The contents of this chapter, unless otherwise indicated, were drawn from [Ref. 22].

DoD STD-2167A superseded DoD-STD-2167 1 April 1987. Developed in conjunction with DoD-STD-2168, the Defense System Software Quality Program, these standards established a well-defined and easily understood software development and acquisition process. All existing DoD standards were superseded which reduced confusion and eliminated conflicts. [Ref. 24:p. 26]

B. APPLICABILITY OF DoD STD-2167A

DoD STD-2167A is approved for use by all Departments and Agencies of the Department of Defense. The intent of the standard is to establish requirements to be applied during the acquisition, development, or support of software systems. The standard provides for total system development when used in conjunction with MIL-STD-499.

The DoD STD-2167A specification format is the standard methodology required for all military system contractors building mission critical software systems. Mission critical projects include:

Intelligence activities

Command and control of military forces

Cryptologic systems relating to national security

Equipment or software forming an integral part or of a weapons system.

Unless specified in the contract, the use of DoD STD-2167A is not required on other system development projects, but it is encouraged. [Ref. 24:p. 28]

C. SOFTWARE DEVELOPMENT PROCESS

DoD STD-2167A is not intended to specify or discourage the use of any particular software development method. The standard permits developers to practice their own software development methodology and even allows them to tailor the standard by eliminating non-applicable requirements. "The standard is compatible with modern methods of software development, and it supports rapid prototyping if the Software Development Plan is tailored and specifies that methodology" [Ref. 24:p. 27]. As a result, the contractor is charged with the responsibility for selecting the process that best supports the achievement of contract requirements. The process selected must include the following activities, which may be overlapped or applied iteratively:

Systems Requirements Analysis/Design

Software Requirements Analysis

Preliminary design

Detailed Design

Coding and Computer Software Unit Testing

Computer Software Component Integration and Testing

Computer Software Configuration Item Testing

System Integration and Testing

Testing and Evaluation

Production and Deployment

Figure 3-1 depicts the standard software development process as mandated by DoD STD-2167A (For clarity, the hardware development processes have been omitted). The standard emphasizes the software development and acquisition process throughout the life cycle by requiring an explicit set of reviews, audits, and deliverable documents at the completion of all milestones.

D. IMPACT ON CASE

The mandate of the DoD STD-2167A format virtually necessitates the application of CASE technology. In fact, the foreword of the standard encourages the use of automated techniques to produce deliverable data. The standard requires a layered top-down approach to design and development emphasizing the requirement analysis and design specification phases of the life cycle. Moreover, DoD STD-2167A requires the employment of well-documented structured methodologies during design and implementation and further specifies that requirements be traceable throughout all layers and phases of the system. As a result, system requirements documents must be written to allow the extraction of compliance information.

[Ref. 23:p. 238]

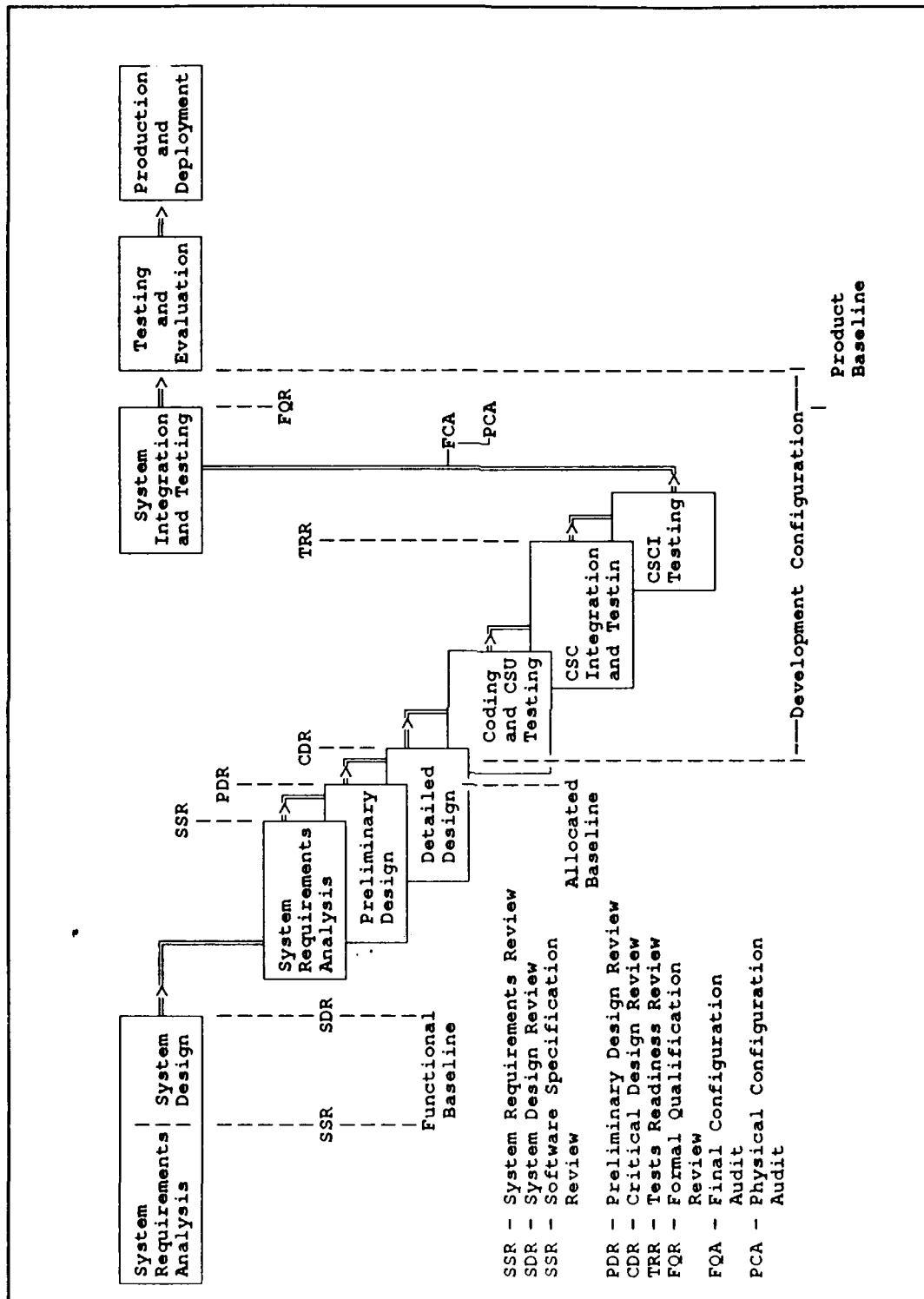


Figure 3-1 DoD STD-2167A Software Development Process

1. Documentation Requirements

Many vendors believe that 30-50% of a system's cost is due to the documentation requirements imposed by DOD STD-2167A [Ref. 25]. The amount of documentation required to support the development effort is enormous. Over 27 separate documents are required which does not include source code and test suites [Ref. 23:p. 240]. Figure 3-2 depicts the main documents required by DoD-STD-2167A (For clarity, Figure 3-1 has been repeated as Figure 3-2). These documents constitute specific deliverables required at the conclusion of a particular development phase. Figure 3-2 also specifies the points where deliverable documents are due and formal audits and reviews are to be completed.

From a CASE standpoint, the most important documents are generated during the requirements analysis, preliminary design, and the detailed design development phases. The key documents are:

- Software Requirements Specification
- Interface Requirements Specification
- Software Top-Level Design Document
- Software Detailed Design Document

These documents are especially suited for CASE since analysis and design skills are required to write and generate them [Ref. 23:p. 240]. There are other required documents

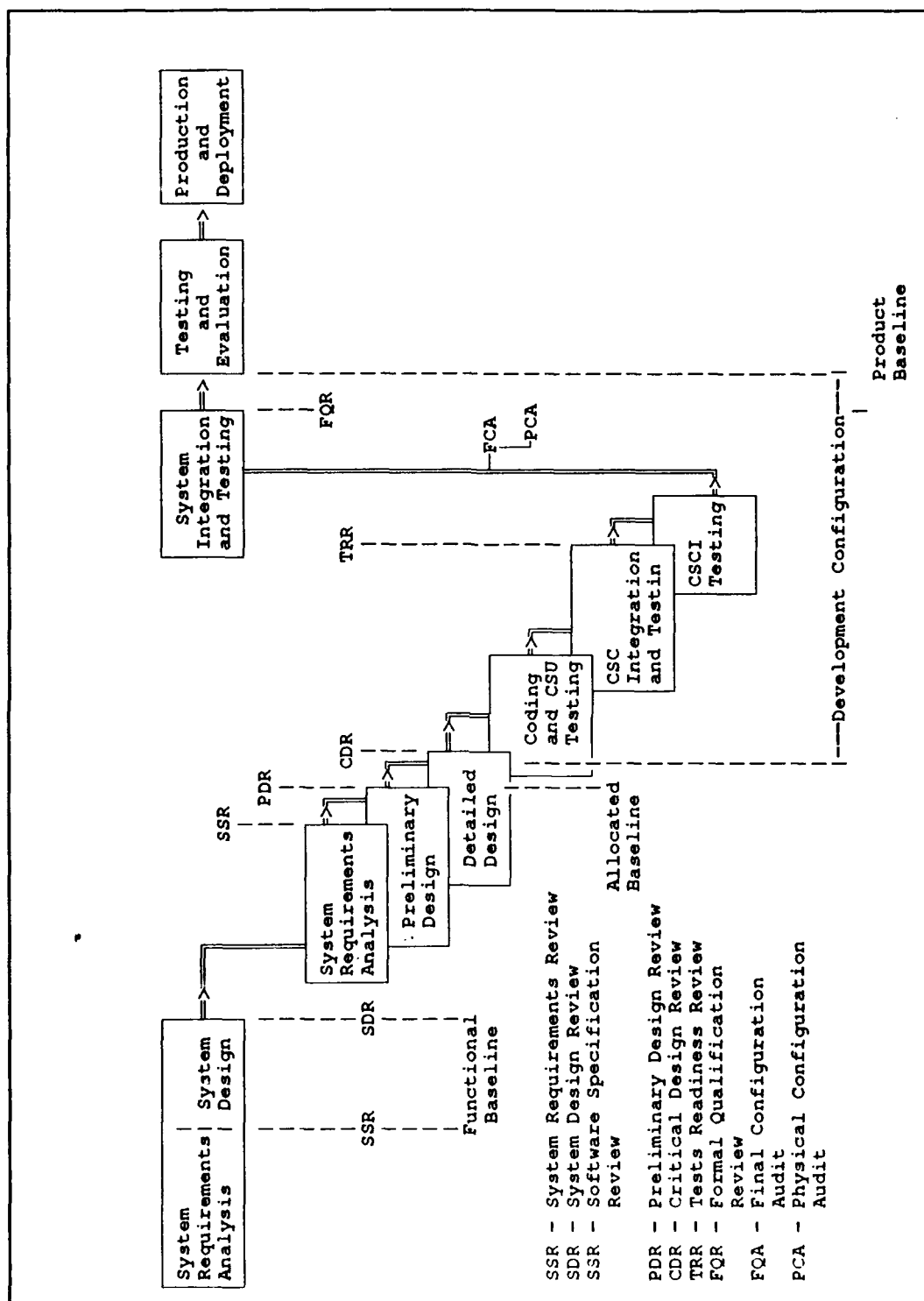


Figure 3-2 DoD STD-2167A Documents and Delivery Points

such as the Operational Concept Document and the Software Development Plan, which are primarily associated with the management of the software project. The emergence of project management tools which interface with analysis and design tools now make these documents candidates for CASE application as well.

2. Traceability of Requirements

Traceability of requirements is another important consideration for CASE application. Section 4.2.8 of DoD STD-2167A dictates that all specification requirements be traceable to the software design. Therefore, the contractor is required to develop traceability matrices to show the allocation of requirements from the system's specification to the individual software components and from the individual software components back to the system's specification. The traceability matrices are documented in the Software Requirements Specification, Software Top-Level Design Document, and the Software Detailed Design Document. Moreover, tests and test cases built to verify the correct operation and performance of the individual components must be traceable to the requirements. Constructing these traceability matrices is a tremendous task, especially for large software systems with hundreds or thousands of individual requirements. As requirements change and evolve

throughout the analysis and design process, this task becomes particularly complex. [Ref. 23:p. 238]

3. First Generation Support Tools

Original efforts to support these requirements were primitive. Word processors were the traditional tools along with manual efforts to cut and paste various CASE diagrams and tables to bridge the documentation gap. Some tools began to employ powerful desktop publishing programs such as Context, Frame, and Interleaf to handle the structured text mixed with graphics common to engineering environments. However, brute force was required to keep the documentation consistent and up-to-date with the CASE design data. These first generation documentation tools laid the foundation for the integration of CASE and automatic documentation. [Ref. 26:pp. 26-27]

4. Second Generation Support Tools

Anderson refers to the current model of documentation tools as "a document synthesis model because it synthesizes or derives, via rules, the detailed document sections directly from CASE data-flow diagrams and structure charts" [Ref. 26:pp. 26-27]. He emphasizes the key to these tools lies in their ability to dynamically link to the CASE design database. Whenever a change is made, the original documentation is dynamically updated via the link instead of destroying the prior version as the first generation tools did. These links help preserve the overall product baseline.

Anderson foresees even tighter data integration between CASE and publishing programs characterized by two environments. One environment is a multi-vendor approach where the integration is provided by linking publishing software from one vendor and a CASE tool from a different vendor. The other is a single-vendor environment where the vendor provides the CASE tool with publishing software incorporated in it. [Ref. 26:pp. 26-27]

E. SUMMARY

DoD STD-2167A provides a comprehensive framework for the software development process. It requires extensive documentation as a part of the development process and the deliverables vital to software projects. The vast documentation required is an area particularly suited for automation and has received substantial consideration for CASE application. The standard also established the importance of being able to detect or trace whether a requirement was identified, but not supported by the system being developed. There are plenty of opportunities to apply CASE within the DoD STD-2167A framework. Organizations which are not involved with defense systems can benefit from the efforts of the standard as well since the requirements traceability concept applies to all software systems. Luckily, many CASE tools today have included mechanisms which establish traceability links between specification and design.

IV. TOOL TAXONOMY

The opening section of this chapter provides an overview of the purpose of the taxonomy. A proposal of how CASE tools can be classified is provided along with a sample format for use by the target audience. The chapter ends by referring to the responses of several vendors and institutions surveyed by the author using the proposed taxonomical format.

A. CLASSIFICATION GOAL

A classification scheme is essential when dealing with diverse, complex items. The tremendous number and variety of tools available today greatly complicate this task. Moreover, given the diversity and functionality of the tools available makes understanding what a tool does and comparing it to other tools a most difficult task. However, it is precisely this diversity that drives the need for a taxonomy.

Simply categorizing the tools is not enough. In addition to the classification framework, description mechanisms are provided to further define the fit and support provided by the tool. Several of the mechanisms are based on the critical areas defined in Chapters II and III. The purpose of this taxonomy is to help organizations involved in evaluating and selecting CASE tools to classify tools and aid in the

development of candidate tool lists as described in the tool evaluation process presented in Chapter V. The framework utilized for the taxonomy is described in the next section.

B. CLASSIFICATION STRATEGY

1. Framework

The Software Development Life Cycle (SDLC) is a standard engineering practice employed by software development organizations to encompass all phases of software development. Yet, even this standard approach can be a cause for user concern when the user is faced with multiple lifecycle models. The lack of single standard lifecycle model for software development can increase the complexity associated with classifying and defining the various tools and the phases they support. In addition, various organizations which adopt the same lifecycle model may name each phase differently. [Ref. 27:p. 114]

Figure 4-1 contains the phases selected by the author for the framework. In the interest of the target audience, this taxonomy will utilize seven of the phases defined by the standard DoD SDLC mandated by DoD-STD-2167A as described in chapter III and two additional phases: Project Management and Other. The development phases are prefaced by project management to reflect the emergence of tool support for this activity. An Other category is provided to accommodate tools

Project Management
System Software Requirements Analysis Phase
Software Requirements Analysis Phase
Preliminary Design Phase
Detailed Design Phase
Coding and Unit Testing Phase
Computer Software Component Integration and Testing Phase
Computer Software Configuration Item Testing Phase
System Integration and Testing Phase
Other

Figure 4-1 Taxonomy Development Phases

which do not match the given phases.⁷

2. Categories

Although CASE is a diverse field given the variety of tools available, there are a few distinct areas in which they can be categorized: lifecycle coverage, integration level and application areas.

a. Lifecycle Coverage

The DoD SDLC framework serves to satisfy lifecycle coverage. Several terms used in the industry also serve to indicate lifecycle coverage. Tools which emphasize upstream activities such as planning, analysis and design are referred to as "Upper CASE" or "Front-end" CASE products.

⁷ DoD STD-2167A does not identify a maintenance phase. Significant tool capabilities now exist for this phase. Additionally, some vendors may have their own representation of the development cycle causing a tool not to fit a particular phase.

Tool which emphasize downstream activities such as programming and maintenance are referred to "Lower CASE" or "back-end CASE" products. [Ref. 28:p. 425]

b. Integration Level

Chapter II addressed the integration architecture of the full CASE environment by describing two basic distinct toolsets: Toolkits and Workbenches. It concluded by asserting full integration efforts could be accomplished by combining various toolkits or individual tools via a framework or by using a workbench (fully integrated lifecycle tool).

Vendors tend to associate the term I-CASE with a workbench tool although some use it when referring to a framework. Therefore, it is imperative to establish a common nomenclature for tools within this category. For purposes of this taxonomy, the term I-CASE or workbench is used to define a tool which provides an integrated set of tools for full lifecycle support. An example of this type of tool is Information Engineering Facility (IEF) from Texas Instruments.⁸ It should be noted that few I-CASE tools today actually provide complete lifecycle coverage. Thus, the taxonomy can be used to delineate which areas are not supported by a workbench.

The term framework was used to describe a tool with an integration architecture enabling users to assemble

⁸ IEF is one of the tools surveyed by the author.

various tools as components of a fully integrated toolset. Industry publications have adopted the term C-CASE (component CASE) for describing this category. For taxonomical purposes, C-CASE or framework is used to describe a tool which provides an open type architecture for assembling tools to achieve full lifecycle support. C-CASE is the direction in which the industry appears to be moving. Some C-CASE tools even provide heterogeneous support which allows the tool to operate on and across multiple hardware and software systems from different manufacturers. [Ref. 13:p. 11]

There are some CASE tools which may integrate with various toolsets, but focus on providing support for a particular aspect of the development cycle such as configuration management or testing. These tools are referred to as "power tools". Although they integrate with various toolsets, they do not provide integration for other tools. They are designed to fit the architecture of the tools they support. An example of this type of tool is CCC (Change and Configuration Control) from Softool Corporation. CCC actually supports a number of the leading tools on the market by providing complete change control and automated configuration management for the entire software development lifecycle. This taxonomy will refer to these tools as power tools or P-CASE.

There is one final integration category of CASE tools: Those that don't integrate with other tools. The

majority of CASE tools today fall into this category. These tools are referred to as designated CASE or D-CASE. Stand-alone CASE tools days are numbered as the trend towards fully integrated environments accelerates. The taxonomy adopts the term D-CASE for tools which fall in this category.

c. Application Areas

"CASE tools are designed to support the development of different types of software" [Ref. 27:p. 425]. Tools can be divided into two major application categories: those designed for Information Systems and (i.e., MIS/DP business applications, such as on-line information systems, order entry transaction systems and traditional data processing) and those designed for Aerospace, Defense and Engineering (ADE) Systems (i.e., engineering and scientific analysis software, process and device control software, etc..). Figure 4-2 depicts these categories further subdivided into subcategories. [Ref. 3:p. x]

Information Systems (MIS/DP)

- On-Line Systems
- Mainframe Systems
- Distributed Systems
- Batch Data Processing and Reporting Systems

Engineering Systems (ADE)

- Analytical/CAD
- Real-Time
- Non-Hardware Specific
- Embedded Systems

Figure 4-2 CASE Application Areas

There are important differences between these groups. On-line information systems on mainframes rely on and must be compatible with resident database management and timesharing facilities. Information systems distributed operating on personal computers require network protocols and data integrity management. Complex processing logic, robust report generation and job control capabilities are required to support batch systems.

Engineering systems involve highly complex operations. Analytical/CAD systems require complex mathematical functions and the ability to handle symbolic logic. Real-time systems typically involve critical, high-speed timing requirements and tend to have complicated control and processing requirements [Ref.29:p. 70]. As a result, they need special constructs to model control behavior, methods to describe multi-tasking and synchronization and facilities supporting performance analysis, rapid prototyping and system simulation. Embedded systems not only require these capabilities, but must also have ways to define close couplings with the target hardware environment. [Ref. 3:p. xi]

Due to the differences in software applications, tools are becoming more specialized by trying to match their design representations and capabilities to the specific requirements of the application domain. Therefore, application areas become a distinctive way of categorizing certain aspects of CASE tools.

3. Attributes

Categorizing and assigning a tool to a particular development phase is not sufficient for classification efforts. In addition to the lifecycle framework, description mechanisms are needed to further define the fit and support provided by the tool. The attributes described in figures 4-3 and 4-4 are provided to help define the full functionality (and limitations) of a tool.

The list of attributes is by no means comprehensive. It reflects those attributes the author considered the most important attributes for initial consideration. The attributes provided are meant as skeletal elements to be used and enhanced by organizations to flesh out a tool. The exact fit of a tool can be determined by applying (or omitting) the appropriate attributes and qualifying attributes as needed. Moreover, organizations can add or delete attributes as needed.

4. Employment

The taxonomy is provided to aid in the development of candidate tool selection lists to augment the tool evaluation process outlined in chapter V. It is designed so that organizations can quickly classify a tool and discern its capabilities and limitations by applying and qualifying the various attributes associated with the tool. Moreover, organizations can expand the attributes to include new or

Code Generation: Tool can generate some programming language from analysis and design representation.

Configuration Management: Tool maintains histories of document versions and configurations of documents.

Design: Tool depicts the module structure of a program being designed either in text (structured English, program design language) or graphically in structure charts or modular block diagrams.

Documentation Support: Tools that provide for the extraction and formatting of the contents of the project database. Others go further to provide standard reports, report generators and templates to meet certain standards (i.e., DoD STD-2167A) with interfaces to technical publishing systems (i.e., Interleaf, Framemaker, etc...).

Fourth Generation Language (4GL): Tool contains a high level language providing database access facilities.

Hardware Systems Supported: Specific hardware systems supported by the tool (i.e., mainframe (IBM etc.), mini-computer (VAX etc.), Workstation (Apollo, DEC, HP, Sun, etc.), PC (IBM, Compaq, etc.), Apple (Macintosh, etc.), other, etc...).

Languages Supported: Specific languages supported by the tool (i.e., Ada, Atlas, C, C++, CMS, Cobol, Jovial, Fortran, Pascal, PL1, etc...).

Lifecycle Supported: Specific lifecycles supported by tool, if any (i.e., Waterfall, Evolutionary, Transform, Spiral, etc...).

Methodology/Diagramming Technique Supported: Specific methodologies supported by the tool, if any (i.e., Bachman, Chen, Curtice/Jones, Customizable, Gane-Sarson, Hatley/Boeing, Hatley/Pirbhai, Information Engineering (Martin), Information Engineering (Finklestein), Jackson, McCabe, Merise, Page/Jones, Petrinets, Proprietary, SADT, Schlaer/Mellor, Ward/Mellor, Warnier-Orr, Yourdan-Demarco, etc...).

Multi-user: Multi-user data access (concurrent data access by multiple users).

Networkable: Tool can operate in a network environment.

Performance Analysis: Tools that measure the complexity software, generate static or dynamic statistics of a program's performance, or analyzes the structure of a program.

Figure 4-3 Taxonomy Attributes

specific capabilities required to satisfy their individual needs. Appendix A contains a sample form prepared by the author to demonstrate the application of the proposed taxonomy

Project Management: Tool provides or reports project management information including number of processes, allocation of work, completion status and, in some cases, schedules, budgets and project dependencies.

Prototyping: Tool provides ability to develop screen or report prototypes and generate appropriate code, or provides capability to rapidly develop algorithms and test the code.

Requirements: Tools providing either text or graphic capability to generate or analyze requirements. If graphic, a popular structured analysis technique is used (i.e., Yourdan/Demarco).

Reverse Engineering: Tool is capable of reading source code or database schema and creating the documentation and design representations (structure charts, entity relationship diagrams, module block diagrams, calling trees, etc...) necessary for enhancing and maintaining the code at the analysis and design level. Some tools allow for new code to be generated from the modified designs.

Simulation: Same as prototyping except that the ability to simulate the behavior of the prototype system is also provided.

Software Systems Supported: Specific operating systems supported by the tool [i.e., mainframe (VM/CMS, etc.), PC (MS-DOS 3.1, 3.2, 4.0, OS-2, etc.), Workstation (Sun 3.5, 4.0 etc...)].

Strategic Planning: Tool is capable of creating an enterprise model or is used to generate a strategic systems plan.

Testing: Tool provides the capability to generate test beds or test suites from the source code. Also includes capability to assist in system integration testing in the target hardware environment.

Testing & Maintenance: Tool provides the capability to generate test plans and test data and manage the test data.

Traceability of Requirements: Tool can track and report the impact of change between documents or trace the development of a requirement throughout the system so compliance and completeness checks are possible.

Figure 4-4 Taxonomy Attributes

along with additional comments and suggestions.

C. SURVEYS

Appendix B contains the taxonomy sheets for several tools surveyed by the author. The tool information is based on

responses from questionnaires sent to the vendors and numerous follow-ups between the author and technical support personnel. The tools selected provide an overall representative sample of CASE tools available today.

D. SUMMARY

This chapter identified a general taxonomy for CASE tools. The taxonomy is provided to help organizations quickly and conveniently develop candidate tool lists for supporting the evaluation process identified in the next chapter. It is designed so that organizations can tailor the description mechanisms (attributes) to fit their own organizational needs. Appendix C contains a blank taxonomy form for use by individual organizations. The evaluation process identified in the following chapter will demonstrate the role of the taxonomy within the tool evaluation process.

V. TOOL EVALUATION PROCESS

There are no formal standards established for the evaluation of CASE tools. In fact, "there are no easy or prescriptive solutions for the evaluation and selection of CASE tools" [Ref. 30:p. 8]. Little if any comprehensive guidelines have been published regarding the evaluation of CASE tools. The most notable comprehensive effort in this area, "A Guide to the Classification and Assessment of Software Engineering Tools", was published by the Software Engineering Institute at Carnegie-Mellon University in August 1987 [Ref. 31]. Another comprehensive effort is currently under development by the Software Technology Support Center (STSC) at Hill Air Force Base in Ogden, Utah. The STSC is a recently established organization in the Air Force whose charter is to "act as central focal point for proactive management of MCCR [Mission Critical Computer Resources] support tools and environments" [Ref. 32:p. 1]. The STSC has proposed the development and adoption of a Software Tool Evaluation Model (STEM) to act as a yardstick to serve as an unbiased model to which software tools, especially CASE, can be compared. The guidelines provided in this chapter are based on the SEI guide. The sectional discussions for both

the Evaluation and Assessments sections have been paraphrased from the guide.

A. PREFACE

As noted in chapter II CASE is no longer just a tool or a group of tools providing analysis, design and programming support for developing software. CASE has evolved into a support environment spanning the entire software engineering lifecycle providing support to the entire engineering team (i.e., managers, analysts, designers, maintainers, etc...) for overall product development [Ref. 12:p. 20]. As such, there is no particular set of requirements which will apply to all organizations, nor can an organization look only at general criteria to evaluate CASE tools. Case succinctly points out:

It is necessary to define the specific requirements for your organization, the processes and information flows that currently exist, and then finally to identify the feature set that will optimize the fit of a specific CASE tool to your environment. [Ref. 30:p. 8]

B. EVALUATION CRITERIA

Classifying a tool does not appraise it. As noted in the previous chapter, a classification scheme provides an indication of what a tool might do and where it could be used, whereas an evaluation attempts to assess how well the tool does it's job from the evaluator's perspective. As such, the evaluation process is inherently subjective since users have

different requirements, work in different environments, and have different perceptions of how tools ought to work. Nonetheless, many questions a user might ask can be standardized, with the understanding that different users will interpret the answers in different ways and affix their own measures of importance to them. Appendix D contains a list of standardized questions provided by SEI to form the basis for the tool assessment process.

C. ASSESSMENT PROCESS

The establishment of formal criteria for evaluation is not enough. Criteria in and of itself is similar to a tool in this respect. It has no inherent value. It derives its value through its application by a particular individual or organization. Since users are varied, what is appropriate to one user, whether an individual or an organization, may be inappropriate to another user. Therefore, the process of evaluating or assessing a tool must be accomplished by the organization that intends to acquire the tool. The SEI effort identified a four step assessment process:

Perform a needs analysis.

Perform an analysis of the existing environment.

Develop a list of candidate tools and acquire descriptions of these tools.

Apply assessment criteria and select a tool for use.

1. Needs Analysis

The initial step in the assessment of a tool is to decide the purpose for which the tool will be used. Tools derive their value from their ability to do something such as perform a function, save time, save labor, save money, or make something possible that is otherwise difficult or not possible. Their capabilities must be relevant to the acquiring organization and must bring utility to that organization. A tool may require specific features to be appropriate for an organization: generate ADA code; generate 2167A documentation; reuse code; reverse engineering. It must contribute to a process controlled by a method. The following points should be considered:

What is the relevant model of software development?

What major tasks does that process require?

Which tasks should be performed or assisted by automated tools?

Which of those tasks currently lack adequate tool support?

What is the estimated benefit to be obtained from specific new tools?

"The organization must clearly understand its software development process, methods and management, and the needs they imply before deciding to acquire tools." [Ref. 31:p. 31].

2. Environment Analysis

The next step in the assessment process is to conduct an analysis of the environment in which the tool will be used. It can normally be performed while the needs analysis is conducted. Tools do not operate in a pristine environment. The success of a tool is determined by how well it fits the environment of a specific organization. Since each organization is different, the decision makers within an organization cope with their own environmental constraints.

Constraints take many forms but, can normally be classified in several distinct areas: economics, time, personnel, vendor relations, etc.... Understanding the environment and the impact of the constraints within it is crucial to the environmental analysis. Equally crucial is to understand there are two ways to deal with constraints: "...live with them or change them." [Ref. 31:p. 32]

Identifying constraints is not enough. The environmental analysis must also identify those constraints which can be eliminated or modified as well as the tradeoffs between them. Figure 5-1 contains the questions which organizations should consider when performing the environmental analysis according to the SEI guide.

3. Develop Candidate List

After it's needs have been identified, an organization should develop a list of candidate tools that

-
1. Is the organization open to change? Have changes occurred in the past? Have there been successes or failures?
 2. Have there been lessons learned from past successes or failures? Do the lessons support introduction of the tool?
 3. Can the organization afford to buy the tool easily, or will the purchase price place extreme pressure on learners for instant success?
 4. Is the investment in the tool so large that it will be difficult to dislodge in the future?
 5. Is there a plan to introduce the tool? Does everyone understand the plan? Are goals, objectives, benefits, risks and milestones clear to all?
 6. Is there an agreed upon way to determine progress in use of the tool?
 7. Is management planning to reinforce progress and initially hold back negative judgement?
 8. Is the tool sponsored by a champion -- someone able and willing to serve as sponsor and focal point, and to monitor and encourage progress?
 9. Is training scheduled to allow real use of the tool shortly after completion of training?
 10. Will those who need to learn the tool be able to do so in a low-pressure environment?
 11. Will learners have adequate access to the tool during the learning period?
 12. Will learners have pilot projects on which to practice the use of the tool?
 13. Will learners have time to experiment?
 14. Has a case been made for increasing benefit over time as users become acquainted with the tool and increasingly exploit its power?

Figure 5-1 Organizational Environmental Analysis Questions

might satisfy those needs. Recognition of the value of CASE has risen sharply in recent years. As a result, many new vendors have entered the expanding market with a variety of tools. Information on available tools can be obtained from

trade publications, trade shows, and technical journals. One such publication is CASE Outlook by the CASE Consulting Group which was instrumental in supplying key information for this thesis. There are also several governmental organizations available for supplying key information on CASE tools. The STSC which was mentioned previously and the Federal Software Management Support Center (FSMFC) of the General Services Administration.

The FSMSC has an established database of CASE tools and the federal employees who use them. The FSMSC contains information such as the tool, its vendor, a functional description and its cost. More importantly, it can provide the user information so that callers can contact the users themselves to discuss their experiences with a tool.

The list of potential tools should be developed as close to the date of selection as possible. New vendors, new tools, and major product upgrades occur on a regular basis. Hence, product information is quickly outdated. Timely product information is critical when deciding on the most appropriate, available tool.

Obviously, the list should only contain tools that appear appropriate to the discerned need and the organizational environment. Tools that clearly do not meet the need or cannot function within the existing environment should be excluded. The classification scheme outlined in Chapter IV provides the means to quickly discern between

various tools. The user can quickly capture and organize data on existing tools and target those tools deemed most appropriate. Appendix C contains a blank classification form for such use.

The candidate list should not just focus on individual tools. One tool may not be able to satisfy all requirements whereas a set of related, compatible tools might jointly suffice. It is up to the organization to determine the importance of having all or most tools produced by the same vendor or with the same characteristics. "There are great advantages in acquiring a tool set with a consistent philosophy - the burden of acquisition, training, support, and use is substantially less". [Ref. 31:p. 33]

4. Apply Criteria and Select

Once the candidate tools have been identified, each tool must be analyzed to determine it's fit to the organization. The application of a set of evaluation criteria to each tool is the final step in the assessment process. The SEI approach suggested the following four phases:

- Establish evaluative criteria
- Define a specific experiment
- Execute the experiment
- Analyze the results

a. Establish Evaluative Criteria

The criteria listed in the previous sections identify attributes that should generally apply across a wide range of tools. The criteria are straightforward and unweighted. Each user or organization must review the checklist and make it's own estimate of their relative importance. For instance, some organizations may prefer tools that are easy to learn if they cannot afford the time and cost of expensive training while others might be willing to incur significant training costs to acquire much more powerful tools.

The resulting checklist must be augmented with the results of both the organization's needs analysis and environmental analysis before a final selection is made. The criteria identified must be listed and ranked in the order of importance. This list serves as the basis for the next phase.

b. Define a Specific Experiment

The prioritized list identified in the previous phase must be translated into tests to be performed on each of the potential (candidate) tools. Each question or criteria on the list must be supported by one or more specific tests tailored to the individual tool being evaluated. Each test must identify exactly what is to be performed and under what set of initial conditions. Each test should also detail

exactly what data is to be collected and the quantities that should be measured to answer the underlying questions.

c. *Execute the Experiment*

It is vital that the tests identified be conducted through hands-on use of the tool. Personnel should not rely solely on product literature or documentation. Even though many questions can be answered by reviewing the literature, it can occasionally be misleading or misinterpreted. The tests identified should be sequenced according to the prioritized list of criteria documented in the first phase. Unacceptable results on early tests indicate the tool will not satisfy the organization's critical needs. Poor or unacceptable early test results can be used as a basis for shortening the testing process. By eliminating poor performing tools early on, organization's can focus their efforts on the most promising tools.

The end product of this phase should be a transcript of the execution of the experiment and the measurements and answers to the criteria that were detailed in the previous phase.

d. *Analyze the Results*

The final phase consists of analyzing the data collected from the experiments. Each tool should be analyzed to determine how well it satisfies each of the criteria. Criteria ranked the highest should receive special attention.

After the criteria have been applied, the decision process begins. The results usually indicate that no tool is a perfect fit for the particular organization. "The final decision must be based on the judgement of those in the organization who will receive the most benefit (or harm) from the tool selection." [Ref. 31:p. 34] The impact of introducing a particular tool must also be considered since it's use can and should alter the software engineering environment.

The assessment criteria will not provide a recipe for absolute success in selecting the most appropriate, useful tool. It is intended as an aid to the selection process. The assessment must be a careful, meticulous process culminated by the planned, monitored introduction and use of the tool selected to enhance its chances of acceptance and use.

D. SUMMARY

There are few guides available for evaluating CASE tools. The SEI guide provides a complete tool evaluation process. It provides a generic checklist which can be used across a wide variety of tools and can be tailored to accommodate specific organizational needs. In addition to the checklist, the SEI guide identifies an assessment process for conducting the evaluation. The guide emphasizes that simply using a tool is not enough. The tool must not only fit the application and needs, but fit the organization as well. It is also vital that organizations consider the impact that introducing the

tool will have on the organization. The benefits from tools do not come without costs. It takes a considerable commitment to introduce a tool successfully in an organization. "All the activities from selection to training to tool set evolution will affect an organization's ability to effectively use the tool and reap the maximum benefit possible from it." [Ref. 31:p. 36]

The tool evaluation checklist identified along with the DoD Std-2167A impact areas identified in chapter III and the critical areas defined in chapter IV serve as the basis for the evaluations contained in the following chapter. Time and scope limitations prevent a complete in-depth analysis of each tool, therefore, the author must select specific emphasis areas. Figure 5-2 depicts the specific areas selected by the author for emphasis. The areas selected reflect those the author considered most appropriate for the target audience.

Methodology Supported
Hardware/Operating System Requirements
Installation
Documentation
Interface to Other Products
Multi-user Access
Network Support
DoD-STD-2167A Support
User-Interface
Traceability of Requirements
Dictionary/Repository
Prototyping
Consistency/Completeness Checking
Training Support
Diagramming/Graphic Facilities

Figure 5-2 Tool Evaluation Areas

VI. Tool Evaluations

This chapter contains the personal evaluations conducted on three commercially available tools: Excelerator/IS 1.9, Software through Pictures (StP) 4.2A, and Engineering and Project-management Oriented Support System (EPOS) 4.0. The tools selected provide a representative sample of CASE: 2 PC, 1 workstation; 1 P-CASE, 1 C-CASE, 1 I-CASE. Time and scope limitations prevent a complete in-depth analysis of each tool. Figure 6-1 contains the specific areas emphasized (as identified in the previous chapter) by the author. The goal of these evaluations is not to develop a software product, but to attempt to identify the major capabilities and limitations of each tool for the target audience. Each evaluation follows the same general approach. A sample textbook software project served as a common model to support each evaluation. Some information, such as the interface to other products, is based solely on the documentation provided and will be identified accordingly. No endorsement of any tool is intended.

Methodology Supported
Hardware/Operating System Requirements
Installation
Documentation
Interface to Other Products
Multi-user Access
Network Support
DoD-STD-2167A Support
User-Interface
Traceability of Requirements
Dictionary/Repository
Prototyping
Consistency/Completeness Checking
Training Support
Diagramming/Graphic Facilities

Figure 6-1 Tool Evaluation Areas

A. EXCELERATOR/IS 1.9 OF INDEX TECHNOLOGY CORPORATION

1. Hardware/Operating System Evaluated On

The tool was evaluated on a 386 clone (20 Megahertz) with an 80 megabyte hard drive and a VGA monitor using MS-DOS 3.3 operating system. A Logitech 3 button mouse was used to provide mouse support. No compatibility problems with any of the hardware nor the operating system were observed.

2. Tool Description

Excelsator/IS is an Analysis and Design tool oriented towards business applications. It contains an integrated set of analysis and design tools focusing on automating the early phases of system development. It concentrates on analyzing and defining the application problem and creating the system specification. Excelsator/IS 1.9 also includes project management capabilities.

3. Methodology Supported

Excelerator/IS is designed to support structured methodologies. To take advantage of it's full capabilities, users need to use a structured methodology or approach. Excelerator supports a wide range of methodologies. It can even be tailored via Customizer to support an organization's own "home-grown" approach or to link with other development tools. Customizer is identified in the section: Interface to Other Products.

Excelerator combines the Yourdan/Demarco Structured Analysis methodology with data modeling and structured design methodologies. It supports both Yourdan and Gane/Sarson notation for data-flow diagrams. It also supports Ward & Mellor notation for state, control, and event modeling. Entity-relationship diagrams are available for data modeling using both Chen and Merise notation. Constantine structure charts and Jackson structure diagrams are provided to help analyze process logic.

4. Hardware/Operating Systems Requirements

The hardware support for Excelerator/IS 1.9 is standard MS-DOS personal computers. Figure 6-2 contains the specific hardware and operating systems requirements for Excelerator/IS 1.9. The tool also supports the following workstation (32-bit environments): VAXstation 2000 family and Apollo DN3000.

PC Hardware Requirements

Disk Space

8 MB for Excelerator Program
1 MB for temporary files created during execution
3.2 MB for data storage (per average 1.9 project)
Recommended Amount: 20 MB -- 30 MB for large projects

Memory

Minimum of 459K of conventional memory
Recommended Amount: 640K

Graphics board (must be 100% compatible)

IBM EGA, IBM VGA, Hercules Graphics Card

Mouse

Driver must be compatible with MS MOUSE.COM or MOUSE.SYS
Version 6.1 or above

Printer Support

Epson FX100, LQ1500
Hewlett-Packard HP7475A, PH7470A, Laserjet+, Laserjet II
IBM 80 CPS Graphics, Proprinter, Proprinter 24 Family
Toshiba P1350, P1351, P351
Texas Instruments TI 855

Operating System Requirements

PC-DOS or MS-DOS Version 3.1 or higher; OS2 in DOS session
BIOS must be IBM XT, AT, or PS/2 compatible

Specific PC Systems Supported:

AT&T 6300	IBM Personal System/2
COMPAQ III	IBM PC/AT, PC/XT
COMPAQ Plus	IBM 3270 PC/AT
COMPAQ Portable 286	HP VECTRA

**Figure 6-2 Hardware and Operating System Requirements
for Excelerator/IS 1.9**

5. Installation

The installation process was straightforward and relatively easy. Tool and documentation was delivered as a complete package. The package included a Release Notes guide which especially helpful regarding product changes not incorporated in the original documentation especially installation sensitive information. The installation was effected by loading the installation disk, executing a batch

file, and then following the prompts. The procedure automatically modified the autoexec.bat and config.sys files to enable DOS to run the software. The use of an automated installation program greatly simplified the entire process.

The only negative incident encountered pertained to the installation of the block security device used to copy protect the software. Neither the installation manual nor the installation program indicated when or exactly where to install the device. The Release Notes guide did mention the device was a new enhancement which replaced the key-diskette approach previously used and could be attached to either a parallel or serial port.

The initial execution attempt failed with the block device attached to the LPT-2 port of the evaluation computer. The matter was easily and quickly resolved by communicating with company representatives via the hotline support number provided in the installation manual. They recommended attaching the device to LPT-1 and attaching the printer cable to the rear of the device which immediately resolved the problem.

6. Documentation

The documentation provided is extensive. The documentation set consisted of a Tutorial, an Application Guide, a two-volume Reference Guide (Facilities & Functions, Data & Reports), a Quick Reference Card and a function key

template. The set also included a Release Notes and Enhancements document describing new features incorporated along with a Services booklet delineating the training services available to support the introduction and application of the tool.

The tutorial lived up to its billing. It was very general and easy to follow. It concentrated on exploring the basic features and learning the mechanics of the tool. Goals and tasks provided within each section were helpful as well. Hints on how best to utilize certain features were also offered and did prove to help in several areas.

The application guide is an overview of the analysis and design process using a case study. It provides details for creating and retrieving data along with suggestions for organizing the project emphasizing structured techniques. The guide follows a structured approach for the analysis (logical) portion, but only offers suggestions on how best to use the tool to support the design phase. The appendix within the guide indicated all the documentation was available, but it lacked the data flow diagrams (DFD) for DFD 3.0, DFD 4.0 and DFD 5.0 and the primitive process specifications (PPS) for PPS 3.0 and PPS 4.0 as specified in the Document Graph for the specification. As a novice user, a complete approach would be preferable. A finely detailed very precise cookbook or exercise approach would greatly benefit the first-time CASE user.

The combination of the tutorial and the application guide did mitigate the initial barrage of features and helped to focus on application techniques. However, the documentation seems to be geared towards the experienced user. The first-time or novice user is easily overwhelmed by the variety and extensiveness of the features available.

The only serious shortfall regarding documentation involves the tool itself. The on-line help provided is very limited. All on-line help is limited to one line. The Quick Reference Card and function key template provided help to a degree, but the user is often forced to refer to the hard copy documentation whenever a question arises.

7. Interface to Other Products

Documentation provided with the tool indicates Excelerator can interface with a variety of other products to provide an integrated development environment. Specific products include:

' Customizer: An add on product from Index Technology with utilities that allow users to modify the System Dictionary and System Forms Library files. Users can add entity types and attributes, change menu structure, include new graphs and new graph objects and more. It includes XL/Programmer Interface which allows C language programs to use it's function library to access Excelerator's project dictionary and graph files.

PC Prism: A planning tool from Index Technology which is used for strategic information systems planning and enterprise modeling. Components of the planning model developed within PC/Prism can be exported into Excelerator to aid analysis efforts. Excelerator can also export data to PC/Prism to help planners model a future system by using an existing one.

XL/Interface for MICRO Focus Workbench product: A customized version of Excelerator/IS that provides ability to design and code a complete COBOL source program.

XL/Interface for ABT's Project Workbench: A customized version of Excelerator/IS that provides ability to integrate Excelerator/IS's tools for system analysis and design with the Project Workbench system's tools for project management.

Excelerator for IBM's CSP/AD: An enhanced version of Excelerator/IS that provides modeling and analysis tools specifically tailored to support IBM's CSP/AD development environment. This version will eventually provide support to IBM's new AD/Cycle.

Excelerator for IBM's DB2: An enhanced version of Excelerator/IS that supports physical database design in DB2. This version includes a capability to link between Excelerator and DB2, a mainframe relational database from IBM, to facilitate physical database design and implementation.

8. Multi-user Support

Excelerator can support multiple projects and multiple users on a single workstation. A Project Manager feature creates projects, assigns users to project tasks, and assigns access levels. When a user logs on to Excelerator, he/she must specify which project to work on. Once logged on, the user works only with the data associated with that particular product and only that data for which access is authorized. **It does not support simultaneous access by multiple users.** This aspect will be fully discussed in the next section.

9. Network Support

Excelerator operates in a network environment and supports various network products. Figure 6-3 depicts the network software systems supported by the tool. However, as noted above, the tool only allows one user access at a time. To support multiple users, multiple copies of the tool must be loaded and configured for each user on the file server. Users working on the same project can read the central project dictionary, if granted access, but must have their own copy of the project data to work with in order to update the contents. With access, users can download current project data and upload data to the central dictionary. The project manager is tasked with the responsibility to ensure users maintain the integrity of all shared data.

Network Support

IBM PC LAN	IBM Token Ring
3COM3+	3COM 3C501
Novell Advanced Netware	3COM 3C501, IBM Token Ring
Novell ELS Netware 286	3COM 3C501, IBM Token Ring
AT&T Starlan	AT&T Starlan

Note: Requires individual system files to be loaded on file server for each user

Figure 6-3 Network Systems Supported by Excelerator/IS
1.9

The installation guide recommends installing a copy of Excelerator on each individual node to optimize the tool's performance. The advantage of operating a tool in a network environment lies in the ability to share output devices, facilitate the sharing of project data by transferring data via the network to the central dictionary, and the ability to back up all project files from a central location. However, this advantage is diminished by having to manage multiple copies of the program in various locations and needing a block security device for each copy of the tool.

10. DoD STD-2167A Support

Excelerator 1.9 does not directly support 2167A requirements. However, it does provide support via an add-on product, XL/Doc. XL/Doc allows Excelerator to conform to government standards by automatically generating in prescribed formats or scripts. Conversations with marketing representatives indicated scripts, although not which ones,

currently available adhere to DoD STD-2167A standards with additional scripts scheduled to be available in new releases of XL/Doc.

11. User-Interface

Excelerator/IS utilizes a menu-driven interface. Navigation through the menu hierarchy was relatively easy. The use of a mouse greatly simplified the navigation process. Most notable feature the author appreciated was not being buried at lower levels. Most menu operations were limited to three levels.

Another important aspect of the interface, is that the menu formats are consistent throughout. All menu selection screens, data entry screens, or diagrammatic screens follow the same general pattern. Familiarity is enhanced by using one section of the tool and finding similar operations available within other sections which apply the same logic. For example, graphing knowledge and techniques learned in a particular section, such as Data Flow Diagrams in Graphics, is easily transferrable to Entity Relationships Diagrams.

Excelerator/IS incorporates the use of colors to enhance the interface as well. The tool used color most effectively in the diagrammatic section when employing graphics. Once an entity, such as a dataflow or a process on a dataflow diagram, is described to the dictionary or a process is exploded to a lower level, it changes to another

color indicating such action has been accomplished. The change in color provides a visual reference to remind the user that certain operations have or have not been accomplished. The tool provides a variety of colors to select from.

Excelerator incorporates the use of a mouse as well. With the aid of the mouse, navigation between menus and selections is quick and easy. The Graphics portion of the tool relies exclusively on the use of the mouse to accomplish diagramming efforts.

12. Traceability of Requirements

Version 1.9 includes a new enhancement which now allows the tool to track requirements. It accomplishes tracking by using two new entity types -- User Requirement (URQ) and Engineering Requirement (ERQ) -- which allow a user to construct a requirements database for each project or proposed system. URQ's specify "what" a system must do while ERQ's are used to specify "how" the system implements URQ's.

A user can take a written requirements specification and assign each unique requirement to a particular URQ. These individual requirements can be incorporated within the description of the various dataflows, stores, data entities, data elements, modules or processes via a "Satisfies Requirement" field. If used in conjunction with ERQ's, users can specify technical details associated with the particular requirement to facilitate tracking in the design phase as

well. As a result, a particular requirement can be traced throughout the entire requirements and design specifications. The guide even recommends using descriptive names instead of numbers for the URQ's. Then, if a source document is modified and renumbered, there is no need to rename (renumber) URQ's wherever they were used.

URQ's/ERQ's can be used with additional entities provided within the dictionary to facilitate tracking as well. Issues and Notes are entities which can be used by analysts to record concerns that might affect the deadline schedule or document the non-support of a requirement. Issues and Notes can be referenced from the description screen of each entity it affects. Figure 6-4 illustrates a User Requirement for a process along with associated Issues and Notes. Users can also generate Issue reports to maintain an current progress status or current status of project requirement support.

13. Dictionary/Repository

The dictionary, XLDictionary, organizes information by projects. The project data is organized in sets of items with the same characteristics known as entity types (i.e. dataflow, process). Each entity is supported by a description screen which lists the attributes that belong to it. Figure 6-5 depicts the format for a dataflow. Associations between entities are another important characteristic maintained by the dictionary.

User Requirement		PROCESS ORDERS	
Alternate Name			
Short Description			
THE SUBSCRIPTION SYSTEM MUST BE ABLE TO MANAGE BOTH MAIL-IN AND TELEPHONE SUBSCRIPTIONS AND MAINTAIN INVENTORY OF EACH			
Priority			
Contains:		Has Associated:	
Type	Name	Type	Name
		REF	REQR'S SPEC PARA 3.0
		ISS	INSUFFICIENT DETAIL; NEED DATA
		NTE	CONTACT SUBSCRIPTION DEPT MGR
			PgDn

Figure 6-4 Excelerator/IS 1.9 Sample User Requirement

The dictionary treats an association between two entities, such as a dataflow to a particular record or element, as a logical relationship. It tracks and cross-references the relationship of every entity defined to other entities within the project. Relationships are established in two ways: by adding graphic components to a graph or via the description screen of another entity. Relationships are an important component of the tool's consistency and completeness checking ability.

Graphic relationships are automatically created by linking objects on the diagram. Relationships entered via description screens are established via an "explodes to" field within the entity description screen. Exploding allows an entity to be described in greater detail by linking it to another entity. The dictionary enforces logical relationships by only allowing the entity to explode to other appropriate entities. The dictionary supports over 50 different entity types and can track over 1000 relationships.

Input to the dictionary can be accomplished in several ways: by defining components as they are added to a graph, by entering records and elements directly into the dictionary or by describing elements to the dictionary via the Screen Design facility. Thus, the dictionary can be populated or modified without entering the graphics facility.

The multiple input capability is accomplished by using a function available within the dictionary called "browse." Browsing allows navigation between related entities with a single keystroke without having to go through any menus. For example, while inputting or updating a record, a user can enter its elements directly into the dictionary by selecting browse and defining each element as it is input in the record. Another important use of browse is to track requirements. It is very easy to pop from a process to a URQ and from URQ to an Issue or Note associated with it to verify a requirement was addressed or check pending issues.

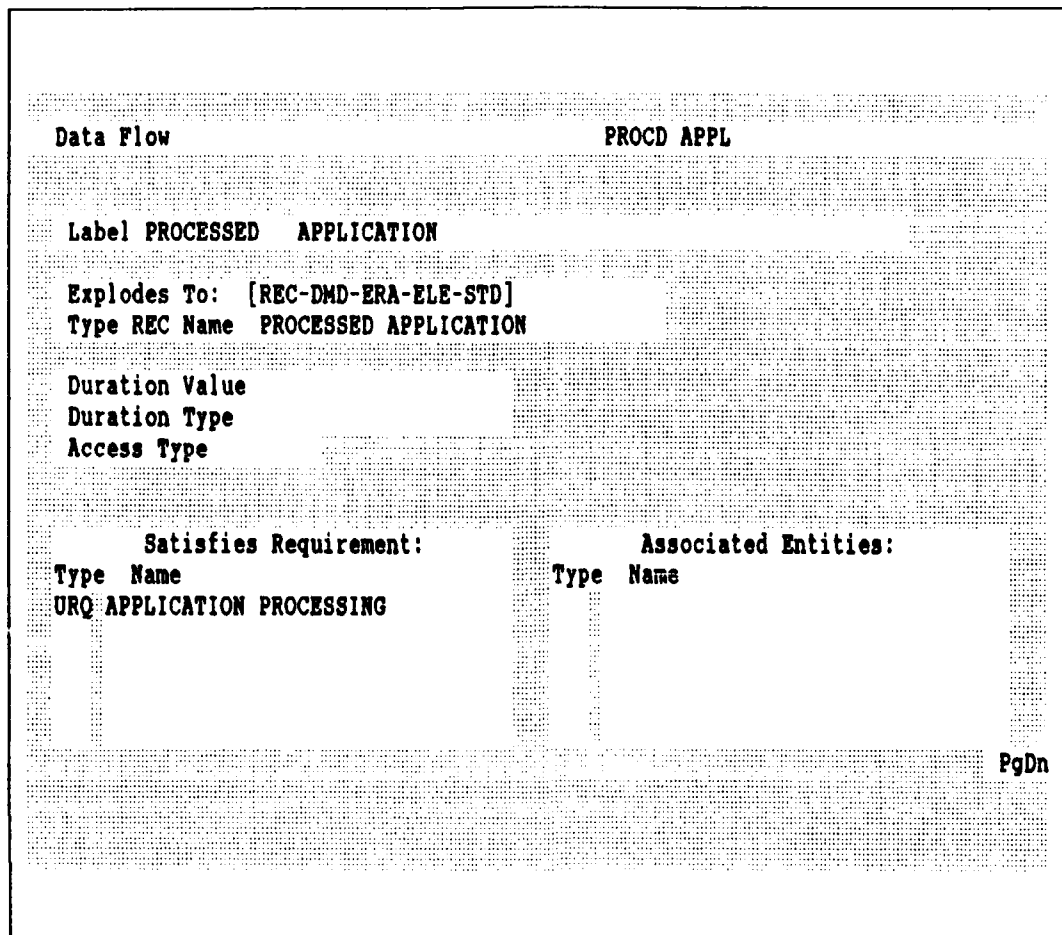


Figure 6-5 Excelerator/IS 1.9 Sample Dataflow

14. Prototyping

Excelerator's "Screens and Reports" facilities provide capability to prototype data entry screens and report output formats up to 132 columns wide. Both facilities offer full screen editing and utilize the mouse to navigate about. Both facilities include a field command feature which is particularly useful in verifying the information on the layout is consistent with the data stored in the dictionary. When specifying a field location on the screen, the tool queries

for the name of the data element it is associated with. If in the dictionary, the specific information is retrieved and automatically formats the screen entry. If not, the element can be described into the dictionary directly (by browsing from the screen to the actual data entry) thus ensuring the items are consistent. Multiple input screens can even be chained together to indicate their sequence.

The screen design facility includes an inspect option to test the screen design. After saving the screen and selecting inspect, data can be input to check the layout, verify field lengths, demo help messages and verify the chaining. This is a very useful feature for communicating with the actual user of the system.

Once verified, the screens and reports can be converted into a compilable data structure. Excelerator can generate the screen and report designs into a data map which is a programming language description of the components and their structure. The tool can generate BASIC, C, COBOL or PL/1. The outputs can be converted into ASCII and transferred to the target system as well via an Interface File option.

15. Consistency/Completeness Checking

Several techniques for ensuring consistency and completeness have already been described in earlier sections. Relationships such as those created via the "explodes to" field within entity description screens and by defining

entities within entities using the "browse" function within the dictionary or from the screen design facility are prime examples. The explosion option is particularly useful when constructing DFD's. If a Process is exploded to another level, the tool will automatically bring down the dataflows associated with the Process to remind the user what flows are associated with the Process.

One of the more important consistency aspects of the "explodes to" involves the transition from the logical representation (DFD's, STD's, etc..) of the project to the physical representations: Structure Charts and Structure Diagrams. Entities such as Processes or PPS on a DFD can be exploded to either of the structured representations used to represent the the physical designs used to implement the activity. The explosion path aids monitoring and analyzing the relationships between logical and physical reviews of the system. Thus, consistency between the logical specification and the physical specification can be maintained.

In addition to the various interactive techniques mentioned, there are formal checking mechanisms provided by the tool. Screen-based relationships can be verified via a "missing entities" function based in the dictionary. Missing entities examines entities that are related to each other via a description screen link. For example, a record might be described, but the elements contained in it were not. Running missing entities on the relationship type "REC contains ELE"

generates a report on the entity record and any undefined elements.

Graphic diagrams have formal checking mechanisms as well. The "Analysis" facility within the tool contains several graph verification options, two of which were particularly useful. The first, "Undescribed Graph Entities" provides a quick check for any entities which might have been identified on a graph, but were not described to the dictionary. The second, "Level Balancing", is a much more powerful option.

"Level Balancing" assesses the consistency of a DFD. Level balancing ensures information, such as a dataflow to or from a process, is not mentioned on one level and ignored on another by comparing two levels at a time. The top level entities are referred to as "parent entities" while the next lower level entities are referred to as "child entities". The tool begins with the DFD specified and examines each process one by one attempting to to balance the parent process's input/output dataflows, signals and prompts with the appropriate explosion entity. A process can explode to another DFD (going to a lower level of detail), a State Transition Diagram or a Primitive Process Specification (PPS). Figure 6-6 contains a level balance report from the sample project. Balancing continues until all levels within a DFD have been checked.

LEVEL NUMBER: 1
PARENT GRAPH NAME: RECORD CLUB SYSTEM OVERVIEW

Graph Object Summary

OBJECT TYPE	I/L	ID OR LABEL	NOT DESCRIBED	CHILD TYPE N/A	CHILD NOT FOUND	IN BALANCE
PROCESS	L	RECORD CLUB SYSTEM	X			
PROCESS	L	SUBSCRIPTION SUBSYSTEM	X			
PROCESS	L	PROMOTION SUBSYSTEM	X			
PROCESS	L	ORDER PROCESSING SUBSYSTEM	X			
PROCESS	L	3.1				X
PROCESS	L	3.2				X
PROCESS	L	3.3				X

Press ESC to exit.

Use arrow keys, PgUp, PgDn, to scroll output.

Figure 6-6 Excelerator/IS 1.9 Sample Project
Consistency Report

These are not all the features offered, but are some of the most useful. The formal mechanisms are easy to use and produce reports which can be sent to either the screen for quick viewing or to a printer for greater review. Although easy to use, the tool takes considerable amount of time to generate some information. Some reports such as a level balance report of the project took up to two or three minutes to process before being output.

16. Training Support

Excelerator has an extensive training support program. Educational and consulting services are available as well as support services.

Public and private training courses are offered for both new and experienced users. Instruction includes theory, techniques and application via hands-on operation using real world exercises based on actual systems. Public courses are

offered at a variety of locations around the country. Private courses can be provided at on-site.

Consulting services also available for Excelerator users. Services range from expert advice and independent analysis to additional manpower support. Strategic planning and evaluation of hardware/software technologies are offered as well.

Support services provided include: 90 day warranty, unlimited hotline support, quarterly newsletter, user conferences and X/L Group Inc., an Excelerator user support group. Extended updates and maintenance plan are offered at extra cost.

Extensive support is available, but most is at user's expense. Organizations can tailor training needs to minimize financial impact. Users can monitor training schedule to attend closest training sites and take advantage of multi-user discount options offered. Bottom line is that extensive support is available, but it can be expensive.

17. Diagramming/Graphic Facilities

Excelerator supports a variety of diagrams and different notations as noted in the Methodology Supported section. Diagramming within the tool does require some effort and experimentation for familiarization which was covered to some degree within the tutorial.

The tool is quite flexible. For instance, users can construct a Context Diagram and then explode to the top level DFD and so on all the way to the Primitive Process Specification (PPS) or users can generate individual DFD's and connect them to the corresponding DFD or PPS. Moreover, the tool does not require the diagram desired to be named. It will list the diagrams available and allow user to select the appropriate one. When updating, users can select an individual DFD to modify without having to traverse the various levels. Therefore, a user can jump in and out of a project almost anywhere in the system.

Other options offer significant flexibility as well. Users can select from three different connecting lines: pipes, straight lines and now curved lines depending on their preference. Mixing different lines on a graph or diagram permits manipulating the layout in a variety of ways. The tool also offers a choice of either "system" or "user" ports for connecting the lines to individual objects. This option allows a user to specify the location of the connection on the object or let the system do it automatically. Much practice is needed with the "user" selection to fine tune the connection on the object which can be frustrating at times. Another frustrating aspect with line drawing involves the movement of objects. If a Process is moved, all connections are moved as well, but the tool tends to maintain the original

connection location which can generate a maze of lines on the diagram forcing the user to reconnect the lines manually.

The layout itself offers a high degree of flexibility. The layout can display up to six pages for a diagrams which exceed one normal page. Individual page size is determined by the printer selected. A "Preview" option allows user to experiment with individual fonts and display their true size for printing options. Unfortunately, the user is only provided a view of the object itself. The user is forced to select and then leave and return to the layout to examine the results. Moreover, the actual page size is not reflected on the screen itself until "Print" is actually selected. Only then can the user see if his format and font selections will fit a particular page. Figure 6-7 depicts the top or system level DFD of the sample project. The figure illustrates the main sub-processes of the sample system being modeled: processing subscriptions for membership, generating monthly promotions for the membership, responding to orders generated from the promotions and issuing orders to the warehouse to satisfy the requisitions.

The tool does provide suggestions which offset the preview limitation to a degree. Such as selecting print and after the page lines appear use "line draw" to mark off the page lines. Thereafter, user can copy the diagram to another name and use it as a format to construct other similar type diagrams.

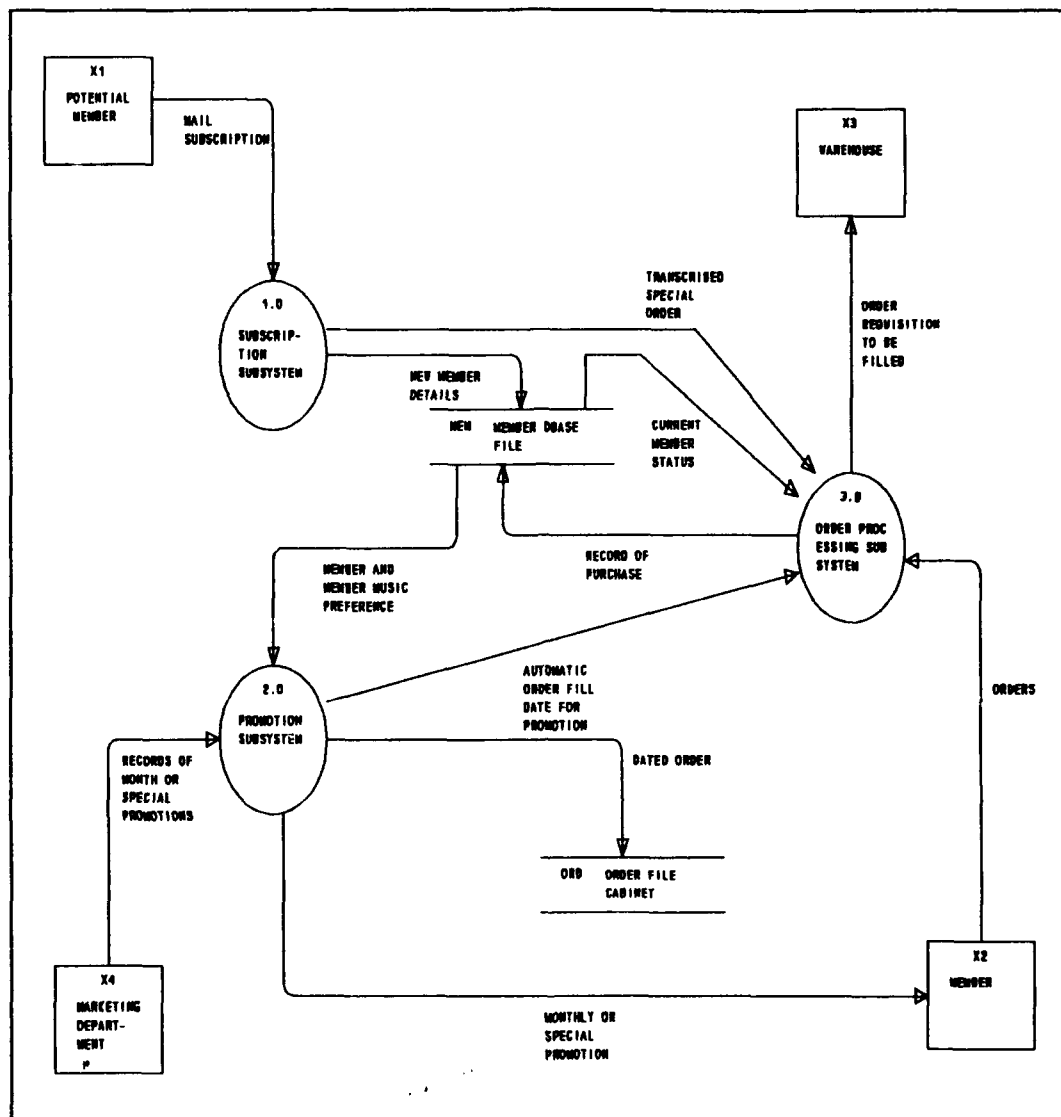


Figure 6-7 Excelerator/IS 1.9 Sample Project Diagram

Excelerator/IS includes a unique Presentation Graph option to help users visualize the proposed system. It provides different objects and icons that can model subjects in a variety of pre-defined notations such as a person object which can be used to represent a customer. This option can be used to adhere to a well-defined set of rules. For example,

it can be used to create a flow chart depicting the procedural logic of a particular activity.

The best use of this option involves the creation of a decomposition diagram representing a pictorial outline of the entire system specification. Objects on a decomposition diagram represent graphs and other components of the specification. The objects presented can be exploded to any XLDictionary entity, such as a Screen Design or DFD. Thus, a user can the outline graph to navigate through a complex design moving from one entity to another.

The diagramming facilities within Excelerator/IS provide a variety of diagrams with a high degree of flexibility. The facilities provide the information in a variety of formats depending on the user preference. For a personal computer based system, the facilities offered by Excelerator/IS are significant.

B. StP 4.2A (SUN) OF INTERACTIVE DEVELOPMENT ENVIRONMENTS

1. Hardware/Operating System Evaluated On

The tool was evaluated on a Sun Model 3180 using Sun Version 3.5 operating system. No compatibility problems with any of the hardware nor the operating system were observed.

2. Tool Description

StP is a full lifecycle support tool which is oriented towards both business and real time applications. The tool contains an integrated set of graphical editors which

focus on the analysis and design phases of system development. Figure 6-8 contains the set of graphical editors supplied with the tool. Full lifecycle coverage is provided via an automatic code generation capability contained within several of the editor facilities. The tool also provides "rapid prototyping" capabilities and a limited reverse engineering capability.

3. Methodology Supported

StP 4.2A supports several methodologies via the graphical editors supplied with the tool. The DFE supports a structured analysis approach using either Yourdan/Demarco or Gane/Sarson notation to provide a "functional perspective" of the system.

A "data perspective" of the system is provided by both the DSE and ERE. The DSE is used to construct data structures which can generate declarations for C, Ada and Pascal. The DSE is designed to be used in conjunction with the DFE to support structured analysis activities and the SCE to support structured design activities. The ERE is used to define entities and their relationships using the CHEN style to support structured analysis efforts and can be used to generate database schemas within StP.

Design support is provided by the SCE. The manual suggests following Yourdan/Constantine structured guidelines.

Graphical Editors

Dataflow Diagram Editor (DFE)

Data Structure Editor (DSE)

Entity Relationship Editor (ERE)

Structure Chart Editor (SCE)

State Transition Editor (STE)

Transition Diagram Editor (TDE)

PICTure Editor (PCT)

Control Specification Editor (CSE)

Document Preparation System (DPS)

Figure 6-8 STP 4.2A Graphical Editors

The SCE also contains the capability to generate program design language or code templates.

Real time support is provided by the STE which is an graphical tool for drawing state transition diagrams. The STE can be used in conjunction with the CFE and the CSE to provide complete real-time extension (control information) to Structured Analysis. The CSE is based on the Real-Time Requirements Specification Methodology.⁹

The RAPID/USE facility within the tool provides capability to prototype the user/program dialogue to model the user interface and build a working version of the system. The

⁹ Hatley, Derek J., and Pirbhai, Imtiaz A., *Strategies for Real-Time System Specification*, New York, NY: Dorsett House Publishing, 1987.

RAPID/USE facility is based on the User Software Engineering Methodology.¹⁰

4. Hardware/Operating Systems Supported

Figure 6-9 contains the workstations and operating systems supported by StP 4.2A (Sun Version) along with the disk space storage requirements for each model. Additional

Hardware/Software Systems Supported

<u>Workstation</u>	<u>Model</u>	<u>Oper/Sys</u>
Sun 3	All Models	3.5, 4.0
Sun 386i	All Models	4.0
Sun 4	All Models	4.0
Sparcstation	All Models	4.0

Hardware/Software System Requirements

<u>Workstation</u>	<u>Oper/Sys</u>	<u>Storage Required</u>
Sun 3	3.5	26 Megabytes
Sun 3	4.0	19 Megabytes
Sun 386i	4.0	19 Megabytes
Sun 4	4.0	19 Megabytes
SPARCstation	4.0	19 Megabytes

Note: The storage requirements reflected above do not include the sample documents provided in the *desktop* and *SampleDocs* directories. If installed, add 11 megabytes to support these files.

Figure 6-9 StP 4.2A Sun Version Hardware/Software Systems Support/Requirements

¹⁰ Wasserman, A.I., "The User Software Engineering Methodology: An Overview," in *Information System Design Methodologies*, ed. T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart. Amsterdam: North Holland, 1982.

workstations supported by StP 4.2A include: Apollo, HP 9000 Series 300, VAXstation and DECstation.

5. Installation

The installation process was performed with the system's administrator. The tool and the documentation was delivered as a complete package. The program consisted of a single 1/4" streamer tape (tape cartridge). The installation was performed by loading the installation cartridge, executing a batch file, and then following the prompts.

Installation was straightforward and uneventful. The administration manual contained step-by-step instructions for the entire process. The Administration manual also included a sample installation process with detailed explanations of each option which was particularly helpful. The process was very easy to follow and understand.

The documentation package included a Release Notes guide which especially helpful regarding product changes not incorporated in the original documentation especially installation sensitive information.

6. Documentation

Ample documentation is provided. The documentation set is unique to each hardware system supported. The Sun Version set consists of three manuals: StP User Manual, StP Reference Manual and StP Administration Manual. The User Manual contains the tutorials and all documentation support

for the editor facilities. The Reference Manual is primarily devoted to the programming facilities of the tool, while the Administration Manual contains the installation instructions and other material directed towards the System Administrator. The set also included a Release Notes document describing new features incorporated, compatibility information with previous releases, repaired problems and known limitations of the current version. Each manual contains both a table of contents and an index with all major sections tabbed and separated. Each major section includes an individual index and an overview which enhances reading and searching efforts.

The User's Manual tutorial provided a Basic Tutorial and an Advanced Tutorial. The Basic Tutorial provided a general overview of the StP environment by demonstrating the basic operations associated with a graphical editor (DFE) used by the tool. Operations were mainly oriented towards the use of the graphic facility, but did provide a brief section on describing data to the dictionary. Overall, the tutorial provided a friendly initial view of one aspect of the tool.

The Advanced Tutorial walks through the analysis and design of a small system by describing the use of four different editors: DFE, ERE, SCE, and OAE. The tutorial concentrates on diagram locking, version control, completeness/consistency checking, the use of the data dictionary, process specification generation, printing diagrams, and supposedly report generation.

The tutorial ends by referring the user to the documentation associated with each editor for the use of their more powerful features and an additional five chapters associated with customizing and extending the StP environment.

The Advanced Tutorial's approach could best be described as one of "tunnel-vision". The user is thrust into a project with a brief overview and then immediately launched into constructing various aspects of the sample system. Most of the uses of the editors involved are very limited. For example, the use of the DSE and the OAE are limited to one example format for entering data structure information with the DSE and data type information with the OAE. The ERE section was more of an overview as well. The SCE section did provide more depth of use than the other sections which helped to tie some significant concepts together. The sections' main value is that they do provide key points of interest regarding the editors such as the distinction between the set of "Note Types" available within the ERE and the DSE. The set of note types available within the ERE is different from those available for elements in the DSE, since the objects are of different type.

Overall the Advanced Tutorial lacked sufficient depth of use for the majority of the editors presented. The limited mental model presented by the sample system and the lack of system requirements and any linkage to such severely limits the user's view of the overall purpose of the system.

Moreover, it ignores several of the powerful features of the tool preferring to have the user confront these features within the individual editor sections.

The tutorials associated with individual sections of the tool due provide an overview of what is expected of the user. They specifically describe which sections of the documentation must be completed before attempting a particular section. For example, the RAPID/USE section requires the completion of the following sections prior to use: Troll/USE Relational Management System, Appendix C: RAPID/USE Commands and Appendix D: Troll/Use Commands. The only negative aspect regarding these tutorials are that they are not a continuation of the system described in the Advanced Tutorial. As a result, the user has to develop a new mental model for the sectional tutorial applications.

Besides the referrals to the customization chapters, the user is left to his own conviction as to what path is best to pursue for continued instruction. As a novice user, a more detailed consistent approach would be preferable. A finely detailed very precise cookbook approach would greatly benefit the first-time CASE user, especially with the variety of features offered by this tool.

One outstanding aspect of the documentation of the tool lies in its extensive on-line help. The help facility within the tool is quite informative. Although some areas are not covered (most of these are intuitive), those that are tend

to provide much more than superficial hints. In most instances, the on-line help is sufficient for the task at hand which prevents the user from having to revert to hardcopy documentation.

7. Interface to Other Products

Documentation provided with the tool indicates the standard version of the tool can be modified and extended by the user. The flexibility is provided via an open software architecture called Visible Connections. The architecture makes all interfaces to the tools within StP visible. These visible interfaces can be modified within the StP environment without having to access any individual tool source code. As a result, users can modify messages, database schemas, editors, fonts, startup menus and other aspects of the environment. The architecture supports the customization of the various templates used within StP as well.

StP contains a Tool Information System (TIS) which acts as the central point for customizing and extending the StP's environment. The TIS works in conjunction with the StP Tools Library and one or more tool information files to assign values to various variables of the environment. The tool information file is typically named "toolinfo".

The TIS along with the Tools Library allow users to customize tools they build within StP. These customized tools can be linked to other tools within StP or operate completely

independent of them. Individual users or entire projects can be customized via the toolinfo file to provide their own tool environment.

The Tools Library contains various routines which facilitate customizing the tool interface via a programming language interface to components within in the suite of StP tools. The routines in the StP Library are logically divided into three groups: 1) Troll/USE group for interfacing with the database management system 2) Toolinfo Group for manipulating information within the Toolinfo file 3) RAPID/USE group which provides runtime support for the RAPID/USE applications development and rapid prototyping system. The tool provides a C language interface for all three groups. It also provides a Fortran 77 interface for the Troll/USE and RAPID/USE groups which is only available for Sun systems.

The interface routines are not intended for the average user. The tutorial within the Library does not provide examples of every routine it contains. It provides a few short programs which demonstrate a sample usage of the most complex routines. By understanding the short programs provided, the user is not expected to have any difficulty using the other routines in the library. The tutorial further assumes the user is familiar with the C programming language. These assumptions dictate the average user must rely on

systems support personnel within their own organization to extend the capabilities of the tool.

StP Visible Connections provides an interface to a wide variety of other products which enable the tool to provide complete lifecycle coverage. Figure 6-10 identifies the third party products supported by StP by application area. The tool does provide specific support for several desktop publishing systems. Documents can be output to files in a particular format and then edited in the desired desktop publishing systems. StP 4.2A generates compatible output files for FrameMaker 2.0 and Interleaf 4.0.

8. Multi-user Support

The StP environment provides multiple-user concurrent access to the tool. Concurrent access allows groups of users to work on the same project at the same time (possibly even the same diagram), but it requires a sophisticated locking scheme to prevent possible problems such as simultaneous editing. StP provides locking features designed to control concurrent access to diagrams.

The tool provides "automatic locks" and "user-defined locks" for all the editors. Once a diagram is loaded, the editor sets an automatic lock which prevents a diagram from being overwritten by another user while the file is in use. Other users can read a diagram that is in use and are notified by the system upon access that the diagram is locked (the

Configuration Management:

Product: CCC
Product: DSEE

Company: Softool Corporation
Company: Hewlett-Packard Company

Database Management System:

Product: StP/INGRES Interface
Product: SYBASE SQL Server

Company: Softool Corporation
Company: Sybase Incorporated

Detailed Design:

Product: KeyOne

Company: LPS s.r.l.

Fourth Generation Language:

Product: UNIFACE

Company: Uniface B. V.

Integration Platform:

Product: HP SoftBench/HP Encapsulator
Product: Software BackPlane

Company: Uniface B. V.
Company: Atherton Technology

Programming Environment:

Product: Saber-C
Product: VADS
Product: VAXset

Company: Saber Software, Inc
Company: Verdix Corporation
Company: Digital Equipment Corporation

Reverse Engineering:

Product: BAT

Company: McCabe & Associates, Inc

Systems Simulation

Product: SES/workbench

Company: Scientific & Engineering Software, Inc

Technical Publishing System:

Product: FrameMaker
Product: Interleaf TPS

Company: Frame Technology Corp
Company: Interleaf, Inc

Testing:

Product: StP/TESTBED Interface
Product: START

Company: IGL Technology
Company: McCabe & Associates, Inc

Version Control:

Product: RCS

Company: Hewlett-Packard Company

X Windows Support:

Product: StP X11-Based Software Development Center

Company: Interactive Development Environments

Figure 6-10 StP Third Party Product Support

"store" is replaced by a "locked" button). Users can edit a diagram that is locked by changing the name of the diagram to an unused name or change the name of the project directory.

Users can also set locks. User locks are necessary since automatic locks are only set when a diagram is loaded. Once unloaded, it can be edited by another user. For example, if a user works on a diagram and then works on a decomposition of the original diagram another user can edit the parent diagram. To prevent this, the user must lock the diagram prior to working on the decomposition. User locks are permanent. Therefore, a user lock can also prevent another user from editing the finished diagram when a user desires to work on the diagram for more than one session.

Both automatic and user locks can be controlled (enabled/disabled) by the system administrator, a project manager or any user defined as a lock administrator. The system provides a User-Interface facility for utilizing locking functions which is quite easy to use. Although it is easy to use, this feature requires some forethought and oversight to ensure it is set up and administered properly.

9. Network Support

StP 4.2A (Sun Version) provides network support via protocols bundled with the operating system supplied with the workstation. The specific protocols provided with the Sun workstations are TCP/IP. Many Sun workstation network

implementations tend to use Baseband Ethernet as the communication medium.¹¹ Documentation provided with the tool was not very detailed regarding network technology.

StP does offer a unique tool network feature in 4.2A called "heterogeneous database support". This feature allows the tool to read project databases located on machines with completely different machine architectures. For example, the tool running on a machine with an Intel 80386 CPU can read a database produced by a machine with a Motorola 680x0 series CPU and vice-versa.

The tool accomplishes the architecture independence by providing a heterogeneous version of troll: htroll. Troll/USE is the relational database management system utilized by the tool. All database definition and manipulation is handled through Troll/USE. StP 4.2A actually provides two versions of troll for each architecture: htroll and otroll. Otroll is the original (native) version of troll provided for each specific platform supported (i.e., troll for Sun 3 or troll for Sun 4). Unless there is a need to access databases on a variety of machine architectures, the manual recommends using the native version of troll since htroll imposes a minor database performance penalty.

Htroll provides a great degree of flexibility. Users are not locked into a specific machine architecture to ensure

¹¹ Telephone conversation between Dennis Freeman, Sun Microsystems, Inc., and the author, 14 June 1990.

StP compatibility throughout the organization. The troll version is specified by the user during the installation process. If different machine architectures are acquired after the tool has been installed, users can quickly and easily convert from otroll to htroll.

10. DoD STD-2167A Support

StP 4.2A directly supports the following DoD-STD-2167A Data Item Descriptions (DID's):

Software Requirements Specification

Interface Requirements Specification

Software Design Document

Interface Design Document

2167A support is primarily provided by the DPS and OAE. Both of these tools include a system of templates that define the information the tool requires as input and produces as output. Both of these tools "must be equipped" with their 2167A templates in order to generate the 2167A reports. There is a 2167A template for each DID supported. The 2167A support is focused on key areas within the analysis and design phases of development.

The DPS templates are used to control the content and format of the document via each different DID template. The template files are made up of code written in the DPS template language. The code is used to perform the various functions to

produce the 2167A reports. The DPS templates can even be modified to customize the document.

The DPS and OAE templates are used to generate the reports required above by extracting the information needed from the data dictionary. The extracted information includes the annotations written with the OAE and diagrams drawn with the graphical editors. The OAE information is derived from the annotation fields that appear when OAE is invoked for a particular object. These fields, called Note Types, are used to specify the general characteristics of an object. Within the Note Types, a user can edit additional fields called Note Items corresponding to the Note Types which allow the entry of more specific information that describes the object being annotated (i.e., integer, constraints, additional text descriptions, etc..). This item will be explained in greater detail in the traceability section.

The DPS formats the extracted information into tables, diagrams and standard text to produce the desired 2167A documents. The DPS contains "user input" fields which prompt the user for additional information required to complete a report which is not provided in the model developed during the analysis and design phases. For example, information such as the name of the contractor and the contract number are requested by the user input fields.

This is an extremely powerful feature. The templates provided obviate the need to compile and format the required

information once the system is complete. Although it saves a substantial amount of documentation effort, it requires meticulous effort throughout development to ensure the information is entered in the appropriate fields within the Note Types.

11. User-Interface

StP provides an easy to use window based interface combined with graphic menus and icons. The interface exemplifies the power of this tool. It provides a great degree of flexibility and control.

The Main Menu window of the tool allows a user to call any tool (i.e., editor) directly by selecting the icon representing the particular tool desired. Once selected, a separate window and menu for the tool selected is provided. Various pop-up menus and submenus supporting additional operations can be accessed from within each window as well. Navigation between and selection within the windows and menus are enhanced by the use of a mouse.

The tool relies on the use of a three button mouse with each button providing a specific capability. The left mouse button is the "Select Button". It is used to make selections from the Main Menu window, the editor windows and to select text fields for text entry. The middle mouse button is the "Undo Button". It supports drawing operations by providing the capability to undo the most recent drawing

operation attempted. The right most mouse button is the "Menu Button". This button is used to make selections from the various pop-up menus and submenus available within each window application.

The window and menu formats are consistent throughout the entire tool. All menu screens and diagrammatic screens follow the same general pattern. This similarity enhances familiarity with the use of the tool. Both the Basic and Advanced Tutorials emphasized this aspect of the interface to reinforce the users efforts.

StP also offers the capability to modify the user-interface by customizing the Main Menu. The documentation indicates can the menu can modified and extended to define local preferences. After modification, users can not only access StP tools, but also non StP tools and programs. The manual emphasizes that **this capability is not meant to be accomplished by the average user.** It stresses System Administration personnel perform all necessary actions. The manual does provide all pertinent information necessary to accomplish desired changes.

StP offers a powerful easy to use interface, but it's capabilities require the user to practice some discipline. For example, users can open as many windows as they want and enter multiple editors and quickly lose control or get confused. The tutorials are designed to ensure the users get a great amount of practice with window control and

manipulation. Therefore, it is vital that users concentrate on the tutorials for the basic operation of the system.

12. Traceability of Requirements

Version 4.2A provides a ReqTrace template family containing three templates for tracing requirements: 1) TraceModToProc 2) TraceModToMod 3) ReqDoc. The TraceModToProc template traces structure chart modules back to dataflow processes. The TraceModToMod traces dataflow processes back to structure chart modules. The ReqDoc template produces a Requirements Document summary report that itemizes how each requirement is satisfied by objects in the project database.

The ReqTrace templates require special Note Type annotations which can only be done with the OAE. The OAE provides a specific Note Type called Requirement to support traceability efforts. Once this note type is added to an object, the user must select Edit Note Item at which point the system provides the following data fields:

Requirement Name:

Requirement Document:

Requirement Paragraph Name:

Requirement Paragraph Number:

Once these fields are completed, the information can be extracted by the ReqTrace templates. The Requirement

Document summary report is generated by tracing all objects associated with a specific Requirement Document. The templates search for specific requirement paragraph information for each object associated with the requirement document identified. Once the dictionary search is completed, the templates format and produce a traceability report containing the extracted information.

The ReqDoc template is very useful, but it was not very clearly detailed. There was only one page in the entire manual which identifies this feature and it instructs the user to use the "Help" button in the Document Definition Area for pertinent information. The help facility provides the right information, but did not detail exactly how to enter the name of the document to trace. Generating a report required several trial and error attempts to figure out how to enter the document name.

13. Dictionary/Repository

The StP Data Dictionary is a set of programs that connect the various StP editors with a relational database. As such, it serves as a medium for connecting the editors to each other which enables the tools to share the same project database for a system. Therefore, a user can enter information with one tool and view it with another. Information within the dictionary is organized by projects.

The main purpose of the data dictionary is to store "data" information and ensure that names are used consistently. Only one rule is uniformly enforced within the data dictionary: all names used must be unique, with only one exception. For example, a name chosen for dataflow may only be used for a data item, and not also as a process name. The only exception to this rule is the name of a process defined in a dataflow diagram may be the same as the name of a module defined in a structure chart.

Input to the dictionary can be accomplished in several ways: by defining data structures as they are added to a diagram in the DFE (data stores and dataflows) by using the DSE or by defining data types and attributes within the ERE. The documentation recommends the use of both approaches to gain the maximum power from the dictionary.

The DSE uses hierarchical data structure diagrams to decompose "dataflows" and "data stores". The dataflows and data stores are defined via dataflow diagrams and structured charts. It supports simple, complex and hierarchically structured data objects. Data types, constraints and values are added as notations to support the generation of data declarations for the various programming languages supported by the tool and as input to the data dictionary.

The ERE uses entity-relationship diagrams to model system data by defining entities and their relationships. It is especially suited for "data intensive" applications. The

ERE uses the data dictionary for the storage and retrieval of all of it's data. One important aspect of the ERE is it's generation capability. It not only generates input to the data dictionary, but database schemas as well. For example, the ERE can generate the Backus-Naur Form an ER diagram which is very similar to the notation used in structured analysis. It also includes checking programs to aid in checking and verifying the decomposition and definition of a data structure, and the consistent use of names in the data dictionary.

The data dictionary also provides its own interface for viewing stored data called the StP Data Dictionary Browser (StP/DD Browser). The StP/DD Browser allows users to browse the dictionary contents to perform checks for individual items as well as listing objects and their definitions. Its primary use is for short sessions or spot checks which would be much slower if accessed through the Main Menu.

14. Prototyping

StP 4.2A provides a facility called RAPID/USE which can quickly generate user-program interfaces and construct complete interactive systems. It is based on the User Software Engineering (USE) methodology which provides a support environment for the development of interactive systems. RAPID/USE is intended to rapidly prototype a system by successively refining models to construct a specification.

It has the capability to construct complete systems which may be used as production systems.

USE utilizes a unique set of state transition diagrams (USE STD's) to model the flow of the interactive dialogue. The USE STD's are used to model the entire interactive session, which it calls a "conversation". The conversation consists of one or more subconversations modeled by USE STD's. In reality, the set of subconversations is simply the set of messages displayed by the system.

According to the RAPID/USE documentation, the goal of RAPID/USE and the USE approach is to provide the developer low-level control over the placement of characters on a display and over the user's input. The tool contrasts its approach to other systems where the approach is "hard-wired" via fixed formats, fixed screen layouts and fixed concepts of user interaction. However, in order to achieve this fine detail, the facility requires a lot of user effort. In essence, a user of this facility **must become a programmer**. As a result, time and scope limitations prevent an in depth attempt by the author to exercise this facility.

The RAPID/USE documentation also specifies that a user must know several other sections of the manual prior to attempting to use it. Unfortunately, one of the other sections specified, RAPID/USE Commands, requires the user to complete the RAPID/USE section before using it. Therefore,

the user is left to his own judgement how best to proceed to learn the facility.

RAPID/USE actually consists of two parts: a Transition Diagram Interpreter (TDI) and an Action Linker. The TDI is used to create an executable version of the user/program dialogue by executing the USE STD's. The TDI also provides direct linkage to the Troll/USE database system. The linkage allows transitions to call sequences of statements written in the Troll/USE data manipulation language and pass any needed parameters. The TDI alone is enough to construct an entire system if the dialogue is comprised only of human interaction that causes database manipulations.

For more complex systems, which include actions programmed in a high level language, the Action Linker is also needed. The Action Linker allows code to be associated with the actions specified in the USE STD's. As a result, the actions can be performed in conjunction with transitions. The code used with the linker can be written in several different languages (not specified).

This a powerful capability, but it carries a cost: it not only requires the user to learn a specific methodology, the user must also be willing to program it.

15. Consistency/Completeness Checking

StP provides several techniques for ensuring consistency and completeness. One such technique was

previously described in the dictionary section. The dictionary contains one rule which is uniformly enforced. All names used for the various objects must be unique except for the names of processes in dataflow diagrams and processes in a structure chart. This rule prevents the same name from being used in various places for different items, thus ensuring no duplication of a name and possible confusion is introduced into the dictionary.

There are several formal mechanisms provided by the tool. The DFE contains a **check decomp** command in the "data dict" submenu which ensures a process is decomposed into two or more low level processes and that all inputs and outputs to the "parent" process are matched in the diagram. The main thrust of the decomposition process is balancing the dataflows from one level to the next. For example, a composite dataflow on an upper level which has not been structurally defined will generate a decomposition error. The tool generates an error since it has not been told that the component dataflows on the lower level belong to it. The check facility will continue to return error messages until the component items have been defined via the DSE to indicate they belong to the composite dataflow.

The **check StP/DD** command within the "data dict" submenu provides another important check for the correctness of the system. The check StP/DD command ensures that all data items are completely defined and that there are no undefined

elements contained in the data dictionary. The check returns all information available for the data within the diagram regardless of the editor used to input it. If a data item had an entity definition associated with it, the entity definition as well as any other information would be returned.

The check commands identified above are not limited to the graphical facilities. These commands can be originated from the Main Menu without having to access an editor. The information provided can be displayed to the Execute Window for quick viewing or to a file if accessed via the Options Area within the Main Menu. These reports are easy to use, but can generate a tremendous amount of information.

16. Training Support

IDE offers educational and consulting services as well as training support services. Training courses are offered for both new and experienced users. Instruction includes methods training in both structured analysis and design techniques and object oriented structured design. The object oriented design can be customized for users of ADA, C++, Objective C and other object oriented languages. Tool training classes are available for all languages and can be customized for current or new StP users.

IDE provides consulting services for CASE application and development. Support is provided for both newly and well established software development organizations. Services

range from assisting in the definition of a formal software development process for a newly established organization to providing an assessment of the current level of software engineering maturity of a well established organization.

Support services provided include: hotline support, quarterly newsletter, user conferences and numerous Regional User Groups covering the entire country specifically dedicated to StP. IDE provides ample support not only for StP, but the entire CASE implementation process. The training provided can be tailored to meet individual user needs. The focus on entire CASE organizational support emphasizes the commitment required to support the introduction of the tool and the proper application of it. It also emphasizes the financial requirements for training and implementation can be significant.

17. Diagramming/Graphic Facilities

The graphics facilities within this tool are quite impressive. The various graphical editors within the tool all provide similar diagrammatic facilities which facilitate learning. New users are well introduced to the tools's diagrammatic capabilities by the Basic Tutorial, hence little effort is required to get started with diagramming efforts.

The most striking aspect of the diagrammatic facilities is their extraordinary "ease of use" despite the tremendous array of features available. The commands

available within each editor are easily accessed by clicking the Menu mouse button (left button of a three) while in the drawing area. Figure 6-11 depicts the pop-up Command Menu available within the DFE. Utilizing the Command Menu and the

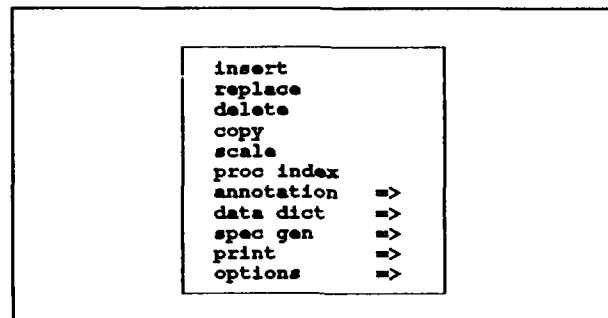


Figure 6-11 StP DFE pop-up
Command Menu

mouse, a user can quickly and easily insert, move, scale and replace various objects. The middle button of the mouse functions as an "undo" button which can be used to cancel previously issued commands/operations within the drawing area. This feature is particularly useful for manipulating lines/arcs (connections) between objects.

The Command Menu also offers a variety of options to support diagrammatic efforts. For example, the Options submenu within the Command Menu include the commands "put" and "get". The put/get commands offer a much more powerful form of copying than the simple copy command available at the Command Menu. The put command places selected material in a buffer which can later be placed in any location specified by the user with the get command. Figure 6-12 depicts the DFE Command Menu displaying the Options submenu.

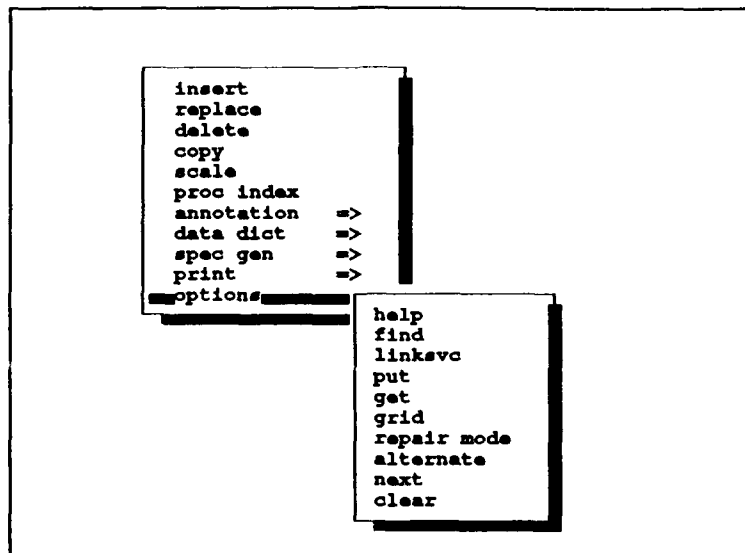


Figure 6-12 DFE pop-up Command Menu with Options Menu

The most useful aspect of the put/get command feature is transferring information between diagrams. A user can select a single node (process) or an entire diagram and open another DFE window and transfer the information to another system. As a result, a user can reuse all or part of a model as needed. The put/get commands can operate for multiple windows within the same editor (i.e., DFE, ERE), but cannot transfer information from one editor to another.

The most impressive feature noticed by the author is the printing capabilities available for diagrams. The drawing area within each editor acts as one big page for each diagram. A user can enter objects at will. As the diagram grows, the user can select the "Zoom" command to display more of the diagram by reducing it's size. When using the zoom command, a user is not only able to see the entire diagram, but can see

the diagram as it will be printed. This **what you see is what you get** is a very useful feature. The user doesn't actually have to print the diagram to know what it will look like.

The tool also offers a high degree of flexibility to the user. Depending on a user's preference, diagrams can be printed from within any editor or from the Main Menu. The tool also offers the ability to print one, all or even selected diagrams. Diagrams can be sent to a variety of desktop publishing systems for enhanced graphics.

The diagrammatic capabilities are indicative of most of the capabilities provided by StP 4.2A. They are powerful, quick and easy to use, but do require some use to master due to the variety of features offered. StP 4.2A takes full advantage of the workstation environment to provide the user with a complete graphics arsenal to attack diagramming efforts.

C. EQOS 4.0 (PC-Vers) OF SOFTWARE PRODUCTS & SERVICES, INC.

1. Hardware/Operating System Evaluated On

The tool was evaluated on a 386 clone (20 Megahertz) with an 80 megabyte hard drive and a VGA monitor using MS-DOS 3.3 operating system. No compatibility problems with any of the hardware nor the operating system were observed.

2. Tool Description

EPOS 4.0 is a fully integrated lifecycle tool which is primarily oriented towards large real time applications, but supports business applications as well.¹² Total lifecycle support is provided via three specification languages and six tool systems. Figure 6-13 identifies the specification languages and tools systems provided. The tool also provides graphical support along with code generation and extensive project management capabilities.

3. Methodology Supported

EPOS 4.0 provides systematic, structured methodologies via the various specification languages and tool systems provided. The tool supports data modeling via dataflow and data structure diagrams and the EPOS-R specification language.

Six different design methodologies are supported by EPOS 4.0: function-oriented, event-oriented, module-oriented, dataflow-oriented, data structure-oriented and device-oriented. Specific modeling tools include: Hierarchy Diagrams, Nassi-Schneiderman Diagrams, Data Structure Diagrams (Jackson), Petri Net Diagrams, Hardware Block Diagrams and the

¹² Lauber, Rudolph, and Lempp, Peter, "What Productivity Increases to Expect from a CASE Environment: Results of a User Survey," *IEEE Software Development: Computer-Aided Software Engineering (CASE)*, 1989, p. 106

EPOS-R: Specification Language for Requirements Definition
EPOS-S: Specification Language for System Architecture and Design (Hardware & Software)
EPOS-P: Specification Language for Process Management and Production Control
EPOS-A: Analysis Tools
EPOS-C: Communication Tool System
EPOS-D: Documentation Tools
EPOS-M: Management Tools
EPOS: Method Support Tools
EPOS: Code Generation Tools

Figure 6-13 EPOS 4.0 Specification Languages and Tool Systems

EPOS-S specification language. EPOS-S with formal syntax and defined semantics can be used for describing systems design.

Project/product management input (i.e., configuration management info, quality assurance info) is accomplished via EPOS-P specification language which is used to generate Work Breakdown Structures, Progress Charts, PERT charts, Gantt Charts, Network Diagrams and Responsibility Matrices.

4. Hardware/Operating Systems Requirements

EPOS 4.0 supports a wide variety of hardware and operating systems. Figure 6-14 identifies the various systems supported by the tool. Figure 6-15 contains the specific hardware and operating systems requirements for AT-compatible systems.

PC Hardware Requirements

Disk Space

7 MByte for EPOS programs
10 - 12 for data storage
Recommended Minimum: 20 MB

Memory

Recommended Amount: 640K

Graphics Board

IBM EGA, Std IBM CGA, IBM Monochrome, CGA, Hercules, Toshiba, VGA

Mouse (Required if tool is used with a graphics package (i.e., AutoCAD))

Printer Support

Requires Laser printer; no models specified

Plotter Support

Graphtec (Watanabe) WX 4731
Hewlett Packard HP7475A
Gould Colorwriter 6210

Operating System Requirements

IBM-PC AT	MS-DOS 3.3
IBM-PC AT Compatibles	MS-DOS 3.3
HP Vectra	MS-DOS 3.1
Intel 8086	iRMK
Intel 80286	iRMK

Figure 6-14 AT-compatible Hardware and Operating System Requirements for EPOS 4.0

5. Installation

The installation process was cumbersome and confusing. The tool and the documentation were delivered as a complete package. The package included an installation manual and a two page PC/AT Installation procedure. The package nor the documents specified which document had precedence which greatly hindered the installation process.

The program consisted of 14 1.2 Megabyte high density diskettes (5.25"). The diskettes were delivered in a DOS Backup Format (standard company practice). To effect the

Standard EPOS 4.0 Implementations

DEC Micro VAX II, VAXstation	VMS 3.5 and up
DEC VAX 11/700 series	VMS 3.5 and up
DEC VAX 8000 series	VMS 3.5 and up
IBM 370	VM/CMS
IBM 370	MVS/TSO
IBM 4381	MVS
IBM-PC AT	MS-DOS 3.3
IBM-PC AT Compatibles	MS-DOS 3.3
HP Vectra	MS-DOS 3.1
Intel 8086	IRMX
Intel 80286	IRMX
Intel Workstation 86/330	IRMX 86 Release 6 and up
Intel Workstation 286/330	IRMX 86 Release 6 and up
Siemens 7000	BS2000
Data General MV series	AOS/VS
PCS Cadmus	MUNIX
Motorola 68000/10/20	UNIX BSD 4.2/UNIX System V
Apollo Workstation	Aegis/UNIX
Sun Workstation	UNIX
HP 9000 Workstation	UNIX

Figure 6-15 Systems Supported by EPOS 4.0

installation, the diskettes had to be restored to the hard drive via the DOS restore command. The process was very tricky because some of the disks were mislabeled. A tremendous amount of trial and error was required to solve the error since DOS does not identify the error explicitly.

The process was further complicated by the lack of specific directions for setting up the disk storage area. The two page procedure did not specify the exact subdirectories needed prior to restoring the disks to the drive. The manual identified the subdirectories needed, but contained a couple

of errors (i.e., **EXAMPLES** vice **EXA**) which only added to the confusion.

The final portion of the installation process consisted of updating the autoexec.bat and config.sys files to enable DOS to run the software. Both the autoexec.bat and config.sys files had to be modified manually. The two page procedure and the manual conflicted on several settings in both files. The manual actually specified an incorrect setting in the config.sys file.

The installation process was laborious and confusing. Upon communicating with company representatives, they indicated the manual was to be ignored and the two page procedure should be used in it's place. This fact should have been specified in the documentation. The process itself would be greatly simplified if it were automated (i.e., batch files) and the guide/manual was written to provide step-by-step instructions.

6: Documentation

Substantial documentation is provided. The documentation set consists of an Installation Manual, a User Manual/Interactive Tutorial, a Pocket Guide, a Graphic Editor Manual and two-volume set of User Manuals for the various specification languages and tool systems.

The installation documentation provided is not precise enough for the process required. Much more detail is

needed to ensure the installation is performed correctly and efficiently. A Pocket Guide is bundled with the Installation Manual. The guide is very informative. It provides a compact description of the syntax of the specification languages and a comprehensive overview of EPOS's user interface, analysis and documentation features. The user interface section provided an excellent explanation of the dialogue mode between the user and the system and system macros available which greatly simplified the initial interaction with the tool. Unfortunately, the guide contained some errors for command functions regarding the use of the tool in the mask mode (menu driven mode). It specified to use the Ctrl-key of the keyboard to invoke certain commands which actually must be invoked by the Alt-key. This problem was easily resolved by using the on-line help which provided the correct information.

The User Manual/Interactive Tutorial was intended to provide the user with an overall feel for the tool. It was able to accomplish the task, but more detail would aid the orientation of the user to the tool. The manual refers to an overview of the tool which was not provided and provides a statement for the introduction indicating the introduction is being revised. Moreover, it refers to a sample project (Heating Control System) which did not appear to be provided (never specified location: the tutorial used a Traffic Control

System). The Heating Control System example was provided in the two-volume User Manual set.

The tutorial was simple, easy and straightforward. It provided step-by-step instructions accompanied by partial expected responses while in the command mode and full screen responses in the mask mode to ensure the user is in the correct facility and performed the desired action. **It was a good learning aid.**

The only negative aspect of the tutorial involved the dialogue mode. The tutorial begins in the command mode, but does not specifically identify it as such. The tool as installed came up in the mask mode which caused some initial confusion. After progressing through the tutorial (to page 31), it addresses the dialogue mode used as the command mode and introduces the mask mode portion. Learnability, especially for new users, would be enhanced if the dialogue mode was specifically identified at the beginning and the mask mode was introduced prior to the command mode.

The two-volume User Manual set provide the exact syntax definition and detailed discussions of the EPOS semantics. Volume I provided an excellent overview of EPOS. They are thorough and well written and contain descriptive, easy to use indices which greatly facilitated information research.

Overall the documentation provided is good except for a few rough spots. The vendor indicated the documentation is

currently being revised and will be updated shortly. There is one surprising aspect regarding the documentation: the lack of specific information in major areas such as network support, multi-user capabilities, training support and particular interface capabilities to other products.

7. Interface to Other Products

The documentation does not elaborate much detail regarding the interface capabilities of EPOS with exception of the Graphic Editor's dependence on an outside vendor's product. It does indicate that the user can define data within the EPOS environment for extraction to other tools and define user commands within EPOS which will read in data from other tools.

The EPOS Graphic Editor is a graphical interface facility within the EPOS environment which relies on an outside product to provide graphical input to the tool. The outside product required to provide graphical input is "AutoCAD" by Autodesk AG. By using AutoCAD, users can draw diagrams on the screen which is saved to a file and then converted to a text specification thereby obviating the need for text specification entry.

Conversations with company representatives indicated a reverse engineering tool called **RE-SPEC** is available from EPOS. This tool transforms source code written in C, Pascal and Fortran R77 into an EPOS-S design specification. Once

transformed, users can manipulate the design specification as though it had been entered via the EPOS-S language. There are no requirements to support the specification as the tool only generates the design specification, but other tools such as EPOS-A (the analysis tools) can be used to identify objects and build requirements if needed.

8. Multi-user Support

The tool does not appear to provide any specific multi-user support. It does indicate accounts can be "initialized" for different users, but each account must have it's own individual database, system files, backup files and set-ups. It further recommends loading the delivered EPOS system in a separate "delivery account" so it can be tailored and kept for easy set-up of user accounts.

Conversations with technical representatives confirmed the tool does not support multi-user access. Files or individual objects and diagrams can be transferred between individual databases via a "conservation" feature. This feature converts the stored binary form of the database to ASCII file which can then be input into the other database.

One notable drawback of the tool is that it has no formal locking mechanisms to prevent different users from accessing different user databases. All control/management must be accomplished by the project manager (his/her own technique) or the users themselves.

9. Network Support

EPOS 4.0 does not identify any specific network communication system supported. The manuals indicate that in a decentralized workstation environment a suitable communications system is required to connect the local databases with the central EPOS project database, but does not mention any network communication details.

10. DoD STD-2167A Support

EPOS 4.0 does not provide 2167A support. However, conversations with technical representatives indicated an upcoming release (Version 5.0) will include the same 2167A support provided on the mainframe and workstation versions of EPOS. Detailed documentation is not available. The following information is based on conversations with technical representatives and sample 2167A literature provided.

Significant 2167A support is provided by the Flexible Document Generator (not available in version 4.0). The generator is designed to produce documentation according to templates supplied with the tool or the overall documentation specification, which can then be tailored to specific applications. Cover sheets, footers, headers and the appropriate outline (i.e., DID's) are automatically provided. EPOS can be delivered with documentation templates defining which graphics and tables to include.

No specific desktop publishing tool is required. A laserprinter is required when diagrams are requested within the text. Special drives for common laserprinters are available and interfaces to others are available through Postscript. Technical representatives stated 13 DID's are supported, but could not identify which ones.

11. User-Interface

EPOS 4.0 provides a simple, but very flexible interactive interface via the EPOS-C component. It's interactive dialogue provides friendly user-sensitive access for both novice and experienced users by providing two different dialogue modes: direct dialogue mode ("command" line mode) and using masks mode ("mask" or menu mode). Users can easily toggle back and forth between modes by invoking simple predefined system macros. For example, to enter the mask mode from the command dialogue mode, users simply type: \$masks <return>. To enter the command line mode, users press Alt-X and enter: \$command <return>.

Another user-sensitive feature provided allows the user to select one of three different familiarity modes: "short mode", "medium mode" and "long mode". In the short mode, the man-machine dialogue includes only the essentials to accommodate experienced users. In the medium mode, the dialogue presents a user with a list of possible inputs which is intended to satisfy the occasional user. The final mode,

long mode, not only provides the user with a list of alternatives, but gives a detailed description of each input and is oriented towards users who are not familiar with the system.

The dialogue mode provides further flexibility by offering two methods of entering information: alphanumeric and function keys. Whenever a specific section is highlighted, the appropriate selections for that section are displayed in a lower window along with a corresponding function key. Users can select the appropriate function key available and is identified at the lower portion of the screen (i.e., F1 for Requirements-Engineering in the EPOS-R section main screen)) or users can simply key in the alphanumeric information.

The mask mode of the tool offers a unique "memory" feature by providing the capability to store the previous selection. The masks have a memory which stores the information entered. If a user presses the wrong key or changes his mind he can quickly backup to the previous entry. The information can even be maintained between sessions which was especially useful in the tutorial. The vendor provided default values for several of the entries which greatly aided the initial interaction.

The tool does not support color operations, but does make extensive use of "reverse video and "heightened contrast". Data entry areas are displayed in reverse video mode while selection items are displayed in displayed in a

heightened contrast form. Although not as effective as color could be, the tool does try to maximize the mode offered.

The tool requires the use of a mouse **if it is used in conjunction with the AutoCAD program** to support the diagrammatic capabilities provided by the AutoCAD system. The documentation indicates the vendor is currently developing a mouse-driven interface for EPOS itself to be delivered in the near future.

The limitation of the interface is that it is a tree-oriented dialogue which restricts navigation efforts and sometimes buries the user at a fairly deep level. The tool does provide a backup (reverse command) along with an exit capability to minimize this impact and is in the process of developing a mouse-driven interface.

12. Traceability of Requirements

EPOS supports traceability requirements via special formats within the specification languages. Requirements and constraints are entered and specified via the EPOS-R specification language. EPOS-R is used to describe the problem statement (customer needs and system requirements documents) and to describe the problem solving concept and the functional requirements. It also contains a special lexicon (dictionary) feature which allows terms to be defined in both the problem statement and the problem concept. The lexicon's job is to store standardized definitions of the concepts and terms

to be used by the designers throughout the EPOS specifications.

EPOS-R is a semi-formal language. It is designed to accommodate a variety of personnel (i.e., business people, project managers, ..etc). It contains only such formal language elements (basic syntax rules) that can be easily understood without special EPOS training. Requirement and constraint are keywords in EPOS-R used for defining identifiable requirement components. When entering the problem statement, users can identify specific requirements and constraints via these keywords. Figure 6-16 shows two examples of a requirement and an example constraint specified in EPOS-R.

REQUIREMENT 2(0) <switch,substation>

"The switching of the switch is only allowed when the substation contains no packages."

CONSTRAINT 1(0)

"A signal light at an entry and exit of each substation tracks the passage of a package."

REQUIREMENT 3(0) <wrong-station>

"A package sent to the wrong substation must be recognized."

Figure 6-16 Example Requirement and Constraint Specified in EPOS-R

EPOS-S is much more formal syntax than EPOS-R. It is designed for hardware and software developers who usually have extensive training in dealing with formal notations. Figure 6-17 depicts the information provided by an edit mask

(template) for the type "action" used to declare design elements in EPOS-S. Since a requirement represents an identifiable component, a logical connection can be made between EPOS-R and the EPOS-S component that satisfies (fulfills) it. The requirement components are used as formal proof of the completeness of the system design, based upon the EPOS-R description of the problem statement and the solution concept.

```
#NEW
ACTION      .
DESCRIPTION:      .
PURPOSE:"      ".
FULFILS: REQUIREMENT      .
DESCRIPTIONEND:
DECOMPOSITION:      .
CODE:"      ".
INPUT:      .
OUTPUT:      .
ACTIONEND
[EOB]
```

Figure 6-17 Edit Mask for Action Type in EPOS-S

Various analysis checks conducted by EPOS-A exist for EPOS-R, but one check is particularly crucial for traceability efforts: Reference Analysis. When a reference analysis is conducted, EPOS-A yields a general survey of the current state of development of a project.

The survey identifies which requirements/constraints have already been specified in EPOS-R, and which of those have not yet been fulfilled, or only partially fulfilled. It also identifies which dictionary terms with the attribute

'REFERENCE' have already been realized by design objects, and which ones have not yet been fulfilled. The attribute 'REFERENCE', assigned to a term in the dictionary, means that a design object with the same name must be generated during system design.

Once completed, the survey is generated as a report in a prescribed format which is used as a requirements traceability document. Users can even input deadline dates via the DATE keyword within EPOS-S to monitor design completion dates and track performance. Surveys can be conducted throughout the project to verify and check for missing specification parts and specification errors. Thus, requirements can be tracked throughout the development process.

13. Dictionary/Repository

The EPOS repository is a home grown link list database. A overview of the database was provided in the Multi-user section which emphasized that the PC version is limited. Each database is unique. To support multiple projects, users must construct (initialize) a database for each project. A user can establish a completely different system in several sub-directories along with each database or load the database desired each time he/she logs in.

The EPOS-C component controls the user-interface which provides access to the EPOS environment and thus to the

dictionary. Figure 6-18 provides an overview of the EPOS environment which shows the user control over the environment and the structure of the support system for the database. The information within the database is organized by the three specification languages used: EPOS-R, EPOS-S and EPOS-P. In addition to generating the original database, users must initialize the individual EPOS sections as well. The tool does allow the user to accomplish the initialization in one operation.

After initialization, the tool prompts the user for a design method selection to be enforced by the database. As noted in the Methodology section, user have a variety of methods to choose from. Selection of a particular method enforces the use of certain types of design objects except in the method-neutral selection mode.

EPOS offers a variety of data input methods to the database. Input can be input or modified either textually or graphically. EPOS supplies it's own text editor to provide for textual input to the system. **It does not provide graphical input capability.** This point is covered in detail in the Diagrammatic/Graphics Facility section. The tool also supports textual input via a file containing EPOS specific input generated by some other process (i.e., a different editor).

The repository (database) provided by EPOS supports a variety of input options and has the ability to enforce

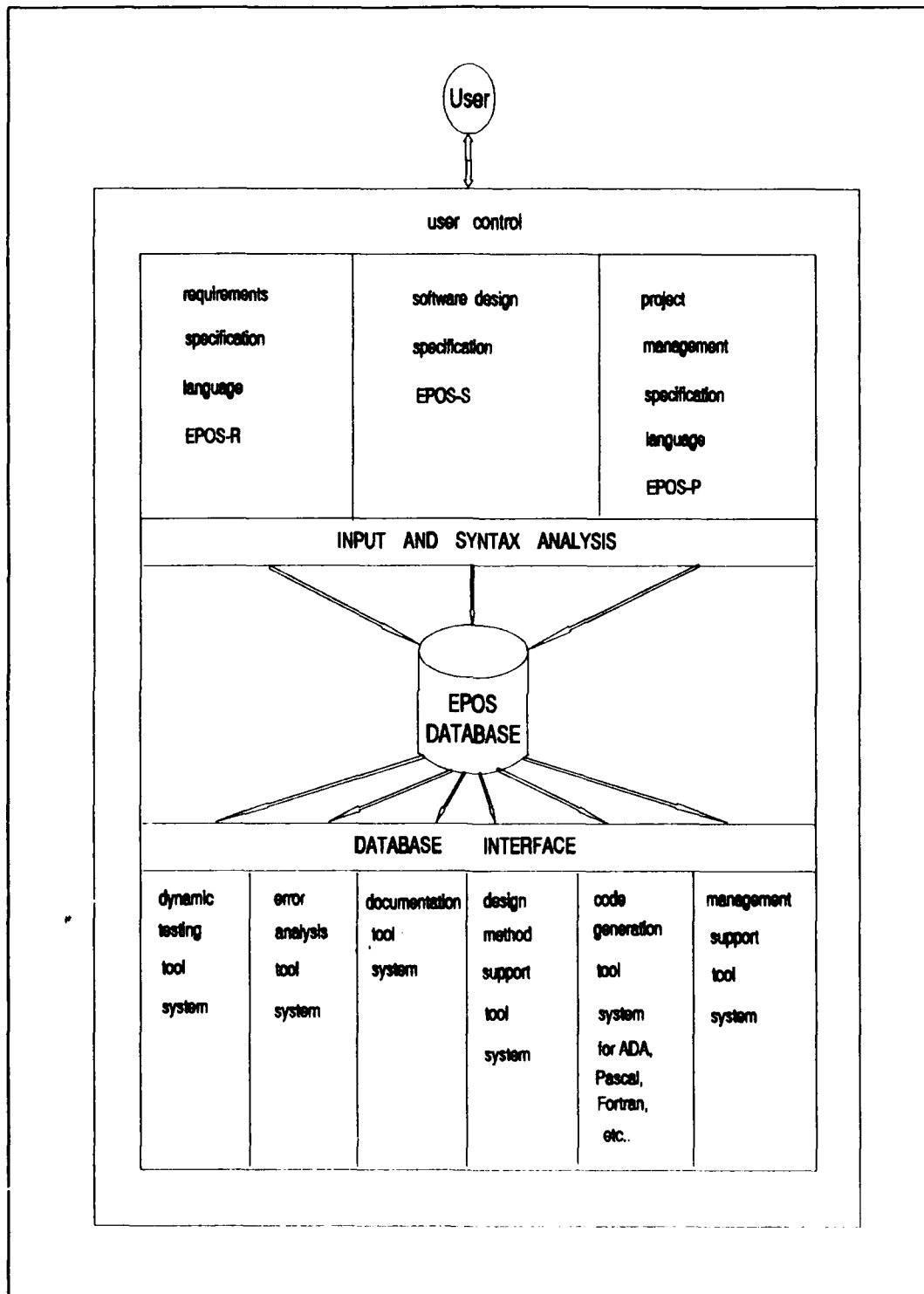


Figure 6-18 Overview of the CASE Environment for EPOS

methodological constraints, but does require a great deal of user manipulation to setup, control and transfer project data.

14. Prototyping

EPOS does not offer any prototyping capability. As a primarily real time application oriented system, EPOS focuses great concern on timing constraint issues. EPOS has a well established track record for the design approach utilized and, therefore, chooses to ignore prototyping efforts.

Although it ignores prototyping efforts, EPOS provides code generation capabilities which greatly accelerate the design to implementation process. The tool includes automatic code generation facilities for Ada, Pascal and Fortran with support for C in development.

It should be noted these facilities do not provide 100% code generation capability. The Ada generator normally generates 75% to 85% of an project's Ada code with the remainder of the code left to manual methods.

EPOS does offer a unique "code feedback" feature for both Ada and Pascal to help maintain consistency between the source file (code) generated and the design specification. For instance, any changes made to the source code during testing result in inconsistencies and differences between the source code and the design specification it was generated from. Code feedback attempts to maintain a link between the

source code and the design specification by allowing the design specification to be changed as well as the source code.

The feedback system provides for partially automatic reconciliation of discrepancies between code changes and the original design specifications. A completely automatic actualization of the system design is not possible since all parts of the design may not necessarily be present in the feedback of the modified code. The manual notes that changes in the source code can reach such a degree of complexity that feedback is virtually impossible. Hence, some specification updates must be done by the user.

Code feedback helps to ensure consistency between design specifications, documentation and source programs. This feature can be a very powerful tool in the arsenal of the user.

15. Consistency/Completeness Checking

Several techniques for ensuring consistency and completeness have already been described in earlier sections. The lexicon feature which is used to standardize terms used throughout the specifications, the REFERENCE attribute used to target a term in the dictionary which must be matched with a design object of the same name, the edit mask (template) in EPOS-S used to declare design elements and the Reference Analysis are prime examples.

In addition to the techniques previously cited, there are other formal mechanisms supplied by the tool. The EPOS-A component includes syntax checks conducted automatically upon entry of EPOS-R and EPOS-S specifications and contains various analyses which can be performed via the EPOS-C component. For example, the EPOS-A provides the following analyses for EPOS-R descriptions:

Syntax check

The word and sentence structure of a description is formally analyzed

Format check

The formatting commands are checked for correct construction

Consistency check

Identifiers (identifiable requirement components, section numbers, etc...) are checked for consistent specification

Decision process analysis

Decision processes are checked for redundancy, unambiguity, consistency and completeness

Reference analysis

References to terms in the dictionary (lexicon) and to identifiable requirement components are analyzed

The checks are automatically performed when the description is entered in the database. The tool notifies the user if an error is present along with specific remarks and prompts for a correction. It will not release the entry to the database until all errors are resolved. The analyses are user initiated. They generate detailed reports such as the requirements traceability document produced by the Reference Analysis.

These are not all the checks and analyses provided by the tool. Similar capabilities exist for the EPOS-S specification as well. These mechanisms are powerful and useful. Analysis reports can be sent to either the screen for quick viewing or the printer for a more in-depth review.

16. Training Support

The documentation and literature provided does not identify specific training support available. Conversations with company representatives indicated training and consulting support is available. Training offered ranges from how to use the system to actual construction of projects via exercises. Training can be provided in-house at SPS or on-site depending on user preference. Representatives pointed out that training can be expensive.

The documentation does identify EPOS as a well established product with a considerable user base noting EPOS has been applied in a variety of applications worldwide. It also identified several support services provided. Support services include: technical hotline support, quarterly publication and user conferences and the EPOS User Group.

The **EPOS User Group** meets **annually** to facilitate the exchange of experiences between the various EPOS users. EPOS-Info, the quarterly publication, provides current reports on new EPOS applications, implementation details and the status of further enhancements. The technical hotline is available

and does provide support, but the number provided is not toll free.

Although the training support is not well documented, the tool does appear to be well established with a considerable user base. Hotline support is available, but at user expense.

17. Diagramming/Graphic Facilities

The graphical capability provided by the PC version of this tool itself is very limited. **It does not provide any graphical input capability**, but it can support it via EPOS's Graphic Editor. The editor relies on an outside product called AutoCAD to provide graphical input capability (See Interface to Other Product section).

EPOS primarily relies on textual based input to enter objects into the specification languages. The tool can generate graphical designs based on text input and display them in color. As noted in the User-Interface section, the graphics mode is the only time the tool supports color operation, but only two colors are provided: red and green. The tool does not provide any edit capability though. User operations are primarily limited to shifting and scaling the diagrams for review purposes.

The Graphic Editor supplied with the tool can support graphical input and provide edit capability if a graphics package is provided. The manual indicates the editor will

work with the basic AutoCAD package (Version 2.5 and up). No special knowledge is required to use the Graphic Editor itself. The Graphic Editor manual supplies all information needed for use. Users will require special knowledge to utilize AutoCAD and must rely on AutoCAD's user manual for support. The editor is provided to allow users to enter and edit input via designs instead of the tool's text specification entry mode.

D. OVERALL EVALUATION

The assorted features and capabilities offered by these tools greatly increased the complexity of their evaluation. The tools evaluated have demonstrated a wide range of significant capabilities for a variety of hardware platforms. Moreover, they offer interfaces to numerous other products which serve to enhance their overall support and enable full lifecycle coverage.

Although these tools provide significant advantages to software development efforts, they impose constraints as well. The learning curve on these tools is formidable. Users must be willing to invest a significant of time in order to be able to fully employ the capabilities provided by a tool. Documentation alone is not enough. Users cannot rely solely on the documentation provided to learn the tool. Some operations may not be covered to the depth required by the user or the manuals may contain errors. Moreover, the tools

themselves are software which implies that no tool can guarantee perfect performance. In fact, tool vendors try to identify known anomalies as they are discovered. Therefore, training and vendor support are vital to tool introduction efforts and become crucial considerations for evaluation. This reliance on the vendor emphasizes the need to evaluate the vendor's history as well as the tool's.

The evaluations provided here are not intended to endorse nor denounce a particular product. They are an attempt to identify the major capabilities and limitations of each tool for the target audience.

E. SUMMARY

Each tool evaluated offers it's own particular advantages to the user. Excelerator/IS focuses on automating the early phases of system development concentrating on analyzing and defining the application problem and the system specification. It also offers several interface utilities to specific products for increased coverage. StP 4.2A offers full lifecycle support emphasizing the analysis and design phases of system development supported by powerful graphical facilities. StP possesses a very useful open architecture which allows users to write (customize) interfaces to other products to provide extensive lifecycle coverage. EPOS 4.0 provides fully integrated lifecycle coverage incorporating extensive project management capabilities. EPOS contains

routines which facilitates the exchange of data with other products.

The tools presented offer a variety of application support. Excelerator/IS can be used to support both large and small business applications. StP 4.2A offers support for a wide range of application domains ranging from information systems to real time embedded systems. EPOS 4.0 supports both software and hardware design across a broad range of application domains, but is primarily oriented towards medium to large real time applications.

All tools are well established products. Excelerator is the leading PC based tool used throughout industry with a very well established user group. StP is a leading workstation tool with an extensive array of third party vendor support in addition to a well established user group. EPOS has been used extensively throughout the world and has been successfully applied in a wide range of medium and large-scale industrial and defense projects.

Multi-user support tended to be the major weakness of the tools. Both PC products, Excelerator/IS 1.9 and EPOS 4.0, do not support multi-user access. Although they can operate in a network environment, the PC products require multiple copies of the tool to be created to support each individual user vice one copy of the tool supporting various users. The workstation product, StP 4.2A, provides extensive multi-user support.

VII. CONCLUSIONS

The objective of this thesis was to identify an evaluation process and provide a general taxonomy of CASE tools from the point of view of potential DoD users. Chapter II describes the evolution of CASE from individual tools to a complete environment of tools. Chapter III examined the extensive impact of DoD's standard for the development of software and the evolution of tools supporting it. Chapter IV proposed a general classification scheme for CASE tools for use with the tool evaluation process identified in Chapter V. In addition, several vendor and institution tools were surveyed and three well established commercial tools were evaluated in Chapter VI.

CASE has evolved from an individual tool or set of tools for software development to a systems approach to software development. The implication of a systems approach necessitates a CASE environment encompassing the organization as well as all aspects of software development. Thus, methodology and management control play crucial roles in the successful introduction and application of CASE. The key feature of the total or full CASE environment in the future will be the linkage between tools, systems and management controls. Therefore, integration is an essential element in

the ability of the full CASE environment to obtain an optimum tool mix or approach for a particular application requirement.

DoD STD-2167A provides a comprehensive framework for the software development process. It imposes stringent documentation and traceability requirements throughout the entire process. These and other requirements within the standard are particularly suited for automation and are beginning to receive considerable support by many current CASE tools.

The tremendous variety and proliferation of tools available today necessitate an orderly, structured approach to their evaluation. To understand what a tool does and compare it to similar tools is a formidable task given the existing diversity of functionality. The evaluation process and its associated checklist along with the proposed taxonomy provide a basis for an organization to assess the true fit of a tool. They are designed so that organizations can tailor them to fit their own individual needs. The proposed taxonomy allows organizations to quickly discern the capabilities and limitations of a tool while the evaluation process assesses how well the tool does its job from the organization's perspective which is the most important perspective to consider.

The tools evaluated offer significant advantages to software development efforts, but they impose constraints as well. The learning curve of these tools is formidable. A

significant investment of time and resources is required if an organization intends to employ the full capabilities offered by a tool. Organizations cannot rely solely on the documentation provided with a tool to be able to fully learn it. Thus, training and vendor support are vital to tool introduction efforts and represent crucial considerations for evaluation. This reliance on the vendor emphasizes the need to evaluate the vendor as well as the tool itself.

Two basic approaches were identified for accomplishing the full CASE environment: combining various toolkits via a framework (C-CASE) or using a workbench (I-CASE) if the environment is limited to one application. Regardless of the approach adopted, the decision to implement CASE within an organization requires a careful, deliberate evaluation process and a total commitment to its use by the entire organization, especially management, if CASE is to be introduced and applied successfully. Thus, the decision for CASE does not just involve evaluating tools, but the organization itself as well.

COMMENTS:

This section should include amplifying information or other information not reflected by the taxonomy.

- * Special features that are especially noteworthy
- * Comments or notes on user interfaces (i.e. is interface consistent throughout all tool efforts)
- * Constraints on the tool (i.e. works only for the Ada programming language)
- * Other required hardware
- * Other required software
- * Basis for classification: vendor supplied, tool user, review of brochures or product documentation.
- * Tool history
- * Other

APPENDIX B

Tool Taxonomy

Tool Name: Information

Engineering Facility (IEF)

Version/Release: 4.0

Vendor/Supplier: Texas

Instruments Plano, Texas

Integration Level: I-CASE

Application Areas: Business

(Supports Real-Time Aspects)

Description: Designed to
automate the complete systems
lifecycle. Consists of

mainframe Encyclopedia and

PC-based graphical toolset.

Project Management	
System/Sware Reqr's Analysis	*
Software Requirements Analysis	*
Preliminary Design	*
Detailed Design	*
Coding and Unit Testing	*
CSC Integration and Testing	
CSCI Testing	
Other	*

ATTRIBUTES:

Methodology/Diagramming Technique Supported: Information Engineering (James Martin) **Techniques:** Entity Relationship Diagram, Entity Hierarchy Diagram, Process Hierarchy Diagram, Process Dependency Diagram, Process Action Diagram, Structure Charts, Dialog Flow Diagram

Hardware Systems Supported: All IBM mainframe & plug compatibles. Typical PC workstation {IBM PC/AT or PS/2 (model 50 or above) with at least 640k RAM}.

Operating Systems Required: PC-DOS or MS-DOS version 3.0 or up

Price Range: High => \$340,000

Other tool interfaces: Provides import/export capability, but no specific tool support was identified.

Lifecycle Supported: Waterfall

Code Generation: Automatically generates 100% complete VS COBOL II programs, DB2 databases, 3270 or MFS interface screens and JCL for batch applications. Generates direct from specification (no source code needed).

Documentation Support: Provides documentation support, but does not specifically support 2167A requirements.

Prototyping: Provides screen design and template facilities; screens can be chained together to simulate system action.

Languages Supported: COBOL

Design:

Multi-user

Networkable

Project Management: Capabilities include: machine-readable task lists, estimating guidelines, function point counting and interfaces to specific products (not identified).

Testing: Accomplished at specification level prior to code generation. (No specific code testing supported)

COMMENTS:

* All applications developed are intended for mainframe use.

* Front-end or early project development (i.e., planning, analysis and design tools) is designed to be accomplished by using PC's while the Back-end (i.e., construction and testing tools) development is designed to be accomplished via the mainframe.

* Company is developing an OS-2 version which will allow users to **develop the entire application** in a PC environment and then port the application to the mainframe.

* Maintenance efforts are accomplished by maintaining or modifying the design specification. (No source code management required)

* Interface does not provide windowing capabilities.

Tool Name: MicroSTEP
Version/Release: 1.4
Vendor/Supplier: Syscorp
International Austin, Texas
Integration Level: I-CASE
Application Areas: Business
(Database & Data Processing)
Description: PC-based tool
with graphical user interface
designed for developing and
prototyping transaction-based
end user systems. (payroll,
accounting, inventory, etc..)

Project Management	System/Share Reqr's Analysis	Software Requirements Analysis	Preliminary Design	Detailed Design	Coding and Unit Testing	CSC Integration and Testing	CSCI Testing	Other
		*	*	*	*			*

ATTRIBUTES:

Methodology/Diagramming Technique Supported: Syscorp
Technique: Flow Diagram

Hardware Systems Supported: PC with at least 564K of memory; requires a mouse and minimum of EGA monitor support. Can use extended memory, if available.

Software Systems Supported:

Price Range: Low => 5k

Other tool interfaces: None

Languages Supported: C (generates C code)

Lifecycle Supported: Evolutionary, RIP (Proprietary)

Code generation: Generates 100% of application code direct from design specification. (No source code required)

Documentation Support: Provides documentation support, but does not specifically support 2167A requirements.

Design:

Prototyping: Accomplished via the graphical interface. Contains a screen format builder which can be used to construct data entry screens and reports.

Testing: Accomplished at specification level prior to code generation. (No specific code testing supported)

COMMENTS:

* Can support a variety of data processing and management information applications (i.e., point of sale, etc.). An application consists of specifications, which correspond to program modules.

* Maintenance efforts are accomplished by maintaining or modifying the design specification. (No source code management required)

* Produces and compiles C code and links object files. Produces a PC program and Dbase compatible files.

* Provides automatic specification analysis.

* Utilizes a graphical specification user interface. Tool has a uniform menu system.

* Interface provides windowing capability.

* Version 1.5 is currently under development. 1.5 will be able to use up to 16 megabytes of expanded memory and also intends to support Novell and IBM Token Ring network application development. PC's running MicroSTEP network applications will require more than 640k and record locking will be limited by the dBASE file structure.

Tool Name: Refine
Version/Release: 3.0
Vendor/Supplier: Reasoning Systems Inc. Palo Alto, CA
Integration Level I-CASE
Application Areas: Business, Limited Real-Time
Description: Interactive knowledge-based programming environment. Primarily oriented towards software reengineering efforts.

Project Management	System/Share Reqr's Analysis	Software Requirements Analysis	Preliminary Design	Detailed Design	Coding and Unit Testing	CSC Integration and Testing	CSCI Testing	Other
	*	*	*	*	*			*

ATTRIBUTES:

Methodology/Diagramming Technique Supported: None

Hardware Systems Supported: Sun 3/4/Sparcstation; Symbolics; Macintosh II; Texas Instruments Explorer

Software Systems Supported: Sun 3.5, 4.0

Price Range: Medium => 10k - 15k

Other tool interfaces: The Project Management Assistant (Provides project management capabilities [i.e. Gantt charts, etc..] and 2167A documentation support). This tool is a product of Kestrel Institute in Palo Alto, CA.

Languages Supported: Refine (Proprietary), C, Ada, LISP. Generates fully executable Refine and LISP code. Commercial compilers for both C and Ada are currently under development.

Lifecycle Supported: Transform

Multi-user

Networkable

Reengineering: Supports the transformation of Ada, C, COBOL, Fortran and PL1. Tool can analyze source code and transform into new optimized source code.

Simulation: Supported via the construction of high level specifications (generate structure i.e., object class, etc..) which are fully executable.

Code Generator: Supports both batch and on-line code generation; can generate a production program. (See language supported section).

Testing: Accomplished at specification level prior to code generation.

COMMENTS:

* Applications are developed using a very high level specification language (Refine). The language integrates advanced specification techniques, including first-order logic, set theory, transformation rules and pattern matching.

* System can be customized to create knowledge-based environments in which Refine tools are tailored and extended for use in specific application areas. (Tool is written in Refine and can be modified via Refine).

* Supports the re-use of general purpose and domain specific knowledge in the form of rules, object-oriented programming and logic formulas.

* Interface does provide windowing capability.

* Contains tool for building graphical interfaces.

Tool Name: Serpent
Version/Release: Rel .8
Vendor/Supplier: SEI
Carnegie-Mellon University
Integration Level: P-CASE
Application Areas: Real-Time
and Business
Description: User Interface
Management System. Used for
developing software systems
with complex user interfaces
that often change.

Project Management	
System/Sware Reqr's Analysis	
Software Requirements Analysis	*
Preliminary Design	*
Detailed Design	*
Coding and Unit Testing	*
CSC Integration and Testing	
CSCI Testing	
Other	

ATTRIBUTES:

Methodology/Diagramming Technique Supported: None

Hardware Systems Supported: Sun Systems (Work/Sparcstations), VAX Systems (Variety), DEC Systems and HP-9000

Software Systems Supported:

Price Range: Available free to all DoD activities.

Other tool interfaces: None (See comments)

Languages Supported: Ada, C (Applications can be written in Ada or C); Slang (special language used for Serpent user interface specification).

Lifecycle Supported: Evolutionary, Spiral

Prototyping: Reasonably sophisticated interface prototypes can be generated.

COMMENTS:

- * Makes user interfaces easier to specify, thereby aiding requirements identification.

- * Supports incremental development of user interfaces via prototyping capability.

- * Provides a "bridge" between prototype and production versions of system.

- * Designed to simplify the integration of I/O media. Supports multiple I/O media for user interaction and the insertion new I/O media. Supports X-windows facilities and both Open Look and Motif look and feel guidelines.

- * **Applications view Serpent as similar to a database management system.** Serpent can be used to develop the user interface portion of an application written in C or Ada. Serpent uses **Slang** (user interface specification language) to compile and generate an executable program. Serpent provides interfaces to C and Ada which allow the application to communicate with the Serpent executable program via a dialogue layer. It functions similar to a runtime version of a database program.

- * Tool provides Serpent Editor supporting both textual and graphical entry. Layouts of user interface can be specified or examined graphically. Logic, dependencies and calculations can be specified textually.

APPENDIX C

Blank Tool Taxonomy Form

Tool Name: _____

Version/Release: _____

Vendor/Supplier: _____

Integration Level: _____

Application Areas: _____

Description: _____

Project Management	System/Sware Reqr's Analysis	Software Requirements Analysis	Preliminary Design	Detailed Design	Coding and Unit Testing	CSC Integration and Testing	CSCI Testing	Other

ATTRIBUTES:

Methodology/Diagramming Technique Supported: _____

Hardware Systems Supported: _____

Software Systems Supported: _____

Price Range: _____

Other tool interfaces: _____

COMMENTS:

*
*
*
*
*
*

APPENDIX D

Tool Evaluation Checklist

A taxonomy is not an evaluation. The former assigns a tool a place in a classification matrix to give an indication of what the tool does and where it is used. The latter attempts to assess how well the tool does its job relative to the needs of the evaluator.

Such evaluations are inevitably somewhat subjective since everyone has different requirements. works in a different environment, and has different ideas about how tools ought to work. However, many questions that a potential user asks about a tool can be standardized. while accepting that different users will interpret the answers in different ways and attach different degrees of importance to them.

The following sections discuss these questions. grouped according to the aspect of a tools acquisition, support, and performance they address. These aspects are:

1. Ease of Use
2. Power
3. Robustness
4. Functionality
5. Ease of Insertion
6. Quality of Commercial Support

The first four sections are mainly of concern to the actual user of the tool; the last two are of concern to the management of the project that contemplates acquiring the tool. Each question is phrased such that a positive response indicates a positive tool attribute.

1. Ease of Use

One measure of a tool's effectiveness is the ease with which the user can interact with it. Clearly, no matter how functional or complete a tool is, if the user spends most time thinking about how to use the tool or making the tool work, then the tool is hindering and not helping with the task. To justify using a tool, the tool's benefits must offset its cost and the time spent using it.

1.1. Tailoring

Tailoring is an important aspect of a tool. A tool can be used by a wide variety of organizations and users. If a tool can be tailored to user needs or to a particular user style, the tool has the potential to be used with more dexterity and at a faster rate than would be otherwise. While tailoring can provide positive benefits, it is important to recognize that indiscriminate tailoring can disrupt team efforts when each user tailors the tool to an individual style.

1. Can various aspects of the interface be tailored to suit user needs, including application and ability level?
2. Can the user define new commands or macros for commonly used command sequences or chain macros together?
3. Can the user "turn off" unwanted functions that might be obtrusive?
4. Can the tool's input and output formats be redefined by the user?
5. Can tailoring operations be controlled to maintain consistency within the using project/organization?
6. Can the tool be configured by the user for different resource tradeoffs to optimize such things as response speed, disk storage space, and memory utilization?
7. Does the vendor support and assist tailoring the tool to the specific users needs?

1.2. Intelligence/Helpfulness

A tool helps the user by performing particular functions. The more intelligent a tool, the more functions it will perform without the user having to directly specify their initiation. In addition, the tool should anticipate user interaction and provide simple and efficient means for executing functions the user requires.

1. Is the tool interactive, for example. does it prompt for command parameters, complete command strings, or check for command errors?
2. Is action initiation and control left with the user?

3. Is quick, meaningful feedback on system status and progress of interaction and execution given to the user?
4. Is the interface simplified by the use of sound or graphics (icons, color coding, shape, texture, etc..)?
5. Can the user access and retrieve stored information quickly and with little effort while using the system?

1.3. Predictability

Unpredicted responses from the tool usually result in unhappy users and unwanted output. Command names should suggest function, and users should rarely be surprised by a tool's response. If an unpredicted response does occur, the user should have a means to "undo" the command. If the result of a particular command has drastic results, the user should be warned before the command is actually executed.

1. Are the responses from the tool expected in most cases?
2. Is it possible to predict the response of the tool to different types of error conditions?

1.4. Error Handling

Not only should the tool be tolerant of user errors, it should check for and correct these errors whenever possible.

1. Does the tool recover from errors easily?
2. Does the tool protect the user from costly errors?
3. Does the tool periodically save intermediate objects to ensure that all work is not in vain if a failure occurs during a long session of tool use?
4. Does the tool protect against damage to its database caused by inadvertent execution of the tool?
5. Does the tool help the user correct errors?
6. Will the tool check for application-specific errors, such as checking if parentheses match?

1.5. System Interface

Not only is it useful for a tool to interact with one user, but it may be appropriate for a tool to accommodate interaction with many users or other tools.

1. Is the tool designed to be used by more than one person at a time?
2. Does the tool provide for management, including access control, of work products for single and multiple users?
3. Is the interface compatible with other tools in a tool set or other commercially available tools?
4. Does the tool provide for output devices such as printers?
5. Does the tool require a particular output device?

2. Power

What does one mean by the power of a tool? Here are some examples concerning a tool that nearly everyone uses, a text editor. A powerful editor can, for instance:

- globally replace 'HAL" with 'DEC"
- recognize 'love' and 'love?' as two instances of the same word, but correctly recognize "glove' as another word
- warn that "necessary" looks wrong
- automatically indent paragraphs, inserting new lines between words at the appropriate points
- automatically number paragraphs or sections, renumbering after insertion or deletion.

The power of a tool seems to arise from two main sources:

- the extent to which the tool "understands" the objects it is manipulating
- the extent to which simple commands can cause major effects.

In addition, a tool can give the impression of greater power by keeping more knowledge about its internal state such

as a command history. Power is also demonstrated by reasonable performance achieved through efficient use of the computing resource.

2.1. Tool Understanding

The objects that a tool manipulates have an inner structure. Structured objects tend to be comprehended in terms of two things: the framework and the particular content. For example, $I := J$ is read as an assignment statement (framework) that assigns J to I (content). This text is being read as the introduction (framework) to a section that explains tool understanding (content).

Hence, questions can be asked about the extent to which a tool understands the structure and its content, and also about the ability of the tool to handle more detailed or more general aspects of that structure.

1. Does the tool operate on objects at different levels of abstraction or at different levels of detail?
2. Can one zoom in or zoom out from one level to another?
3. Can the tool modify collections of objects so as to preserve relationships between them?
4. Can the tool remove objects and repair the structure, and insert objects with proper changes to the new structure?
5. Does the tool perform any validation of objects or structures?
6. Can the tool automatically create structural templates?
7. Can the tool expand objects into templates of the next level of detail, with consistency checking between levels?

2.2. Tool Leverage

Leverage is the extent to which small actions by the user create large effects. The leverage of any interactive tool is a function of its command set. The usual way to increase this leverage is to allow a user to define *macros* - short commands that stand for longer command sequences. Another way, more in keeping with an object-oriented view of the world, is to define a command as an action to be applied to a specific object. Commands can then be "overloaded," i.e., the same

command name can have a different implementation for different objects. Commands can also be inherited, composed, and so on.

To illustrate this, consider a command `print` applied to a fragment of a parse tree. One print style can be defined for expressions and another for comments (overloading). If an attempt is then made to print a commented expression, the right thing (composition) should be obtained automatically.

If this facility is missing, a tool can be made to do more only by multiplying the number of commands. e.g., having a `printcomment` command and a `printtree` command. Most people would agree that this doesn't make a tool more powerful; the increase in the number of things it can do is matched by a corresponding increase in the effort the user has to expend to learn, remember, and select the commands. Indeed, since such extensions to the command set typically address more and more marginal areas of the requirement, "creeping featurism" dilutes the power of the tool.

1. Can commands be bound to specific object types or structure templates?
2. Can commands be applied systematically to entire collections of similar objects?
3. Can polymorphic commands be applied to entire structures that contain diverse objects?
4. Can commands be executed indefinitely until a predicate is satisfied?

2.3. Tool State

This is an inductive approach to increasing the power of a tool. If a tool remembers how it has been used in a current session or in previous sessions, it can provide the user with simpler ways of invoking effects by saying, "Do to this what you just did to that."

1. Does the tool keep a command history?
2. Can commands be reinvoked from the history?
3. Can the command history be saved to be replayed by a new run of the tool?
4. Can the reinvoked commands be modified when they are replayed?

5. Can one save the current state of the tool and the objects it is manipulating, and subsequently restore it?
6. Does the tool learn, i.e., does it keep state across invocations?
7. Does the tool keep and/or employ statistics of command frequency and operand frequency?

2.4. Performance

The performance of a tool can greatly affect the ease with which it is used and can ultimately determine the success of a tool within an organization. A tool must be able to function efficiently and be responsive to the user. Poor tool performance can create costs that negate many of the benefits realized from tool use; a tool that performs inefficiently may result in missed schedules or frustrated users who are skeptical that the tool really helps them.

1. Is the tool's response to commands acceptable relative to the complexity of the operations performed by the command? For example, is the user waiting for unreasonable amounts of time, or is there any response lag on simple or frequently used commands?
2. If the tool supports multiple users, is response and command execution time acceptable with the maximum load of users?
3. Can the tool, running on the user's hardware, handle a development task of the size required by the user?
4. Does the tool provide a mechanism to dispose of any useless byproducts it generates?

3. Robustness

This section is concerned with the robustness of the tools operating on a system. The robustness of a tool is a combination of such factors as: the reliability of the tool, the performance of the tool under failure conditions, the criticality of the consequences of tool failures, the consistency of the tool operations, and the way in which a tool is integrated into the environment.

While the robustness of individual tools is important, it is secondary to the robustness of the environment in which the tools operate. Although the tool and the tool set of which it

is part can be robust and consistent, many characteristics of robust operation are best done on a more global environment where the tool writer has to worry about correct interfaces to the environment, but does not have to be concerned with a great many services that are provided by the environment to maintain system integrity. For example, most tools should not be concerned with security issues, access authorization, archiving, device interfaces, etc.. These should be handled by the environment in which they are embedded. The tool should be concerned with having the correct interfaces to be inserted in the environment and to operate properly within the environment. The issues described in the following sections are tool-related robustness issues.

3.1. Consistency

These issues are concerned with the consistency of operation of the tool.

1. Does the tool have well-defined syntax and semantics?
2. Can the output of the tool be archived and selectively retrieved and accessed?
3. Can the tool operate in a system with a unique identification for each object?
4. Can the tool re-derive a deleted unique object or does the re-derivation create a new unique object?
5. Does the tool have a strategy for dealing with re-derivation of existing objects, such that it finds the objects rather than re-deriving them? (This has important consequences on the performance characteristics of the system.)

3.2. Evolution

In all but the most unusual cases, tools evolve over time to accommodate changing requirements, changes to the environment, correcting detected flaws, and performance enhancements. The questions below are related to the evolution of the tool.

1. Is the tool built in such a way that it can evolve and retain compatibility between versions?
2. Can the tool smoothly accommodate changes to the environment in which it operates?

3. Can new versions of the tool interface with old versions of other related tools?
4. Can new versions of the tool operate correctly on old versions of target objects?
5. Can old versions of the tool operate correctly on new versions of the target objects?
6. Can separate versions of the tool coexist operationally on the system?
7. Has the tool been implemented on/porting to various hosts?
8. Can the tool's output be interchanged between hosts?

3.3. Fault Tolerance

There are many ways of defining fault tolerance. This work is not concerned with the general problem, but with fault tolerance that specifically is related to individual tools.

1. Does the tool have a well-defined atomicity of action? (**Note:** This does not necessarily mean that each invocation of the tool must have an atomic effect on the system. It simply means that no intermediate states should be registered, and that any environmental failures during execution of the tool do not cause irreparable damage once the failure has been repaired and the system restarted.)
2. If the tool is found to be incorrect, can the system be rolled back to remove the effects of its incorrect actions?

3.4. Self-Instrumented

A tool is a piece of software performing a function and, like all other software, may have various types of bugs or flaws associated with it at any point in its life cycle. Most bugs are detected during testing and deployment, but there are often latent bugs remaining after deployment, and maintenance activities are well known to introduce bugs. For these reasons, a tool must self-instrumented to assist in determining the cause of a problem once the symptom has been detected.

1. Does the tool contain any instrumentation to allow for ease of debugging?

2. Are there tools for analyzing the results collected by the instrumentation?
3. Does the tool contain self-test mechanisms to ensure that it is working properly?
4. Does the tool record, maintain, or employ failure records?

4. Functionality

The functionality of a tool is not only driven by the task that the tool is designed to perform but also by the methods used to accomplish that task. Many tools support methodologies. The accuracy and efficiency with which the tool does this can directly affect the understandability, and performance of the tool, as well as determine the quality and usefulness of tool outputs. In addition, a tool that generates incorrect outputs can lead to frustrated users and extra expenditures needed to "fix" tool outputs. These additional costs may weigh heavily against tool benefits.

4.1. Methodology Support

A methodology is a systematic approach to solving a problem. It prescribes a set of steps and work products as well as rules to guide the production and analysis of the work products. Automated support for a methodology can aid its use and effectiveness. However, it must be made clear that the following questions *do not deal with assessing a particular methodology*. Methodology assessment should occur separately from and prior to the assessment of tools that support the methodology. The questions presented here deal with how well a tool automates and supports a methodology, not with the methodology itself.

1. Does the tool support one or more methodologies that meet the users' needs?
2. Does the tool provide a means to integrate other methodologies?
3. Does the tool support all aspects of the methodology?
4. If some aspects are excluded, are the important parts or concepts of the methodology (parts that are either important to the methodology itself or important to the development project) supported?

5. Does the tool support the communication mechanisms of the methodology (such as a textual or graphical language) without alteration?
6. Does the tool build in any functionality, in addition to direct support of the methodology, that is useful?
7. Is the tool free of functionality that is useless or a hinderance?
8. Does the tool flexibly support the methodology, for example can the user initially skip or exclude some parts of the methodology and return to it later?
9. Does the tool provide an adequate scheme to store, organize, and manipulate the products of the application of the methodology?
10. Does the tool provide guidance to ensure that the concepts of the methodology are followed when using the tool?

4.2. Correctness

To be useful, a tool must operate correctly and produce correct outputs. A tool evaluation must pay special attention to this critical area.

1. Does the tool generate output that is consistent with what is dictated by the methodology?
2. Does the tool check to see if the methodology is being executed correctly?
3. Is there no case where data items entered by the user are unintentionally or unexpectedly altered by the tool?
4. Are executable outputs generated by the tool "bug free"?
5. Are outputs generated by the tool correct by all standards?
6. Do transformations executed by the tool always generate correct results?

5. Ease of Insertion

An important aspect of tool use that is often overlooked is the ease with which a tool can be incorporated into the

target organization or environment. Management and users need to be aware of how well the tool fits within the existing environment and accept changes that the tool may inflict upon the environment in which they work. Questions on ease of insertion fall into the categories listed below.

5.1. Learnability

Depending upon how complex it is, learning how to use a tool can result in considerable expense, time, and frustration. Not only should the tool's command set be consistent and understandable, the tool should interact with the user to help learn to use the tool properly.

1. Is the complexity of the tool proportional to the complexity application; i.e., does the tool simplify a problem rather than complicate it?
2. Do prospective tool users have the background necessary to successfully use the tool?
3. Can the users use the tool without memorizing an inordinate number of commands?
4. Do the interactive elements imply function in the problem domain. i.e., do command names suggest function or graphical symbols representative of function?
5. Are the commands and command sequences consistent throughout the system?
6. Can the user quickly do something to see what happens and evaluate results without a long set-up process?
7. Can the results, i.e., the work products produced, of learning exercises be disposed of easily? For example, can they be removed from a database without action by a database administrator?
8. Is the tool based on a small number of easy to understand/learn concepts that are clearly explained?
9. Does the tool provide a small number of functions (commands, directives) that allow the user to do the work the tool is intended to do?
10. Can the user learn a small number of simple commands initially and gradually add more advanced commands as proficiency is developed?

11. Does the tool provide the user with templates or other aids to guide interaction?
12. Is there a method of using a help facility that aids the novice user by providing a step-by-step description of what to do?
13. Is the time required to understand and become proficient in using the tool acceptable:
 - for the average user?
 - for the average project manager?
 - for the project team?

5.2. Software Engineering Environment

Successful use of a tool requires a fit between the tool and the environment in which it will be used.

1. Is the tool in some ways similar to what the organization currently does and knows, for example, is there some commonality in the underlying method, process, vocabulary, notation, etc?
2. Is the command set free of conflict with the command set of other tools the organization uses. i.e., same or similar commands with different actions?
3. Does the tool run on the hardware/operating system (O/S) the organization currently uses?
4. Is installing the tool a simple, straightforward process?
5. Does the tool use file structures/databases similar to those currently in use?
6. Can data be interchanged between the tool and other tools currently employed by the organization?
7. Can the tool be cost-effectively supported by those responsible for maintaining the environment?

6. Quality of Support

Without adequate commercial support, a tool may become useless before it is used. The quality of commercial support connotes many things: it ranges from the cost of maintenance agreements to the level of training required and provided.

When evaluating a tool, one should also consider its "track record." The evaluator should be aware of the past performance and uses of the tool as well as the past support the vendor has or has not provided.

6.1. Tool History

What is the tool's track record?

1. Does the tool have a history that indicates it is sound and mature?
2. Has the tool been applied in a relevant application domain?
3. Is a complete list of all users that have purchased the tool available?
4. Is it possible to obtain evaluations of the tools from a group of users?

6.2. Vendor History

Often one can infer the quality of the tool and the quality of support for the tool by looking into the track record and reputation of the vendor selling the tool.

1. Is there a staff dedicated to user support?
2. From talking to others who have had experience with the vendor, does the vendor live up to commitments, promises?
3. Are the projections for the future of the company positive, for example, does the company's future appear stable?

6.3. Purchase, Licensing, or Rental Agreement

1. Is the contract or agreement explicit enough so that the customer knows what is or is not being acquired?
2. Is there a cost reduction for the purchase of multiple copies?
3. Is there a corporate site license available?
4. If the user wishes, can the tool be leased?

5. Does the user have the ability to return the tool for full refund during some well-defined reasonable period of time?
6. Is the customer given full rights and access to source code (in the event the vendor goes out of business, no longer supports the tool, and is unable to sell off rights to the product)?
7. Is the user free of all obligations to the vendor regarding use or sale of the objects generated by the tool?

6.4. Maintenance Agreement

1. Does a warranty (written guarantee of the integrity of the product and of the vendors responsibility for the repair or replacement of defective parts) exist for the tool?
2. Can the user purchase a maintenance agreement?
3. Can the vendor be held liable for the malfunctioning of the tool?
4. Will maintenance agreements be honored to the customer's satisfaction in the case that the vendors sell out?
5. Is the frequency of releases and/or updates to the tool reasonable (e.g., fast enough to respond to problems, slow enough not to overburden the user with change)?
6. Does the maintenance agreement include copies of releases/updates?
7. Is the turn-around time for problem or bug reports acceptable?

6.5. User's Group/User Feedback

1. Does a user's group (or similar group that addresses problems, enhancements, etc., with the tool) exist?
2. Does the vendor provide a responsive, helpful hot-line service?

6.6. Installation

1. Is the tool delivered promptly as a complete package (object code, documentation, installation procedure, etc..)?
2. Does the vendor provide installation support and consultation?

6.7. Training

1. Is training available?
2. Has prerequisite knowledge for learning and use of the tool been defined?
3. Is training customized for the acquiring organization and individuals with attention paid to the needs of different types of users (engineers, project managers, etc..)?
4. Do the training materials or vehicles allow the user to work independently as permits?
5. Is the User provided with examples and exercises?
6. Are vendor representatives (marketing, sales, service, training) product knowledgeable and trained?

6.8. Documentation

1. Is the tool supported with ample documentation (e.g., installation manuals, user's manuals, maintenance manuals, interface manuals, etc..)?
2. Is on-line help available?
3. Is a tutorial provided?
4. Does the documentation provide a description of what the tool does ("big picture view") before throwing the user into the details of how to use it?
5. Is the documentation:
 - o readable
 - o understandable
 - o complete

- accurate
 - affordable?
6. Does the documentation have an indexing scheme to aid the user in finding answers to specific questions?
 7. Is the documentation promptly and conveniently updated to reflect changes in the implementation of the tool?

LIST OF REFERENCES

1. Barry W. Boehm, "Understanding and Controlling Software Costs," in *Journal of Parametrics*, Vol VIII, No. 1, International Society of Parametric Analysts, 1988.
2. QED Information Sciences, Inc, CASE : The Potential and the Pitfalls, p. 51, Chantico Publishing Company, 1989.
3. CASE Consulting Group, AN INTRODUCTION TO CASE, CASE OUTLOOK, Inc., 1988.
4. Boehm, Barry and Standish, Thomas, "Software Technology in the 1990's: Using an Evolutionary Paradigm," IEEE Computer, Vol. 16, No. 11, November 1983.
5. Martin, James, "The Future of CASE Technology," ShowCASE Conference III, The Center for Study of Data Processing and The School of Technology and Information Management, Washington University in St. Louis, Missouri, St. Louis, Missouri, 19-21 September 1988.
6. Carma McClure, "Proceedings of the Computer-Aided Software Engineering Symposium," in AN INTRODUCTION TO CASE, CASE OUTLOOK, Inc., 1988.
7. Case Consulting Group, "Quick Reference to Computer-Aided Software Engineering," The CASE OUTLOOK, Inc., 1989.
8. Telephone conversation between Tanya Coy, Case Consulting Group and the author, 3 July 1990.
9. CASE Studies Consortium, "From Initial Excitement to Adoption: Pathways to CASE Success," CASExpo-Spring'90 Sheraton Washington, Washington D.C. 2-6 April 1990.
10. Bentley, L.D., and Whitten J.L., *Using Excelerator for Systems Analysis @ Design*, 1st Ed., Times/Mirror/Mosby College Publishing, 1987.
11. Yourdan, Edward, Modern Structured Analysis, Prentice-Hall, Englewood Cliffs, New Jersey, pp. 126-128.
12. Frey, Wayne K., *Computer Aided Software Engineering Issues*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1987.
13. *The Repository's Role in CASE Integration*, CASE OUTLOOK, Vol. 89, No. 4, December 1989.

14. Loh, Marcus and Nelson, Ryan, "Reaping Case Harvests," DATAMATION, Vol. 35, No. 13, July 1, 1989.
15. Hanner, Mark, "CASE TOOLS Productivity for the Masses," DCE Professional, Vol. 7, No. 12, December 1988.
16. Martin, James, "CASE & ICASE," High Productivity Software, Inc., Marblehead, Massachusetts, 1988.
17. Forte, Gene, A Mecca for CASE: All Roads Lead to the Repository, CASE OUTLOOK, Vol. 89, No. 4, December 1989.
18. Wallace, Steve, Methodology: CASE's Critical Cornerstone, Business Software Review, Vol. 7, No. 4, April 1988.
19. T. Capers Jones, The Cost and Value of CASE, CASE OUTLOOK, Portland, OR, Vol. 1, No. 4, October 1987 in An Introduction to CASE, CASE Consulting Group, 1988.
20. Forte, Gene, AD/Cycle (Part 1), CASE OUTLOOK, Vol. 89, No. 4, December 1989.
21. Yeh, Raymond, Specification Compilers: A step towards next generation CASE systems, SYSTEM BUILDER, April/May 1990.
22. Department of Defense Military Standard DoD STD-2167A, DEFENSE SYSTEM SOFTWARE DEVELOPMENT, 4 June 1985.
23. Fisher, Alan, CASE, Using Software Development Tools, John Wiley & Sons, Inc., New York, New York, 1988.
24. Batt, Gary, CASE Technology and the Systems Development Life Cycle: A Proposed Integration of CASE Tools with DoD STD-2167A, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1989.
25. Sherwood, Don, "Analysis and Design Tools," CASExpo-Spring'90, Sheraton Washington, Washington D.C. 2-6 April 1990.
26. Anderson, Leigh, "Supporting Traceability Requirements with CASE," Defense Computing, Vol. 2, No. 4, July-August 1989.
27. Galal, G. and Hall, A., "Computer-aided software engineering," Computer Aided Engineering Journal, Vol. 6, No. 4, August 1989.

28. Chen, Mender, Nunamaker, J.F., and Weber, E.S., *Computer-Aided Software Engineering: Present Status and Future Directions*, CASExpo-Spring'90, Sheraton Washington, Washington D.C. 2-6 April 1990.
29. Hatley, Derek, *CASE tools still not ready to meet the real-time challenge*, COMPUTERWORLD, Vol. XXIII, No. 13, 27 March 1989.
30. Case, Albert F. Jr., "Evaluating and Selecting CASE Tools," CASE Outlook, Vol. 1, No. 1, July 1987.
31. Firth, Robert, Mosely, Vicki, Pethia, Richard, Roberts, Lauren, and Wood William, "A Guide to the Classification and Assessment of Software Engineering Tools," Technical Report, CMU/SEI-87-TR-10, August 1987.
32. Alder, Rudy, "THIS IS THE STSC," CROSSTALK, Software Technology Report, Headquarters Ogden Air Logistics Center, 1 December 1989.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2. Library, Code 52 Naval Postgraduate School Monterey, California 93943-5002	2
3. Computer Technology Curricular Office, Code 37 Naval Postgraduate School Monterey, California 93943-5000	1
4. Commandant of the Marine Corps Code TE 06 Headquarters, U.S. Marine Corps Washington, D.C. 20380-0001	1
5. System Integration Code C20S Attn: Captain James E. Minnema Marine Corps Research Development and Acquisition Command Quantico, Virginia 22134-5080	1
6. Marine Corps Tactical Systems Support Activity Attn: Mr. Vivian Pacus Camp Pendleton, California 92055-5080	1
7. Center for Naval Analysis 4401 Ford Avenue Alexandria, Virginia 22302-0268	1
8. Director of Research Administration Attn: Professor Howard, Code 012 Naval Postgraduate School Monterey, California 93943-5000	1
9. Department of Computer Sciences Professor Luqi, 52Lq Naval Postgraduate School Monterey, California 93943-5000	30
10. Department of Administrative Sciences Professor Tarek Abdel-Hamid, 54Ah Naval Postgraduate School Monterey, California 93943-5000	1

11. GMD-F2G2 1
Potsfach 1240
Attn: Dr. Bernd J. Krämer
5205 Sankt Augustin 1
West Germany
12. Department of Administrative Sciences 1
Professor Tung Bui, 54Bd
Naval Postgraduate School
Monterey, California 93943-5000
13. Software Engineering Institute 1
Attn: Mr. Robert Seacord
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213-3890
14. Index Technology Corporation 1
Attn: Ms. Pamela Meyer
One Main Street
Cambridge, Massachusetts 02142
15. Interactive Development Environments 1
Attn: Mr. Ramana Gogula
595 Market Street, 12th Floor
San Francisco, CA 94105
16. Reasoning Systems Inc. 1
Attn: Mr. Cordell Green
3260 Hillview Avenue
Palo Alto, CA 94304
17. SPS Software Products & Services, Inc. 1
Attn: Ms. Claudia Gamlien
14 East 38th Street, 14th Floor
New York, NY 10016
18. Syscorp International 1
Attn: Mr. Raymond T. Yeh
9420 Research Blvd., Suite 200
Austin, Texas 78759
19. Texas Instruments Incorporated 1
Information Engineering Facility
Attn: Mr. Bob Barker
P.O. Box 869305, MS 8474
6550 Chase Oaks Blvd.
Plano, Tx 75806

20. Chairman, Code 52 1
Computer Science Department
Naval Postgraduate School
Monterey, California 93943-5000
21. Chief of Naval Research 1
800 N. Quincy Street
Arlington, Virginia 22217
22. Ada Joint Program Office 1
OUSDRE (R&AT)
Attn: Dr. John Selicman
The Pentagon
Washington, D.C. 20301
23. Defense Advanced Research Projects Agency (DARPA) 1
Integrated Strategic Technology Office (ITSO)
Attn: Dr. B. Boehm
1400 Wilson Boulevard
Arlington, Virginia 22209-2308
24. Defense Advanced Research Projects Agency (DARPA) 1
Director Naval Technology Office
1400 Wilson Boulevard
Arlington, Virginia 22209-2308
25. Defense Advanced Research Projects Agency (DARPA) 1
Director, Prototype Projects Office
1400 Wilson Boulevard
Arlington, Virginia 22209-2308
26. Defense Advanced Research Projects Agency (DARPA) 1
Director, Tactical Technology Office
1400 Wilson Boulevard
Arlington, Virginia 22209-2308
27. Code K54, NSWC 1
Attn: Mr. William McCoy
Dahlgren, VA 22448
28. Chief of Naval Operations 1
Attn: Dr. R.M. Carroll (OP-01B2)
Washington D.C. 20350
29. USC-Information Sciences Institute 1
4676 Admiralty Way
Suite 1001
Marina del Ray, California 90292-6695

30. Computer Science Department 1
Attn: Dr. Ted Lewis
Oregon State University
Corvallis, Oregon 97331
31. IBM T.J. Watson Research Center 1
Attn: Dr. A. Stoyenko
P.O. Box 704
Yorktown Heights, New York 10598
32. Kestrel Institute 1
Attn: Dr. C. Green
1801 Page Mill Road
Palo Alto, California 94304
33. Department of Computer Sciences 1
Attn: Professor D. Berry
University of California
Los Angeles, California 90024
34. National Science Foundation 1
Division of Computer and Computation Research
Attn: K.C. Tai
Washington, D.C. 20550
35. National Science Foundation 1
Division of Computer and Computation Research
Attn: Tom Keenan
Washington, D.C. 20550
36. Naval Ocean Systems Center 1
Attn: Linwood Sutton, Code 423
San Diego, California 92152-5000
37. Naval Sea Systems Command 1
Attn: CAPT A. Thompson
National Center #2, Suite 7N06
Washington, D.C. 22202
38. NAVSEA, PMS4123H 1
Attn: William Wilder
Arlington, Virginia 22202-5101
39. New Jersey Institute of Technology 1
Computer Science Department
Attn: Dr. Peter Ng
Newark, New Jersey
40. Fleet Combat Directional Systems Support Activity 1
Attn: Dr. Mike Reiley
San Diego, California 92147-5081

41. Office of Naval Research 1
Computer Science Division, Code 1133
Attn: Dr. Van Tilborg
800 N. Quincy Street
Arlington, Virginia 22217-5000
42. Office of Naval Research 1
Computer Science Division, Code 1133
Attn: Dr. R. Wachter
800 N. Quincy Street
Arlington, Virginia 22217-5000
43. Office of Naval Research 1
Applied Mathematics and Computer Science
Attn: Mr. J. Smith, Code 1211
800 N. Quincy Street
Arlington, Virginia 22217-5000
44. Software Group, MCC 1
9430 Research Boulevard
Attn: Dr. L. Belady
Austin, Texas 78759
45. Computer Science Deptment 1
Southern Methodist University
Attn: Dr. Murat Tanik
Dallas, Texas 75275
46. Department of Computer and Information Science 1
The Ohio State University
Attn: Dr. Ming Liu
2036 Neil Ave Mall
Columbus, Ohio 43210-1277
47. U.S. Air Force Systems Command 1
Rome Air Development Center
RADC/COE
Attn: Mr. William E. Rzepka
Griffis Air Force Base, New York 13441-5700
48. Department of Electrical Engineering and Computer 1
Science
Computer Science Division
Attn: Dr. C.V. Ramamoorthy
University of California at Berkeley
Berkeley, California 90024
49. Department of Computer and Information Science 1
Attn: Dr. Nancy Levenson
University of California at Irvine
Irvine, California 92717

50. Department of Computer Science 1
Attn: Dr. William Howden
University of California at San Diego
La Jolla, California 92903
51. Chief of Naval Operations 1
Attn: Dr. Earl Chavis (OP-162)
Washington, DC 20350
52. College of Business Management 1
Tydings Hall, Room 0137
Attn: Dr. Alan Hevner
University of Maryland
College Park, Maryland 20742
53. Department of Computer and Information Science 1
Attn: Dr. John A. Stankovic
University of Massachusetts
Amherst, Massachusetts 01003
54. Department of Computer Science 1
Attn: Dr. Alfs Bertziss
University of Pittsburgh
Pittsburgh, Pennsylvania 15260
55. Computer Science Department 1
Attn: Dr. Al Mok
University of Texas at Austin
Austin, Texas 78712
56. Honeywell Systems & Research Center 1
Attn: Mr. Steve Huseh
Minneapolis, Minnesota 55418
57. U.S. Army Headquarters CECOM 1
AMSEL-RD-SE-AST-SE
Attn: Mr. George Sumiall
Fort Monmouth, N.J. 07703-5000
58. United States Laboratory Command 1
Army Research Office
Attn: Dr. David Hislop
P.O. Box 12211
Research Triangle Park, NC 27709-2211
59. Computer Science and Artificial Intelligence 1
Department of the Air Force
Attn: Dr. Abraham Waksman
Bolling Air Force Base, D.C. 20332-6448

60. NSWC, U-33 1
Attn: Dr. Phil Hwang
Silver Spoon, Maryland 20903-5000
61. NOSC, Code 805 1
Attn: Mr. Jack Stawiski
San Diego, California 92152-5000
62. Office of Naval Research 1
Applied Mathematics and Computer Science
Attn: Mr. J. Smith, Code 1211
800 N. Quincy Street
Arlington, Virginia 22217-5000
63. Chief of Naval Operations 1
Code OP-940C
Attn: Mr. Al Lezerns
Washington D.C. 20350-2000
64. Navy Center for Applied Research in AI 1
Navy Research Lab
Attn: Ms. Laura Davis, Code 5510
Washington D.C. 20375-5000
65. Lockheed Software Technology Center 1
09610/3307
Attn: Dr. Herb Krasner/ Dr. Wright
2100 East Street
Austin, Texas 78744
66. Gary W. Manley 2
391 B Ricketts Rd
Monterey, California 93940