

DTIC FILE COPY

①

FROM: AFIT/CI

11 July 1989

SUBJECT: Review of Thesis/Dissertation for Public Release

TO: PA

1. Request you review the attached for public release prior to being sent to DTIC.

2. Reply by indorsement to CI NLT _____.

Ernest A. Haygood
ERNEST A. HAYGOOD, 1st Lt, USAF
Executive Officer
Civilian Institution Programs

1 Atch.
THESIS 89-077
SIMS

1st Ind, AFIT/PA

TO: CI

Approved/~~Disapproved~~ for public release.

Log Number: 89-10-121

Harriet D. Moultrie
HARRIET D. MOULTRIE, Capt, USAF
Director, Office of Public Affairs

08 FEB 1990

DTIC
ELECTE
FEB 22 1990
S E D

AD-A218 276

A REVIEW OF THE SUITABILITY OF AVAILABLE COMPUTER AIDED
SOFTWARE ENGINEERING (CASE) TOOLS FOR THE SMALL SOFTWARE
DEVELOPMENT ENVIRONMENT

Author: Marc L. Sims

Rank: Captain

Service: United States Air Force

Date: 1988

Pages: 106

Degree: Master of Science in Computer Engineering (M.S.Cp.E.)

Institution: University of South Florida

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



00 00 00 154

A REVIEW OF THE SUITABILITY OF AVAILABLE COMPUTER AIDED
SOFTWARE ENGINEERING (CASE) TOOLS FOR THE SMALL SOFTWARE
DEVELOPMENT ENVIRONMENT

by

Marc L. Sims

An Abstract

Of a thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Engineering
in the Department of Computer Science & Engineering in
the University of South Florida

December 1988

This research informally investigates the use of Computer Aided Software Engineering (CASE) tools by a small software development organization. These tools are heralded as being productivity enhancers for software development personnel. Most of the literature discusses the generic use of this new generation of software development tools. This research effort focuses on the appropriateness of using these tools by the small software development organizations consisting of less than seven personnel. Provided in this thesis is a brief presentation of the software engineering discipline and descriptions of the CASE family of tools along with a discussion of 33 specific tools. A small software development organization model is provided and its areas of concerns are discussed. This research also performed a survey of users of current CASE products and the results from the 76 respondents are presented.

LITERATURE CITED

- [Boeh73] Boehm, Barry W., "Software and its Impact: A Quantitative Assessment," Datamation, 19, 5, (May 1973), pp 48-59.
- [Boeh76] Boehm, Barry W., "Software Engineering," IEEE Transactions on Computers, C-25, 12, (December 1976), pp 1226-1241.
- [Boeh81a] Boehm, Barry W., Software Engineering Economics, Prentice-Hall, Inc., Englewood Cliffs, NJ, (1981).
- [Boeh81b] Boehm, Barry W., "An Experiment in Small Scale Application Software Engineering," IEEE Transactions on Software Engineering, SE7,5, (September 1981) pp 482-493.
- [Booc87] Booch, Grady, Software Engineering with Ada, 2ed, Benjamin/Cummings Publishing Company Inc., Menlo Park, CA, (1987).
- [Broo82] Brooks, Frederick P. Jr., "The Mythical Man-Month, Essays on Software Engineering," Addison-Wesley Publishing Company, Reading, MA, (1982).
- [Broo85] Brookshear, Glenn J., Computer Science: An Overview, Benjamin/Cummings Publishing Company Inc., Menlo Park, CA, (1985).
- [Brow88] Brown, Alice C., "Review of the Availability of CASE Tools for the PC and Workstations," IEEE & ACM Professional Development Seminar, Tampa, FL, (June 4, 1988).
- [Char86] Charette, Robert N., Software Engineering Environments: Concepts and Technology, McGraw Hill, Inc., New York, NY, (1986).
- [Dema79] DeMarco, Tom, Structured Analysis and Specification, Prentice-Hall, Inc., Englewood Cliffs, NJ, (1979).
- [Dijk65] Dijkstra, Edsger W., "Programming Considered as a Human Activity," in Proceedings of the 1965 International Federation of Information Processing Congress, North Holland Publishing Company, Amsterdam, Netherlands, (1965) pp 213-217.
- [Dijk72] Dijkstra, Edsger W., "The Humble Programmer," Communications of the ACM, 15, 10, (October 1972), pp 859-866.
- [DOD88] Department of Defense. Defense System Software Development, DOD-STD-2167A, Department of Defense, Washington D.C., (1988).

- [Edwa88] Edwards, William W. II, "A Methodology for CASE Tool Selecetion," in Proceedings of CASE Studies 1988. Ninth Annual Conference on Applications of Computer Aided Software Engineering Tools, (May 23-27 1988), Section C8801, pp1-35.
- [Gibs88] Gibson, Michael L., "A Guide to Selecting CASE Tools," Datamation, 34, 13, (July 1, 1988), pp 65-66.
- [Glas82] Glass, Robert L., "Recommended: A Minimum Standard Software Toolset," ACM Software Engineering Notes, 7,4, (October 1982), pp 3-13.
- [Hoar81] Hoare, C. A. R., "The Emperor's Old Clothes," Communications of the ACM, 24,2, (February, 1981), pp 75-83.
- [Jack83] Jackson, M.A., System Development, Prentice-Hall, Inc., Englewood Cliffs, NJ, (1983).
- [Knut73] Knuth, Donald E. The Art of Computer Programming. Vol 1/ Fundmental Algorithms, 2ed, Addison-Wesley Publishing Company, Reading, MA, (1973).
- [Lick85] Licker, Paul S., The Art of Managing Software Development People, John Wiley & Sons, New York, NY, (1985).
- [Merl88] Merlyn, Vaughn P., "CASE - Today and in the Future," in Proceedings of CASE Studies 1988. Ninth Annual Conference on Applications of Computer Aided Software Engineering Tools, (May 23-27, 1988), Section C8820, pp1-12.
- [Naur76] Naur, Peter, Randell, Brian, Buxton, J.N., Software Engineering Concepts and Techniques: Proceedings of the NATO Conferences, Petrocelli/Charter, New York, NY, (1976).
- [Orr77] Orr, K.T., Structured Systems Development, Yourdon Press, New York, NY, (1977).
- [Parn72] Parnas, D. L., "On Criteria to be used in Decomposing Systems into Modules," Communications of the ACM, 15, 12, (December 1972), pp 1053-1058.
- [Pres87] Pressman, Roger S., Software Engineering: A Practitioner's Approach, 2ed, McGraw-Hill Book Company, New York, NY, (1987).
- [Stay76] Stay, J.F., "HIPO and Ingrated Program Design," IBM Systems Journal, 15, 2 (1976), pp 143-154.
- [Voel88] Voelcker, John, "Automating Software: Proceed with Caution," IEEE Spectrum, 25, 7, (July 1988), pp 25-27.
- [Ward86] Ward, Frank, "Keynote Address", Proceedings: Workshop on Future Directions in Computer Architecture and Software, (Dharma P. Agrawal, ed.), 5-7 May 1986, pp 1-15.

- [Warn74] Warnier, Jean D., Logical Construction of Programs, 3ed., Van Nostrand Reinhold Company, New York, NY, (1974).
- [Wass82] Wasserman, Antony, I., "Automated Tools in the Information System Development Environment," in Automated Tools for Information Systems Design, (Hans-Jochen Schneider & Antony I Wasserman, eds.), North Holland Publishing Company, Amersterdam, The Netherlands, (1982), pp 1-9.
- [Whit88] Whitmore, Sam, "Programming Shortcuts are Not Time Savers in Long Run," PC Week, (June 28 1988), pg 32.
- [Wein71] Weinberg, Gerald M., The Psychology of Computer Programming, Van Nostrand Reinhold Company, New York, NY, (1971).
- [Wirt71] Wirth, Niklaus, "Program Development by Stepwise Refinement," Communications of the ACM, 14, 4, (April 1971), pp 221-227.
- [Your79] Yourdon, Edward N., and Constantine, Larry L., Structured Design: Fundamentals of a Discipline of Computer Program and System Design, Prentice-Hall, Inc., Englewood Cliffs, NJ (1979).

Graduate Council
University of South Florida
Tampa, Florida

CERTIFICATE OF APPROVAL

MASTER'S THESIS

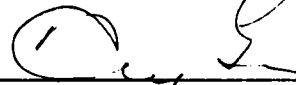
This is to certify that the Master's Thesis of

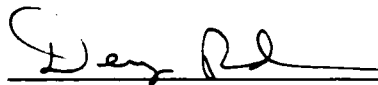
Marc L. Sims

with a major in Computer Engineering
has been approved by the Examining Committee
on 2 November, 1988 as satisfactory for the
Thesis requirement for the Master of Science in
Computer Engineering degree.

Thesis Committee:


Major Professor: W. Clark Naylor, Ph.D.


Member: Harvey Glass, Sc.D.


Member: Dewey Rundus, Ph.D.

**A REVIEW OF THE SUITABILITY OF AVAILABLE COMPUTER AIDED
SOFTWARE ENGINEERING (CASE) TOOLS FOR THE SMALL SOFTWARE
DEVELOPMENT ENVIRONMENT**

by

Marc L. Sims

**A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Engineering
in the Department of Computer Science & Engineering in
the University of South Florida**

December 1988

Major Professor: W. Clark Naylor, Ph.D.

ACKNOWLEDGEMENTS

I would first like to express my gratitude to the special ladies in my life; Dorothy, Dori and Tonya who have provided unswerving support to me in all of my endeavors. With their faith, humor and stability, my journey has been both pleasurable and rewarding.

I would also like to especially thank my major professor, Dr Clark Naylor for his counsel and encouragement. He tirelessly provided his valuable time and energy to keep me on course for these past months. My thanks to my committee members; to Dr Glass for suggesting a research topic that was both professionally relevant and personally interesting and to Dr Rundus for his stimulating discussions. I also offer my appreciation for the camaraderie and fellowship provided by my good friends and colleagues at the University of South Florida.

In the past 19 years, I have had the good fortune to work with and learn from exceptional people and I would like to take this opportunity to acknowledge their contributions. Special thanks go to John Matson, Harold Gordon, Don Amitrani, Bill Mace, Jim Senter and Bob Alford. They have been exceptional friends.

Thanks go to the vendors who provided assistance and support for this research. Special thanks go to the dozens of software professionals who spent their valuable time to help our effort. To all, my sincere appreciation.

TABLE OF CONTENTS

LIST OF TABLES	vi
LIST OF FIGURES	vii
ABSTRACT	viii
I. INTRODUCTION	1
1. Background	1
2. Statement of the Problem	2
3. Research Objectives	2
4. Methodology	3
II. THE SOFTWARE CRISIS	4
1. Historical Perspective	4
2. Current Status	7
3. Future Trends	7
III. SOFTWARE ENGINEERING	9
1. Definition	10
2. Software Development Methodologies	12
2.1. Waterfall: The Classic Model	13
2.2. Prototyping: A Rapid Alternative	16
2.3. Fourth Generation Techniques: Model of the Future?	18
3. Software Design Techniques	19
3.1. Process-Oriented Design	19
3.2. Data-Structure Design	20
3.3. Object-Oriented Design	21
4. Summary	21
IV. AN OVERVIEW OF COMPUTER AIDED SOFTWARE ENGINEERING (CASE) SOFTWARE DEVELOPMENT TOOLS	23
1. Introduction	23
2. Tools for the Software Development Environment	24
2.1. Project Management Tools	25
2.2. Requirements Definition and Analysis Tools	26
2.3. Design Tools	27
2.4. Coding Tools	27
2.5. Testing Tools	30
2.6. Maintenance Tools	30

3.	CASE Tools Hierarchy	31
3.1.	Front-End CASE Tools	32
3.2.	Back-End CASE Tools	33
3.3.	Reverse Engineering/Maintenance CASE Tools	34
4.	CASE Product Descriptions	34
4.1.	ProKit*Workbench	34
4.1.1.	Hardware Requirements	35
4.1.2.	Project Management Capabilities	35
4.1.3.	Front-end Capabilities	35
4.2.	RECODER	36
4.2.1.	Project Management Capabilities	38
4.2.2.	Reverse Engineering/Maintenance Capabilities	38
4.3.	Power Tools	40
4.3.1.	Front-end Capabilities	40
4.3.2.	Back-end Capabilities	44
4.3.3.	Reverse Engineering/Maintenance Capabilities	44
4.4.	CASE Products Capabilities	44
5.	Summary	50
V.	THE SMALL ORGANIZATION DEVELOPMENT ENVIRONMENT	51
1.	Introduction	51
2.	The Small Organization Model	51
2.1.	Organization	52
2.2.	Personnel Responsibilities	52
2.3.	Functional Responsibilities	54
3.	Difficulties in a Small Organization	55
3.1.	Limited Personnel	55
3.1.1.	Limited Overlap of Functional Responsibilities	55
3.1.2.	Limited Depth of Technical Knowledge	55
3.1.3.	Shared Software Engineering Responsibilities	56
3.1.4.	Heavy Maintenance Workload	56
3.1.5.	Backlog of Application Development	57
3.2.	Informal Development Attitudes	57
3.2.1.	Poorly Enforced Development Practices	58
3.2.2.	Uncompleted Projects	59
3.2.3.	Poor Documentation	59
4.	CASE Tool Requirements of a Small Organization	60
4.1.	Front-End CASE Tools	60
4.2.	Back-End CASE Tools	60
4.3.	Reverse Engineering/Maintenance CASE Tools	61
4.4.	Project Management CASE Tools	61
5.	Summary	61
VI.	INDUSTRY SURVEY	63
1.	Introduction	63
2.	Survey Methodology	63

3. Results	65
3.1. Organization Composition	66
3.1.1. Organization Size	66
3.1.2. Software Project Characteristics	68
3.1.3. Organization Software Development Practices	68
3.2. Use of CASE Tools in General	69
3.3. Use of Specific CASE Tools	73
3.4. Additional Comments Received	73
3.4.1. Overall System Capabilities	75
3.4.2. Project Management Capabilities	75
3.4.3. Front-end Capabilities	76
3.4.4. Back-end Capabilities	76
3.4.5. Reverse Engineering/Maintenance Capabilities	76
 VII. CONCLUSIONS	 77
1. Findings	77
2. A Comment Concerning Future CASE Tools	79
3. Future Research	79
 LITERATURE CITED	 81
 APPENDIXES:	 84
APPENDIX A. VENDOR CONTACTS	85
APPENDIX B. SAMPLE SURVEY QUESTIONNAIRE	88
APPENDIX C. SURVEY DATA	94

LIST OF TABLES

TABLE 1	Project Management Tools.	26
TABLE 2	Requirements Definition and Analysis Tools.	27
TABLE 3	Design Tools.	28
TABLE 4	Coding Tools.	29
TABLE 5	Testing Tools.	30
TABLE 6	Maintenance Tools.	31
TABLE 7	CASE Products Capabilities Matrix.	45
TABLE 8	Questionnaire Distribution.	65
TABLE 9	Combined Results of Opinions Provided for All CASE Tools.	73
TABLE 10	Response Averages for the Individual CASE Tools.	74

LIST OF FIGURES

FIGURE 1	Hardware/Software Cost Trends.	5
FIGURE 2	Components of a Software Development Methodology.	13
FIGURE 3	Classic "Waterfall" Model Life-Cycle.	14
FIGURE 4	Prototype Model Life-Cycle.	17
FIGURE 5	Umbrella of Software Development Tools.	25
FIGURE 6a	Categories of Software Development Tools.	32
FIGURE 6b	Categories of Software Development Tools (Revised).	33
FIGURE 7	Sample ProKit*Workbench Data Flow Diagram.	37
FIGURE 8	Sample RECODER Structure Chart.	39
FIGURE 9	Sample Power Tools Leveled Set of Data Flow Diagrams.	41
FIGURE 10	Sample Power Tools Data Flow Diagram with Consistency and Balance Reports.	42
FIGURE 11	Sample Power Tools State Transition Diagram.	43
FIGURE 12	Reported Sizes of the Software Development Groups.	67
FIGURE 13	Average Composition of the Software Development Group.	68
FIGURE 14	Level of Organizational Software Development Formalisms.	69
FIGURE 15	Recommendation for the Use of CASE Tools by Small Organizations.	70
FIGURE 16	Amount of Personnel Who Use the CASE Tools.	71
FIGURE 17	Modifications to the Organizations Caused by Installing the CASE Tools.	71
FIGURE 18	Phases in Which the CASE Tools are Used.	72

**A REVIEW OF THE SUITABILITY OF AVAILABLE COMPUTER AIDED
SOFTWARE ENGINEERING (CASE) TOOLS FOR THE SMALL SOFTWARE
DEVELOPMENT ENVIRONMENT**

by

Marc L. Sims

An Abstract

**Of a thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Engineering
in the Department of Computer Science & Engineering in
the University of South Florida**

December 1988

Major Professor: W. Clark Naylor, Ph.D.

This research informally investigates the use of Computer Aided Software Engineering (CASE) tools by a small software development organization. These tools are heralded as being productivity enhancers for software development personnel. Most of the literature discusses the generic use of this new generation of software development tools. This research effort focuses on the appropriateness of using these tools by the small software development organizations consisting of less than seven personnel. Provided in this thesis is a brief presentation of the software engineering discipline and descriptions of the CASE family of tools along with a discussion of 33 specific tools. A small software development organization model is provided and its areas of concerns are discussed. This research also performed a survey of users of current CASE products and the results from the 76 respondents are presented.

Abstract approved:

W. C. Naylor
Major Professor: W. Clark Naylor, Ph.D.

Associate Professor,
Computer Science & Engineering

11-2-88
Date of Approval

I. INTRODUCTION

"Where shall I begin, please your Majesty?" he asked. "Begin at the beginning," the King said gravely, "and so on till you come to the end: then stop."

Lewis Carroll: Through the Looking-Glass

1. Background.

The cost of producing computer software continues to increase. It has become by far the most expensive portion of most current automated projects. When the cost of maintaining this software is identified for the entire life of the project, we find that the software expense can become as much as 80% of the total system cost.

The *software crisis* was identified in the late sixties and is still with us. In fact recent studies reveal that the backlog of software projects often exceeds 3-4 years. We have more software developers today, but we also have a much greater demand for additional software. With the proliferation of the new generation of microcomputers, and greater dependency on automated support by every faction within the corporate world and government agencies, there is simply more software required. Instead of an easing of the much discussed *software crisis*, it is instead becoming more acute. Because of this tremendous demand for software, extensive research has taken place to improve computer programmer productivity. A plethora of methodologies, tools and management techniques is described in the current literature. Additionally, many vendors have produced tools and productivity aids that are advertised to provide tremendous improvements. These tools are being provided to assist the software engineers create code better, faster and cheaper.

Recently a new wave of computer software development tools has been created. They are called Computer Aided Software Engineering (CASE) tools. These new tools have

been heralded by many professionals and discounted by others as another over-hyped marketing ploy. The true usefulness and capabilities of these CASE tools can only be determined after they have been utilized by software engineers. It is clear that the current generation of CASE tools possesses promise and demands the attention of both the software development professionals and the academic community. Only after rigorous review and practical experience may the true capabilities of the new CASE tools be identified.

2. Statement of the Problem.

There appears to be an overwhelming consensus in the literature that installing the available Computer Aided Software Engineering (CASE) tools in large organizations is appropriate. However there is little discussion of the use of these CASE tools by small software development organizations. The small software development organization has special concerns such as limited personnel resources and management practices that are peculiar to the smaller sized organization.

The purpose of this research is to attempt to identify the special concerns and requirements of a small organization and to evaluate whether the current generation of CASE tools is effective for their use.

3. Research Objectives.

This research will first identify a small software development organization model. This model will be used to identify the special requirements of a small organization. This research will then use this model to evaluate the potential suitability of commercially available CASE products. This research does not evaluate the CASE products with the intent to recommend the purchase of any product or group of products. An excellent discussion of how to select an individual CASE product is provided by Edwards [Edwa88]. He provides a methodology to quantify the customer's requirements which are

then used in the selection process.

The aim of this research is to identify a set of requirements for the small organization model and perform a review of the existing CASE products for their suitability to satisfy those requirements as a class of tool. It will also identify potential enhancements to existing CASE tools that would be responsive to the small organization.

4. Methodology.

This research did not restrict the hardware environment required by the CASE tools so that we would be able to review the largest number of products. Historically, it is simply a matter of time before a good application or capability is transported from the original environment to a new one if there is sufficient demand. The organization model was intentionally made small to ensure that the survey results would apply to the majority of organizations. If the CASE products are appropriate for this model then they should also be appropriate for larger sized organizations.

The methodology in this research consisted of the following: a literature search of software engineering history, practices, development tools and methodologies was performed and a selection of representative commercial CASE products was reviewed. Product literature, product documentation, demonstrations were reviewed and discussions with vendor representatives were held. Finally software engineering professionals who have used representative CASE products were surveyed. This survey requested the software engineer's opinions concerning the use of their specific CASE tool and also the use of the general class of CASE tools by the small organizations. Follow-up interviews were conducted with selected respondents to obtain additional information and to ensure that the results were being correctly interpreted.

II. THE SOFTWARE CRISIS

To be fair, I must say that although the compiler didn't work, it was easy to use

Karen Underwood: CHIPS, January 1987

What is meant by the term *software crisis*? Does it still exist? Should we as software engineers be concerned? Is there anything we can do about it? This chapter provides a brief historical background of what is commonly referred to as the *software crisis*. It will then identify the current status and provide some projections for the future.

1. Historical Perspective.

The first public recognition of the existence of a *software crisis* is attributed to the International Conference on Software Engineering convened at Garmish, West Germany in 1968 [Naur76]. The term *software crisis* has since been used by software development professionals to identify any perceived difficulty relating to writing computer software. Authors and vendors promoting their ideas and products are quick to announce that their solution corrects the *software crisis*.

The original problems that were identified at the conference in 1968 consisted of three areas of concern. These were the design, production and service of software. The design issues were related to the design criteria and the techniques of good design implementation. The production area consisted of the technical aspects of actually creating the code such as problems of scale, planning, personnel factors and tools. The final area dealing with service was concerned with issues relating to timing and frequency of software releases, distribution, system testing/evaluation, maintenance and documentation.

Over time, different authors and software professionals have added to the understanding of what are the fundamental causes of the *software crisis*. A result of the *software crisis* has been the problem of having a backlog of applications. Computer programming has long been considered an imprecise "art". Because of this, most programmers in the past did not have rigorous development formalisms to use but instead would write code "on the fly". A minority of these programmers were truly artists and were exceptional, however, most of the programmers suffered from a lack of precisely identified development procedures which contributed to the crisis.

In 1973, Boehm [Boeh73] documented the state of the *software crisis* by identifying some of the factors found to cause problems in developing computer systems. He identified that software repeatedly is the subsystem of a project that most often delayed its completion and was also the most difficult to gauge and properly control. Figure 1 is the hardware/software cost trends graph he presented. This graph illustrates that the cost of software as a percentage of the cost of the entire effort was less than 20% in 1955 but had grown to approximately 75% by 1972.

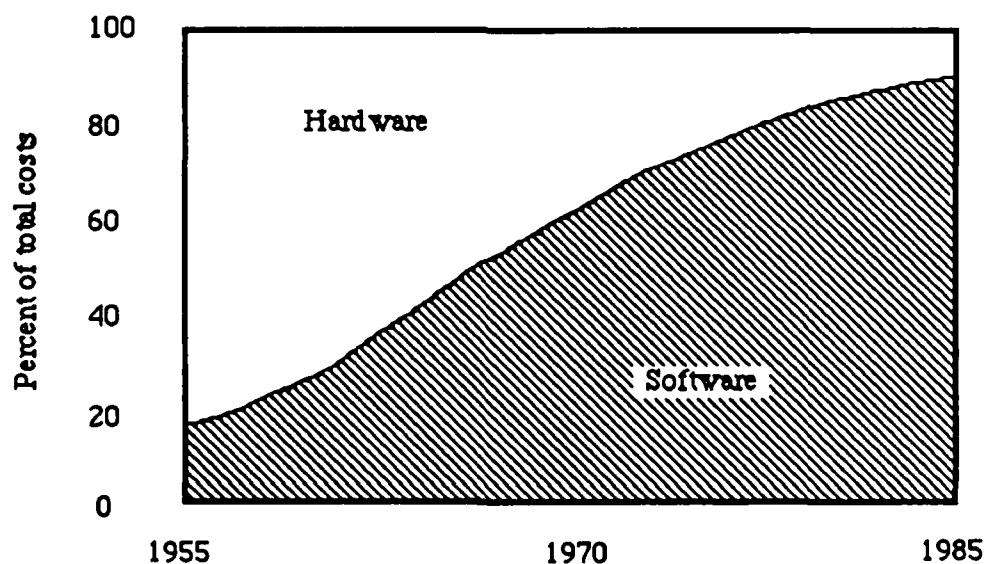


Figure 1 Hardware/software cost trends.

The trend of rising software costs is what focused the discussion of the *software crisis*. Project managers were now facing the reality of software being the major project expense. While the hardware was the overwhelming expense, managers tended to overlook software development inaccuracies and their associated problems. However in the late sixties and early seventies, the proportional cost of software had risen to such a level that management was forced to focus its attention on the software development problems and related productivity issues. This led to a determined effort to transform the "art" of programming into the science of computer programming by formulating a new discipline called software engineering. This was the first attempt to identify and introduce the solid engineering principles of rigorous scientific methodologies in order to produce more efficient, reliable software.

Dijkstra [Dijk72] provided his insights concerning this new attention to the software development practices during his presentation lecture while accepting his 1972 ACM Turing Award.

as long as machines were the largest item on the budget, the programming profession could get away with its clumsy techniques; but that umbrella will fold very rapidly

Dijkstra went on to project that as the programming profession is able to solve the programming problems of that day, that the bigger, better, faster machines of the future will allow us to solve the larger applications that could not even be attempted in 1972. In this manner the *software crisis* will not only continue but in fact expand.

It is significant that the actual costs of software were not sharply increasing. Instead the proportional cost of software in relation to the cost of hardware was rising. This was due in large part to the rapidly decreasing prices of computer hardware. With hardware relatively inexpensive, the steadily rising prices of software received the focus of the industry's attention. As a result, structured programming techniques, rigorous program design efforts, documentation and improved software development practices became

popular with industry. These improved development methods gained favor and prominence within industry since they had the beneficial side effect of increasing programmer productivity thereby reducing software expenses.

2. Current Status.

Presently, studies reveal that Boehm's projections of 1973 were indeed prophetic since software costs currently represent an average cost in excess of 80% of total project costs. Additionally, the backlog of automated projects can approach 3-4 years in some organizations [Char76].

Today's focus on the *software crisis* consists of three major areas. Pressman [Pres87] explains that the following are the current problems posed by the *software crisis*.

The software crisis is characterized by many problems but managers responsible for software development concentrate on the "bottom line" issues: (1) schedule and cost estimates are often grossly inaccurate; (2) the "productivity" of software people hasn't kept pace with the demand for their services; and (3) the quality of software is sometimes less than adequate.

Pressman has accurately stated that the three principle problems all relate to the "bottom line" of software cost which is the most important to industry. These consist of the direct production costs as they relate to programmer productivity and also the deferred costs of maintenance. The goal of eliminating the *software crisis* is to generate more software with less expense, thereby increasing industry's profit.

3. Future Trends.

It is extremely difficult to project accurately what will happen to the backlog of software projects or identify improvements that will be made in developing software. It is clear however that continued research is being performed by industry and the academic community. The formation of the Software Engineering Institute (SEI) which was

sponsored by the Department of Defense (DOD) at Carnegie Mellon University in 1985 is a prime example of trying to bring the *software crisis* under control.

Projections identify that the costs of software in relation to hardware will continue to grow. In May 1985, Lieutenant Colonel Ward, Office of the Secretary of Defense, in his keynote address [Ward86], stated that the United States was lacking 50-100,000 software professionals and that by 1990 the shortage will grow to over 1 million. He also projected that the costs of the military tactical software, excluding business software, will increase from \$10 billion in 1985 to \$30 billion in 1990.

It is easy to locate precise descriptions of the problems associated with the *software crisis*. Numerous authors inform us that software costs too much, is not reliable, and that we have a large backlog of software programs still waiting to be written. What is much more difficult is to implement solutions to these problems. Therefore the charter for all software engineers is to aggressively seek improvements in the software development methodologies to help reduce the crisis of the future. This thesis will attempt to advance this cause by investigating a category of software development tools that may help relieve these problems.

III. SOFTWARE ENGINEERING

War is not, as some seem to suppose, a mere game of chance. Its principles constitute one of the most intricate of modern sciences; and the general who understands the art of rightly applying its rules, and possesses the means of carrying out its precepts, may be morally certain of success

Major General H. W. Halleck, USA, 1846

For twenty years, the software industry and academic community have sponsored conferences and seminars to discuss and promote the software engineering discipline. Indeed, two of the foremost computer science professional societies publish journals dedicated to software engineering. The Institute of Electrical and Electronic Engineers, Inc. (IEEE) has published the Transactions on Software Engineering since 1975. The Special Interest Group on Software (SIGSOFT) of the Association of Computing Machinery (ACM) has published the Software Engineering Notes (SEN) since 1976. The use of the software engineering discipline is arguably the most common approach identified to overcome the *software crisis*. The software engineering discipline is a result of the evolution from when programming a computer was considered an art form rather than the application of scientific practices.

Early authors were torn between the process of writing programs being an artistic creation or a science. Knuth [Knut73] in fact entitled his reference texts The Art of Computer Programming. He described programming a computer as "a programmer's craft" and that it "can be an asthetic experience much like composing poetry or music". Knuth in his 1974 ACM Turing Award lecture [Knut74] he explained why he used the term "art" rather than "science" for his titles. He stated that

Science is the knowledge which we understand so well that we can teach it to a computer; and if we don't fully understand something, it is an art to deal with it.

He went on say that the "process of going from an art, to a science means that we learn how to automate something".

The material in Knuth's works while labeled "art" can not be considered light reading. However, the careful implementation of his ideas result in the creation of programs that could be judged artistic. This conflict was prevalent in the seventies. To be a good programmer required capabilities of a true artisan. But due to the *software crisis*, the industry had to generate a great deal of code and most programmers did not have Knuth's insights to the beauty and "art" of computer programming. They were technicians trying to write good code as quickly as possible. From this situation, the new discipline of software engineering was born. This chapter will provide a very brief overview of software engineering. It will then identify and briefly explain software development methodologies and also three software design techniques.

1. Definition.

Many definitions of software engineering exist. Bauer [Pres87] provided the following definition at the first major conference that dealt with software engineering.

The establishment and use of sound engineering principles in order to obtain economically, software that is reliable and works efficiently on real machines.

In 1976, Boehm [Boeh76] refined the above definition of software engineering to specifically include software design and documentation.

The practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate and maintain them.

Boehm felt that previous definitions did not sufficiently cover the entire software life-cycle. Boehm specifically included "design" in his definition to cover the critical aspect of software development that was often slighted and he also included "software maintenance" to cover the software support required to maintain the software after its initial delivery.

Currently, even textbooks used in introduction to computer science courses discuss the software engineering discipline. In the Brookshear text [Broo85], the author states that "the software development process is called software engineering". He goes on to state that it entails such subjects as "personnel management, equipment management, system structure, and design methodologies". It is difficult to find a more encompassing definition than this one. This reflects the tendency to include all things that could possibly have anything to do with software development as software engineering. We would like to offer a little more restrictive definition.

Software Engineering is the scientific application of formally defined methods and procedures to develop and support reliable, economical and easily maintainable computer software and associated documentation such that these methods and procedures are precise, reproducible and understandable.

Three main points of the above definition should be explained. The first point concerns the "scientific application of formally defined methods and procedures". These methods and procedures must be formally defined so that they are transferable from one environment to another. They must be precisely applied in a scientific manner in that they can not be haphazardly or partially applied. The second point is that these methods are to be used "to develop and support reliable, economical and easily maintainable" software and documentation. Foremost, the software should be reliable. It should do what is intended.

It should also be economical in design, cost and resource efficiencies. The software must also be integrated into the maintenance phase. Since extensive efforts are performed to maintain delivered software, care must be taken to "build in" good maintenance capabilities. Finally, these methods and procedures need to be "precise, reproducible and understandable". They should not be ambiguous to the programmers and they also need to be reproducible so that they can be scientifically taught and applied. They must certainly be understandable so that they can be effectively used by all software development technicians and not only by an elite group of enlightened personnel. They must be available to all personnel and in order to accomplish this, the methodologies must be easily understandable.

2. Software Development Methodologies.

As we have seen in the previous section, software engineering consists of the scientific application of formal software development methodologies. It is now time to more fully identify these methodologies. Wasserman [Wass82] presented his ideas about software development methodologies during the 1982 International Federation of Information Processing (IFIP) Working Group 8.1 Working Conference on Automated Tools for Information Systems Design and Development. Figure 2 illustrates what these methodologies contain.

Wasserman explains that the software development methodology is the heart of the software development environment. The software development methodology is the set of formal procedures that are used to develop software. This methodology according to Wasserman actually consists of three major areas; management procedures, technical methods and automated tools. The management procedures can be considered the "what" of software development which specify which actions are to be taken in the life-cycle. They also may identify when and how reviews are to be conducted along with coding,

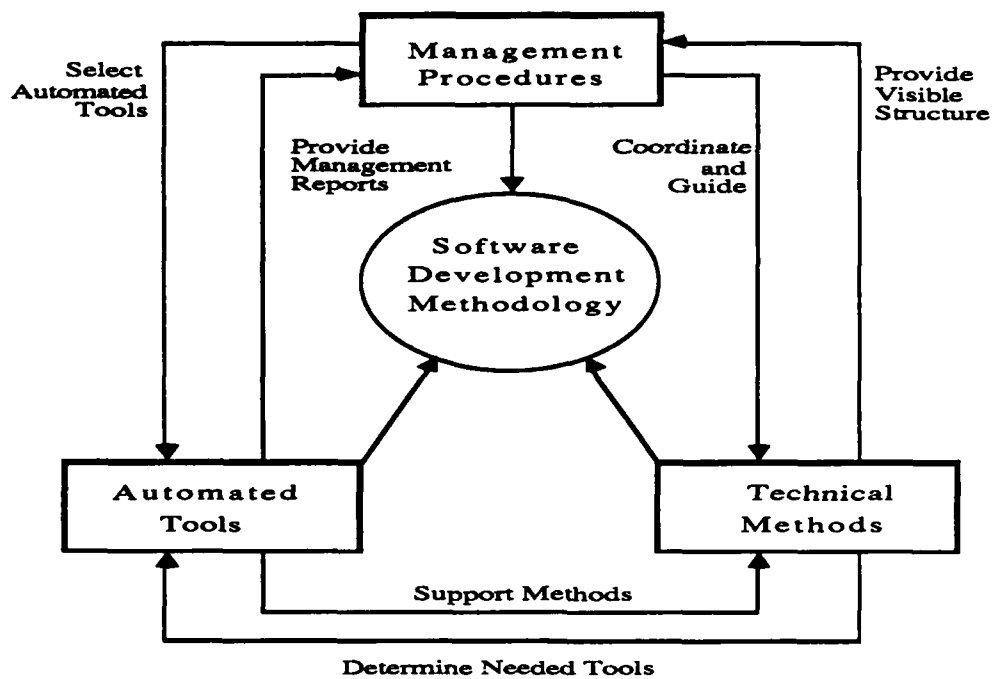


Figure 2 Components of a software development methodology.

systematic testing, and software configuration management". These can be considered the "how" of developing software. Finally, the automated tools assist both software engineering technicians in generating the actual design, code, testing and maintenance functions in addition to providing needed management support such as configuration management, reports and other controlling functions. As illustrated in Figure 2, the software development methodology consists of all three subfunctions all tightly coupled. We will next discuss three of the significant software development methodologies.

2.1. Waterfall: The Classic Model.

The classic "waterfall" model of the software development life-cycle is the oldest and is widely discussed by a number of authors. Variations of it are used throughout the world including the Department of Defense [DOD88]. There is not a consensus opinion of the

particular phases that exist within the model. However, Pressman [Pres87] provides a representative "waterfall" model life-cycle paradigm. Figure 3 identifies the six phases of his software development model. The name of this model is apparent due to the waterfall appearance of the process flow from top left to bottom right.

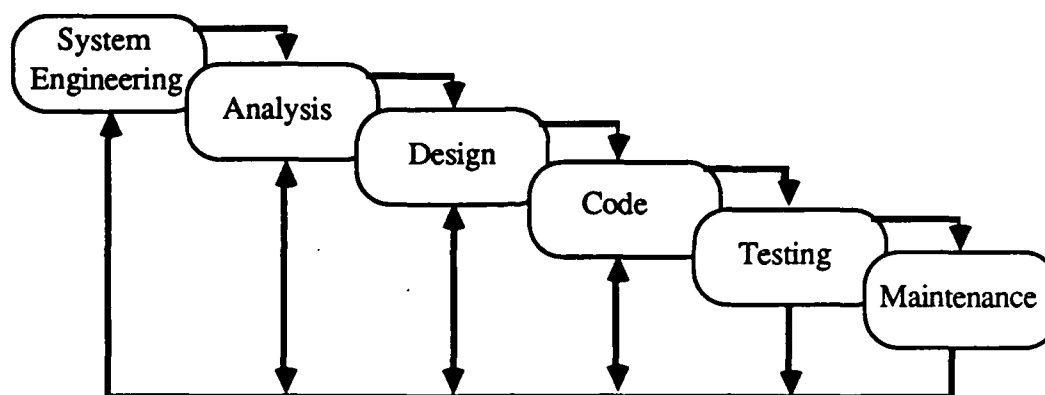


Figure 3 Classic "waterfall" model life-cycle.

Each phase of the life-cycle contains within it a validation or verification step that ensures that the project is correct. Boehm [Boeh81a] informally explained that the validation process answered the question "are we building the right product?" while the verification process answers the question "are we building the product right?" If the output from any phase is incorrect, then the model returns to the appropriate preceding phase for corrective action. These feedback loops are denoted by the flow arrows in the diagram looping back up the process chain.

The system engineering phase is concerned with discovering how the software product will fit into the overall system environment to include impact by other software, hardware and personnel. An example would be if the customer wanted to add a billing program to an existing business system. The software engineer would then perform an analysis of how this new program would interact with the existing system. Would the hardware have sufficient capacity? Will there be interfaces with other computer systems or personnel? This phase also includes the requirements analysis where approval is provided to begin the

project once it has been verified as being necessary. Approval is then given, and the process proceeds to the next phase.

The analysis phase concentrates on the software requirements. This is where the customer and software engineer interact and both groups understand what is being requested and what the delivered system's functional capabilities will include. The validation of this phase is critical and should not be overlooked or neglected by the software engineer. It is easier to design a system when the developer knows what the user desires. Often, however, the customer does not know what he wants. At this phase, the customers are not briefed on how the software will operate but instead will be briefed about "what" the software will do.

The design phase may actually consist of several levels of design. It will identify the high level capabilities and operation of the software. It may also contain the lower level details of the software execution. Usually this phase should be accomplished with the senior software engineers on the design team and a few of the more knowledgeable customers. The "how" of the software operation design is identified during this phase and is of no legitimate concern to the customers. Customer involvement should be limited to answering questions concerning the interpretation of their requirements.

The coding phase is the implementation of the design into the computer environment. This phase is usually performed by the more junior personnel since the more complex design work has already been accomplished. Verification of this phase typically consists of desk checking by senior software engineers and structured walk-throughs of the software design by management personnel.

Testing is performed next and will be performed at different levels for distinct purposes. One type of testing will ensure that the internals of the code works correctly. This is referred to as unit or component testing. Another type of testing is the system or integration testing where the entire system operation is verified. Additionally, there is usually some acceptance testing performed by the customer prior to delivery. The

verification process for this phase usually consists of a test report.

The final phase is the maintenance phase which begins once the product is delivered and lasts for the remaining life of the system. This phase will perform the corrective repairs, customer enhancements and modifications to the software as required for hardware or software environmental changes. It is significant that this phase will also include all of the previous phase activities to some level and is usually the phase that the most manpower is expended for most projects.

As Figure 3 illustrates, feedback is provided from each successive phase back to the earlier ones. This is required to correct identified discrepancies. Each phase contains a verification or validation step that ensures that indeed what is produced matches what is desired from the previous phases.

The approval to proceed from one phase to the next varies with the size of the project, organization and type of activity. If the project is a small application, the team leader may provide his or her verbal approval to the software engineer to proceed. If it is a large complex military application being developed by civilian contractors, a document of several hundred pages and a formal approving review may be required. The "waterfall" method has the flexibility to be tailored to each application and is useful for either a small or large software project.

The "waterfall" approach is very rigorous and most effective when the requirements are stable and do not change during development. However, it begins to break down and becomes difficult to manage when modifications are desired during the project. Because the customer does not see the system until the end of the development life-cycle, alternative models have been developed.

2.2. Prototyping: A Rapid Alternative.

Because of the inherent problems experienced with the "waterfall" model, the prototyping model was developed. Pressman [Pres87] has provided a diagram (Figure 4)

that defines a five phase prototyping life-cycle. This life-cycle is similar to the "waterfall" model in that requirements are still gathered and a design is performed. However, these phases are abbreviated versions.

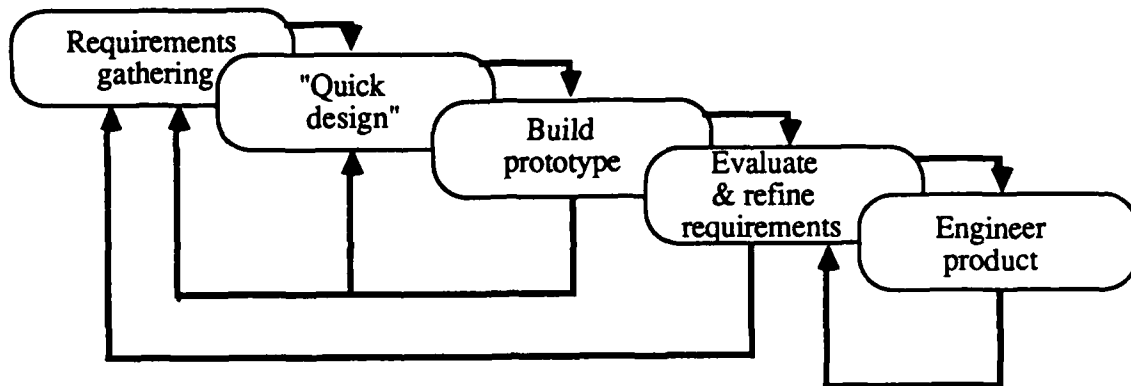


Figure 4 Prototype model life-cycle.

The first four phases will be repeated each time producing a more mature version of the system with more detailed refinements until the final system is created. At that time, the software engineer will produce the finished product.

The "first system" or initial prototype will be a simplified version of what the customer thinks he wants and that the developers are able to provide. In this manner, a version of the system is rapidly created that can be evaluated by both the customers and software engineers for deficiencies which will be corrected in the subsequent versions. The benefits of this model are that developers can create "something" very quickly so that the customers can see what the system will look like. This allows for early discovery of problems with the design or a chance to identify missing details. The software engineer can then correct the problems well before the normal delivery date. The prototype does not necessarily need to be coded but instead could be a "paper" prototype which merely illustrates the screen displays or could be a demonstration of the screens with out any "real" code to support them. It is highly recommended that a draft copy of the users manual be provided to the

customer with the first prototype demonstration.

This development model is well suited for applications where all of the information is not available at the start of the project or where the customers are not sure about exactly what capabilities they would like the system to possess.

A shortfall of this model is that the customer who sees a prototype that is apparently working may exert undue pressure for immediate delivery. This may cause shortcuts to be taken by the developers and thereby cause future problems with the delivered system since software engineering procedures may not have been completed. Another potential problem area is that the developer may take some inappropriate shortcuts to field a quick prototype thinking that they will be removed or overcome later. Later, the developer has grown so familiar and comfortable with these compromises that they remain in the system upon delivery. This model appears to be best suited for applications that are not highly complex in nature so that they can be quickly fielded once approval is provided by the customers.

2.3. Fourth Generation Techniques: Model of the Future?

Today's fourth generation techniques (4GT) are identified as a potential model to quickly develop many structured applications. The 4GT paradigm focuses on the capability to generate applications from a near natural language context using a fourth generation computer language. This model consists of first identifying the requirements of the system and then generating the software directly using the fourth generation application generator. An example would be a business application such as creating a new weekly sales report using the dBase III Database Management System (DBMS) language.

The customer would first identify what is required and then the developer would construct "super" high level instructions that would in turn generate the required source code. This function acts as an ultra high level set of macro instructions. The difficulty is that all of the management and "transitional activities" would still have to be accomplished. It appears that the 4GT paradigm at the present time is highly restricted to simple business

applications and is not responsive to large or complex efforts.

3. Software Design Techniques.

All software development life-cycle paradigms include a software design phase. No matter the paradigm, good design of the software is critical to the successful completion of any project. There has been considerable effort in attempting to create the "perfect" design technique. What can be said is that there are many to choose from and the proper one to use depends upon the application being developed. Many software development organizations maintain proficiency with several different techniques so that they can utilize the most appropriate one for each project. The different software design techniques can be classified in three general categories. They consist of process-oriented design, data-structure design and object-oriented development techniques. It is our intent to briefly explain the characteristics of each design type and list some of the known techniques of that category. References are provided to the interested reader for additional information.

3.1. Process-Oriented Design.

This design technique, as the name implies, is oriented toward developing the software design as it relates to the processes or functions of the software. This design technique is commonly known as top-down structured design. These techniques were first identified in the late sixties and were used for structured programming design efforts. Dijkstra is credited with being one of the leaders in the field of structured programming. His paper [Dijk65] is considered by many to be the keynote to the succeeding revolution of using structured programming. Dijkstra clearly explained how humans are limited in the scope and complexity of their thoughts and can not handle highly complex matters of design. He proposed that programmers use the ancient method of "Divide et impera (Divide and rule)". He advocated that a programmer should perform the following steps when developing software.

- he makes complete specifications of the individual parts
- he satisfies himself that the total problem is solved provided he had at his disposal program parts meeting the various specifications
- he constructs the individual parts, satisfying the specifications, but independent of one another and the further context in which they will be used

The above ideas clearly identify the stepwise methodology of functional decomposition that all process-oriented design techniques use. Since Dijkstra's presentation, the modern era of structured programming techniques and design have matured. Wirth [Wirt71], Parnas [Parn72], Hoare [Hoar81] and many others have expanded on the subject resulting in a keen awareness within the software industry.

The common approach with this technique is to functionally decompose the system from the top-level in a divide and conquer approach until the complete system is clearly defined in easily understandable modules. The principle differences between the individual process-oriented design techniques appear to be how the system is decomposed into modules and the different methods of graphic representations.

The following are samples of process-oriented design approaches. Hierarchy, plus Input, Process, Output (HIPO) which was developed by IBM [Stay76]. Structured Design (SD) was introduced in the mid-seventies and is credited principally to Larry Constantine [Your79]. Edward Yourdon is also credited as being a major contributor to the top-down design technique and is one of the pioneers of teaching the technique to the software industry. He continues to be a popular and well read author. Structured Analysis (SA) evolved from the Structured Design techniques in the late seventies [Dema79] and remains very similar to Structured Design and uses many of the same process flow representations.

3.2. Data-Structure Design.

The second major category of design techniques focuses on the data representations. This technique first identifies the different data that the system will contain. In this respect,

the system is data driven. The modules are identified as they transform the data. This approach also makes extensive use of the top-down approach but instead concentrates on how the data flows. Specific methods consist of Warnier Diagrams that were developed by Jean Warnier in the early seventies [Warn74] and later extended by Ken Orr [Orr77]. This technique is now known as the Warnier-Orr methodology and is also called Data Structured Systems Development (DSSD) methodology. Another data oriented technique was developed by Michael Jackson in the early seventies [Jack83] and is referred to as the Jackson design technique.

3.3. Object-Oriented Design.

This final approach was created as a result of shortfalls with both of the preceding techniques. This approach focuses on objects as being the key element in software computations. Booch [Booc87] described this approach and presented his notation for identifying objects and their relationships which is commonly used. The object-oriented technique utilizes the information hiding process where each model contains its own copy of its local data. This technique is different from the proceeding ones in that these objects have common characteristics with all other objects of the same class. It is this relationship which provides the strength of this technique. The objects in a common class inherit attributes and also processing algorithms.

A shortcoming of this technique is that it is only a partial life-cycle which is applicable for only the design and implementation phases. This necessitates that another life-cycle model be used for the other phases.

4. Summary.

Presently, we have college courses and training seminars which teach the software engineering discipline. Entire texts have been dedicated to teaching the formalisms of software engineering. It is uncommon to find a programming textbook that does not at

least briefly discuss the topic. Concerned software professionals have worked diligently to propagate good engineering practices as they concern software development and maintenance in order to assist programmers to efficiently and economically create software products. It is indeed promising that today's students are being exposed early in their studies to good software engineering methods and practices. They are the future programmers and managers that will be called upon to eliminate the backlog of programs.

Today, we tend to take for granted the traditional "waterfall" development life-cycle model and structured programming techniques. However, it was not long ago that we did not even have this roadmap to assist us in writing programs. Now we have a virtual cornucopia of methods and techniques available for our use. The current trend of using Computer Aided Software Engineering (CASE) tools is yet another effort which provides automated support to the modern programmers which is heralded as improving programmer productivity. We are indeed trying to automate the "art" of computer programming which was Knuth's suggestion of how an art evolved into a science.

IV. AN OVERVIEW OF COMPUTER AIDED SOFTWARE ENGINEERING (CASE) SOFTWARE DEVELOPMENT TOOLS

"When I use a word," Humpty Dumpty said, in a rather scornful tone. "It means just what I choose it to mean --- neither more nor less."

Lewis Carroll: Through the Looking-Glass

1. Introduction.

It is no small task to define what are computer aided software engineering (CASE) tools. Albeit it is not from a lack of discussion in the trade journals or literature. It is certainly a very popular topic of the software industry. The difficulty is one of agreement. There has not been a consensus of terms and this has caused some confusion. This chapter will endeavor to explain the current state of the CASE tool environment. Appendix A contains a list of the 33 vendors of CASE products reviewed for this research.

Training seminars, trade shows and conferences dedicated to the topic of CASE are held in every major city in the country. Newsletters are published by vendors and *experts* in the field that are dedicated to the topic of CASE. What is CASE? CASE can generally be thought of as being those software products that assist the programmers in developing software and its related support material such as documentation.

The CASE label originally was applied to microcomputer or workstation software but has since grown to include mainframe support software and project support tools. It would appear that since "CASE" is one of the current *buzzwords* of the industry, that there is no lack of vendors quick to claim that their software is actually a "CASE" tool. Most times the vendor is correct which only adds to the industry's confusion.

It is appropriate to first describe the tools that are useful within the software development life-cycle before specifically describing the CASE family of tools. This

chapter will provide a very brief overview of the current CASE tools available. It is not our intent to provide a comprehensive listing of each CASE product available. It would be impossible to attempt to identify them all --- much less provide a discussion. However, this chapter will provide a description of potential tools that are helpful for the software development phases described in the previous chapter.

The phases that we will use for our discussion will be a modified version of Pressman's "waterfall" model. The tools we identify would also be helpful in supporting any of the modern software development paradigms. This chapter will then identify the different types of tools that are labeled as CASE and how they relate to these phases. A sample of the selected CASE tools will then be discussed in more detail in order to provide the reader an understanding of the capabilities of those particular types of CASE products.

2. Tools for the Software Development Environment.

Before we describe the specific CASE tools, it is appropriate to briefly identify the tools and programming aids useful for the complete life-cycle. After constructing this foundation, we will then be able to present details of how the CASE tools relate to the software development life-cycle. This section will generically identify tools that are useful for each phase of a modified version of the classic "waterfall" software development model. The "waterfall" model is appropriate for discussion since all of the other software development paradigms will use a subset of this model to some degree. The "waterfall" model is also widely used and understood by most software engineers and therefore provides a suitable platform for discussion.

Figure 5 illustrates the six categories of software development tools. This includes a category of software development tools for project management activities that are used for the entire life of the project. This project management category is identified in Figure 5 by the functional umbrella which provides protection for the development effort and is used throughout the entire life-cycle.

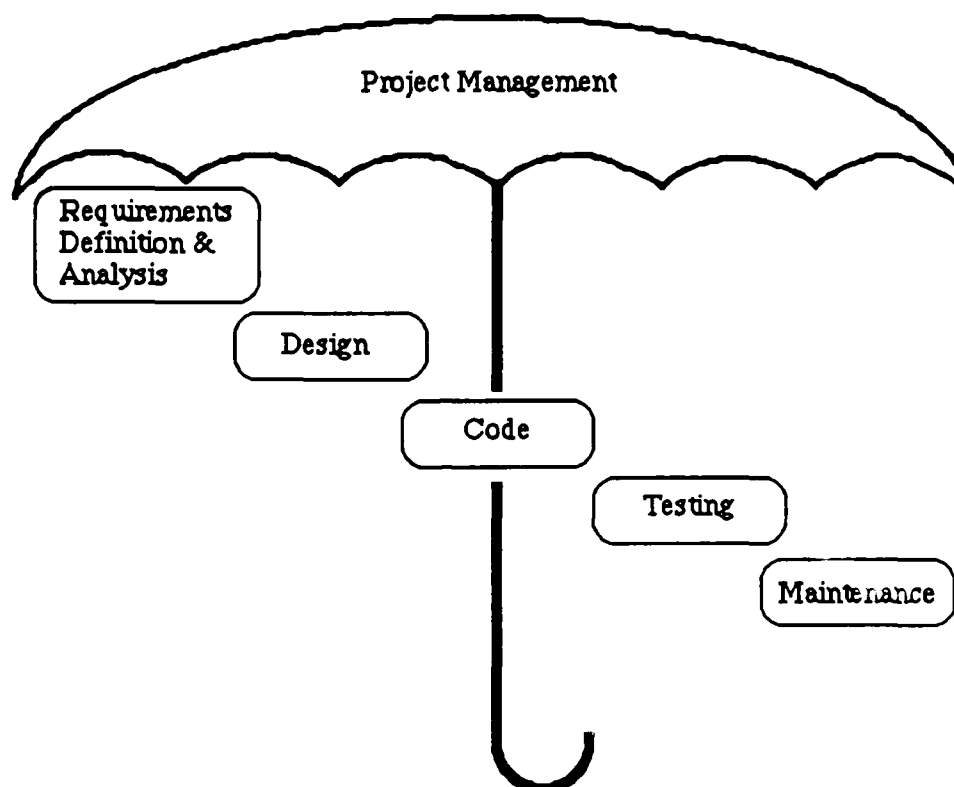


Figure 5 Umbrella of software development tools.

A general description of each category of software tool is provided in the following sections. These sections also contains tables which list some of the representative tools of each category. A mnemonic code is included for each of the tools in these tables which will be referenced in later chapters. Gibson [Gib88], Glass [Glas82] and Voelcker [Voel88] provided the majority of the information for these tables. It is difficult to identify some of the tools as belonging to one phase since many of the tools are clearly used in several phases.

2.1. Project Management Tools.

The tools used in this phase are identified in Table 1 and are those normally associated with management responsibilities. However, this category also contains tools that are

required by the software engineer responsible for the entire life-cycle. These tools assist the project leaders and software engineers to document, audit, monitor, and control the

Table 1 Project management tools.

<u>Tool</u>	<u>Function</u>
Text Editor (TE)	Word Processor, includes spell checker and thesaurus.
Project Tracking/ Control (PT)	Breaks down work into tasks, tracks schedules.
Configuration Management:	
Audit Trail (AT)	Identifies when and how modifications were made.
Project Permissions (PP)	Controls access to software modules.
Baselining (B)	Provides version baselining and change control.
Version Control (VC)	Controls the version of software being worked on.
Documentation Template (DT)	Documentation shells or outlines.
Typesetter (TS)	Enables machine readable text to be easily moved to typesetting device.
File Compare (FC)	Compares two files and identifies all differences.
Desk-Top Publishing (DTP)	Provides near or full typesetting capabilities at the software developer's workstation.
Standards Auditor (SA)	Checks software for conformance to standards.
Project Library (PL)	Contains historical data and explanations of why decisions were made.

software project. Additionally, configuration management, and project scheduling tools belong in this category. Also included are those tools that will help monitor the development effort and store project history data.

2.2. Requirements Definition and Analysis Tools.

We have combined the systems engineering and requirements analysis phases of the Pressman waterfall model. This phase will require those tools (Table 2) that assist the project leaders and customers to identify and define the project to be developed. These

tools assist management in determining what interfaces may be impacted by the new project. Requirements tracking is performed to ensure that all of the capabilities required are included in the subsequent phases.

Table 2 Requirements definition and analysis tools.

<u>Tool</u>	<u>Function</u>
Requirements Tracer (RT)	Links software requirements to the documentation and software elements created.
Requirements Language (RL)	Specialized descriptive languages used to describe systems requirements.
Requirements Illustrators (RI)	Graphically illustrates the requirements of the software and systems as a whole.
Cost Estimator (CE)	Estimates the development effort and resources.
Requirements Analyzer (RA)	Lists the functions that the system must perform and models the initial data structures.
Functional Specification Generator (FSG)	Generates functional specifications as a by product.

2.3. Design Tools.

These tools (Table 3) assist the developers perform their design of the software. These tools tend to be highly flexible and allow the software engineers to quickly change relationships, and data definitions along with the input and output designs. These tools are usually heavily dependent on graphic capabilities.

2.4. Coding Tools.

These tools (Table 4) are those that are commonly used by programmers for writing the source code. They therefore tend to be language dependent. These tools also help the software engineer analyze program complexity, format the code for readability and also automate some of the documentation generation.

Table 3 Design tools.

<u>Tool</u>	<u>Function</u>
Diagrammer (D)	Draws pictures, flowcharts of the system organization.
Data Flow Diagrams (DFD)	These are specific diagramming methods.
Structure Charts (SC)	
State Diagrams (SD)	
Entity/Relationship (E/R)	
Control Flow Diagrams (CFD)	
Process Diagrams (PD)	
Data Structured Systems Diagrams (DSSD)	
User Defined Icons (UDI)	
Customer Defined (CD)	
Screen Painter (SP)	Macro generation device to quickly create a screen display. Code will be automatically generated.
Report Generator (RG)	Like the screen painter only these generate code for reports.
Data Dictionary (DD)	Lists data elements, cross references and ensures consistency among software modules.
Consistency Checking (C)	
Balancing (B)	
Control Flow Analyzer (CFA)	Maps interactions of programs and modules; ensures all elements are referenced.
Subsystem and Interface Modeler (SM)	Tracks individual data elements; ensures that modules pass data correctly and consistently.
Data Flow Modeler (DFM)	Models the path of data through the system, shows connections between modules; ensures that no data goes unreferenced.
Tasking Analyzer (TA)	Ensures that parallel tasks in real-time systems are synchronized.

Table 4 Coding tools.

<u>Tool</u>	<u>Function</u>
Code Generator (CG)	Creates programs in a standard language automatically from an input design which is machine readable.
Subroutine Library (SL)	Contains code for common functions to be used by all programmers.
Context Sensitive Editor (CSE)	"Understands" format and syntax of source code being edited, including identifying code syntax errors.
On-line Code Debugger (OD)	Shows data during program execution, highlights errors in expected results.
Context Driven Debugger (CD)	While tracing the source code, this tool will display code that is called via calls or other context switching means.
Code Feedback (CF)	Automatically changes specifications and documentation to reflect code changes during the coding or later phases.
Compiler/Assembler (CA)	Compiles or assembles the high level source code into machine-orientated object modules.
Loader/Linker (LL)	Loads and links the software object-level modules together forming an executable program.
Conditional Compilation (CC)	Compiles only the portions of the software that are identified. This tool provides a mechanism for ignoring marked portions of code.
Preprocessor/Macro Generator (PMG)	Provides a high level or macro level instruction set for a programming language.
Instruction Level Simulator (ILS)	Allows a development computer to execute object codes of programs written for others.
JCL Generator (JG)	Generates Job Control Language instructions required to execute the software.
Static Code Tester (SCT)	Simulate the execution of the code given the initial variable and system values.
PDL Generator (PDL)	Generates pseudo code.

2.5. Testing Tools.

These tools (Table 5) are the test generators and monitors used by both the software engineers and quality assurance personnel. Also very helpful are report writing aids since this phase is heavily reliant upon producing test reports.

Table 5 Testing tools.

<u>Tool</u>	<u>Function</u>
Scripting Tool (ST)	Records transactions or conversations between users and the system, validates test data.
Test Data Generator (TDG)	Develops complete sets of data to test all modules and their interactions.
Test Data Manager (TDM)	Tracks relationships between versions of test data and test conditions.
Interface Simulator (IS)	Provides simulation of software interfaces, such as external devices or program stubs.
Formal Verification Tool (FVT)	Assists in developing formal proofs of program correctness.
Real-time Environment Simulator (RES)	Creates simulated environment to allow test execution real-time embedded software.

2.6. Maintenance Tools.

This phase will use all of the previously identified tools. Depending upon the size and complexity of the project, extensive use of the additional tools unique to this phase may be required. These unique tools (Table 6) assist the software engineer to modify the existing code and perform maintenance. These tools are commonly called reverse engineering tools. Reverse engineering is when one evaluates a previously created product and works backwards or in a "reverse" manner to discover the meaning and purpose of previously developed software.

Table 6 Maintenance tools.

<u>Tool</u>	<u>Function</u>
Pretty Printer (PP)	Formats source code into an identified structure.
Code Restructurer (CR)	"Tidies up" existing source code enabling easier understandability and maintainability.
Global Cross Referencer (GCR)	Identifies all modules and documentation elements that interface with each module and variable in the system.
Design Generator (DG)	A "reverse engineering" tool that will back out the design of existing code that can then be easily understood and modified.
Disassembler (DA)	Produces assembler source code from object code.
Timing/Performance Analyzer (TPA)	Runs at program execution time to identify areas of poor or slow execution.
Documentation Generator (DG)	Generates documentation directly from existing source code.
Structure Chart Generator (SCG)	Generates structure charts from the source code.
Metric Analyzer (M)	Measures the complexity of the source code.
Translator (T)	Translates the source code from one language to another.

3. CASE Tools Hierarchy.

CASE tools are usually referred to as being in one of the two principle types. The first type is known as front-end or "upper" CASE while the second type is known as back-end or "lower" CASE. We would like to suggest that there is a third type that we call the reverse engineering/maintenance (RE/M) CASE tool. Most of the CASE tools fall into one of these three category types. In his keynote address at the Ninth Annual Conference on Applications of CASE Tools hosted by META Systems Inc. in Ann Arbor, Michigan, Merlyn identified his definition of the three major categories of today's CASE tools [Merl88] which is provided in Figure 6a. We would suggest that that this figure should be

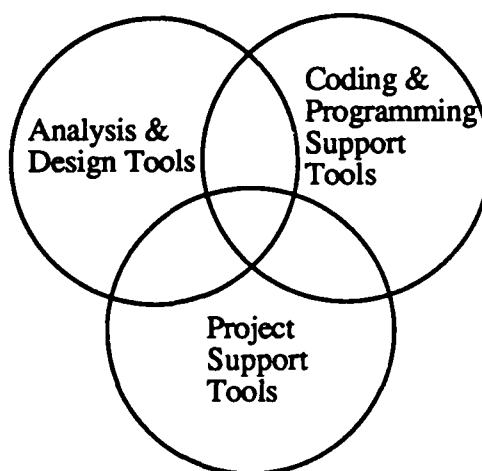


Figure 6a Categories of software development tools.

revised as illustrated in Figure 6b. This revision shows that there are three major categories of CASE tools along with an additional group of support tools that we have identified as project management tools that support the entire environment.

3.1. Front-End CASE Tools.

The front-end or "upper" CASE tools are used in the "front-end" or early phases of software development. This typically consists of tools that support the first three phases of the waterfall model. Usually these tools have extensive graphic capabilities since they are principally used to represent software designs either at the high system level or at the code level. The tools will create a design document that is then provided to the programmers to code from. A growing number of products have an automated interface to the coding phase which is the start of the next category of CASE tools.

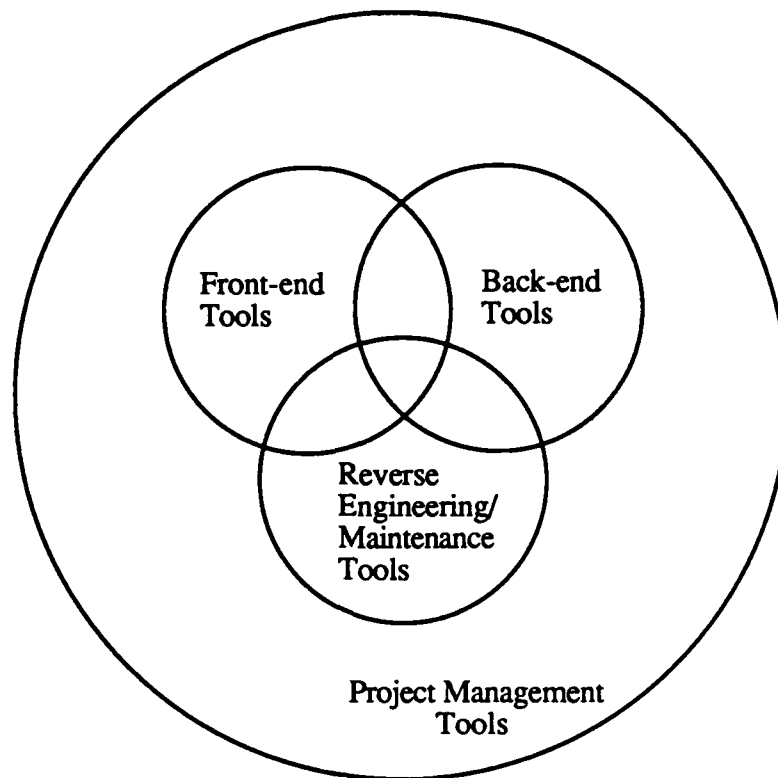


Figure 6b Categories of software development tools (revised).

3.2. Back-End CASE Tools.

The back-end or "lower" CASE tools support the actual generation of code. These CASE tools support the coding and testing phases. Some of the tools automatically generate the code from the design of the front-end CASE tools but these are still quite uncommon. Most of the CASE tools in this category have existed for some time but were not labeled CASE tools. Nevertheless, that does not diminish the fact that these tools assist the programmers in generating code. This category of CASE tools is arguably the most populous group since many of the software development tools have been used for a considerable amount of time. For example, programmers would be hard pressed to develop code without a compiler.

3.3. Reverse Engineering/Maintenance CASE Tools.

This final category supports the "pure" maintenance functions. By this, we mean that these functions consist of supporting existing code. We have identified this category as reverse engineering/maintenance (RE/M). This category is interesting and arguably the most exciting category of the CASE family of tools. One tool may simply reformat the existing code in accordance with new standards or procedures. Another tool may generate a design structure that can be modified which in turn can be input to an automatic code generator thereby creating new code. Still another capability is to process the existing code and create a more structured and understandable program. These capabilities translate directly to monetary savings since they will enable fewer software engineers to maintain far more source code.

4. CASE Product Descriptions.

This section will briefly describe a sample of the representative CASE products. We will first describe in some detail a representative front-end CASE tool, a RE/M CASE tool and finally a CASE tool providing support for the entire life-cycle. These descriptions will identify some of the major capabilities of these individual products. Every effort has been made to describe the capabilities that were *clearly* identified by the product literature, viewed by product demonstrations or explained by the vendor representative. Additionally, this section will also provide a matrix of CASE products capabilities.

4.1. ProKit*Workbench.

ProKit*Workbench is a microcomputer based front-end CASE tool that is marketed by McDonnell Douglas. This system can support multiple concurrent users working on the same project. This enables programming teams to be able to work on the same effort independently.

4.1.1. Hardware Requirements.

ProKit*Workbench operates on IBM Personal Computers (PC), XT or AT, PS/2 (Models 30, 50, 60 or 80). Additionally, it will also operate on 3270-PC (Models 2, 4 or 6) or other microcomputers which are fully compatible. In addition, the microcomputer requires 640 KB Random Access Memory (RAM) and also an expanded memory which supports the Lotus/Intel/Microsoft (LIM) specifications with at least 512 KB of memory. McDonnell Douglas provides an expansion board LIM memory enhancer with its software. A hard disk drive is also required. The vendor also highly recommends the use of a math co-processor chip and the system should use an Enhanced Graphics Adapter (EGA) or Video Graphics Array (VGA) monitor. The use of a plotter is also suggested.

4.1.2. Project Management Capabilities.

This tool provides several features that are useful to manage projects. First, it provides documentation templates or shells that are used to create users manuals, test plans and other reports used by the software development personnel. A text editor is also included in the package which performs standard word processing functions and also allows text to be exported to external files. This tool also provides some configuration management support. An individual worker, responsible for controlling the development effort, is assigned the special "project director" user identification number and password.

4.1.3. Front-end Capabilities.

This tool has extensive front-end design capabilities. It contains three major services that it calls the Analyzer, Data Modeler and the Prototyper. These contain three versions of design generators or diagramming tools that can be used to translate the strategic business objectives into a logical model which is then used to generate the system specifications.

ProKit*Workbench utilizes data flow diagrams (DFD) based on Gane-Sarson graphic conventions and rules. Additionally, it also uses an extended version of Bachman or Chen

entity relationship (E/R) diagrams to produce strategic and detailed data models. The customers are also able to create their own graphic symbols.

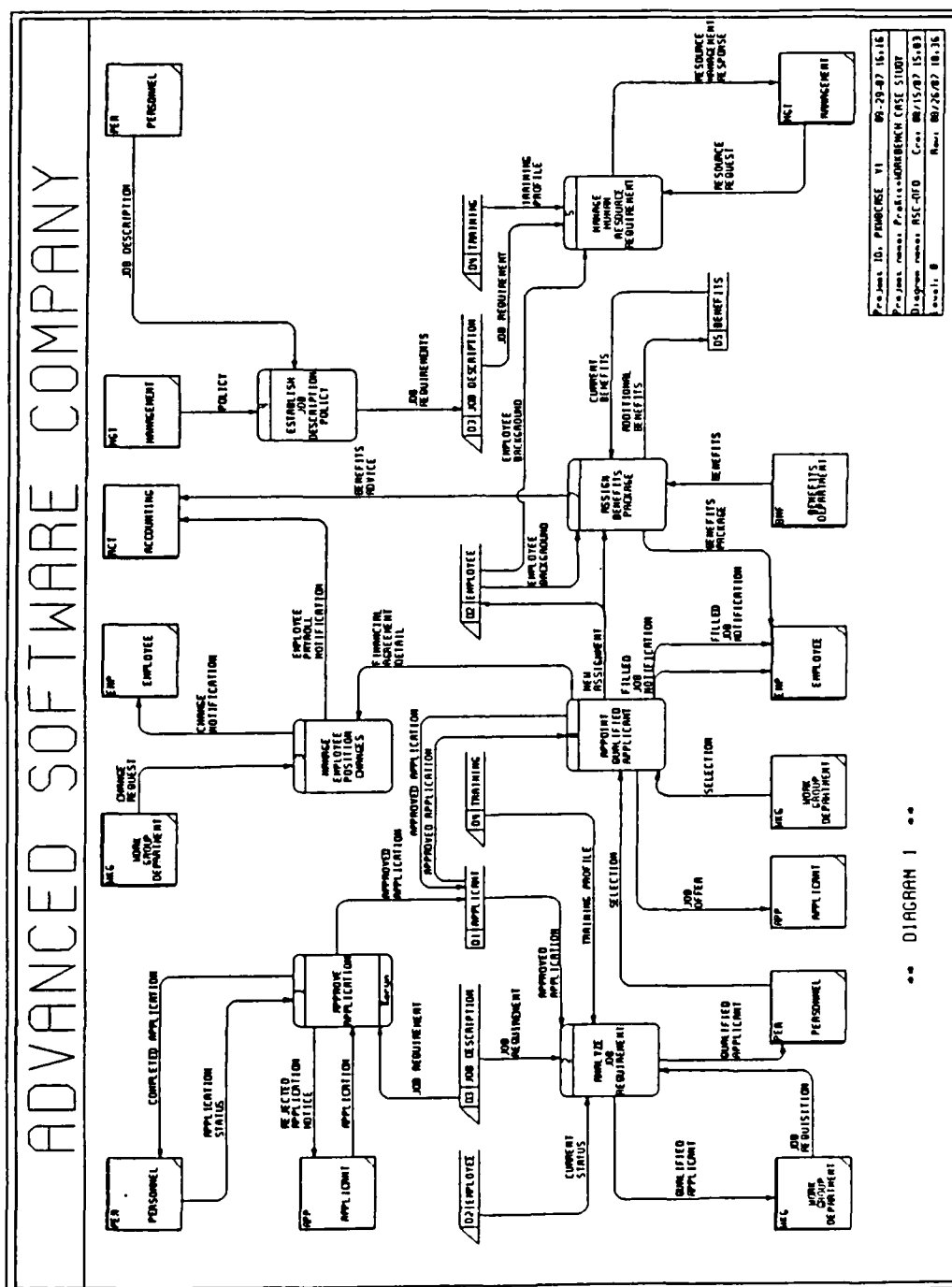
ProKit*Workbench uses a fully integrated active data dictionary that stores information about the processes, relationships, data elements and structures. Using this data dictionary, error reports such as balancing and consistency checking are generated that identify missing paths or incorrect data. Documentation elements are also automatically produced based upon the data stored in the dictionary.

Like all front-end design tools, ProKit*Workbench is heavily graphic oriented and makes extensive use of the graphic illustrations of the system design. Figure 7 contains a sample of one of their data flow diagrams that uses the Gane-Sarson conventions. These diagrams are quickly updated when changes are made by the software developers. The data dictionary is also automatically updated and when the new diagrams are generated.

We feel that this product is representative of the front-end CASE tools that are currently available. It appears to provide the features necessary for use by a small sized organization for the design effort.

4.2. RECODER.

RECODER is a mainframe based reverse engineering/maintenance CASE product which is marketed by Language Technology. RECODER generates new structured COBOL source code from existing COBOL programs. INSPECTOR is another related product that is marketed which augments the RECODER tool. We will briefly describe the RECODER capabilities and mention those that are provided by INSPECTOR. The product literature did not specify which specific IBM mainframes are supported.



4.2.1. Project Management Capabilities.

RECODER provides a text editor and it is also able to automatically generate documentation based upon the produced source code. The product literature did not identify any additional project management features.

4.2.2. Reverse Engineering/Maintenance Capabilities.

RECODER is a highly complex tool which translates existing COBOL source code into structured "cleaned up" code. This is accomplished in a six step process. In these steps, the original source code is translated into a mathematical form called an abstract syntax tree. This step defines which character strings in the program count as well formed expressions and which expressions are equivalent. RECODER then repeatedly scans and reduces the translated program into a tightly organized structured form. Typically, it takes less than 100 passes to transform the program into an optimal design.

From this new design, RECODER then produces a new structure graph and syntax tree representing the structured code. Additional passes are then performed to implement local in-house standards. Finally, a new correctly structured syntax tree is produced, and its COBOL code generator creates the new source code. Figure 8 illustrates a sample structure chart created by RECODER.

The INSPECTOR tool is an evaluation device used to determine which programs have the most promise for processing by RECODER. INSPECTOR produces a report that identifies the programs complexity value using the McCabe Essential Complexity method.

This RE/M tool is highly sophisticated and is advertised to provide a rapid pay-back for the cost of the tool due to reduced maintenance costs of maintaining structured code rather than unstructured code. This tool does not actually modify or enhance the source code or provide a means of updating the abstract design. Rather, the maintenance programmers are able to update the newly structured code which is easier to understand and therefore easier to modify.

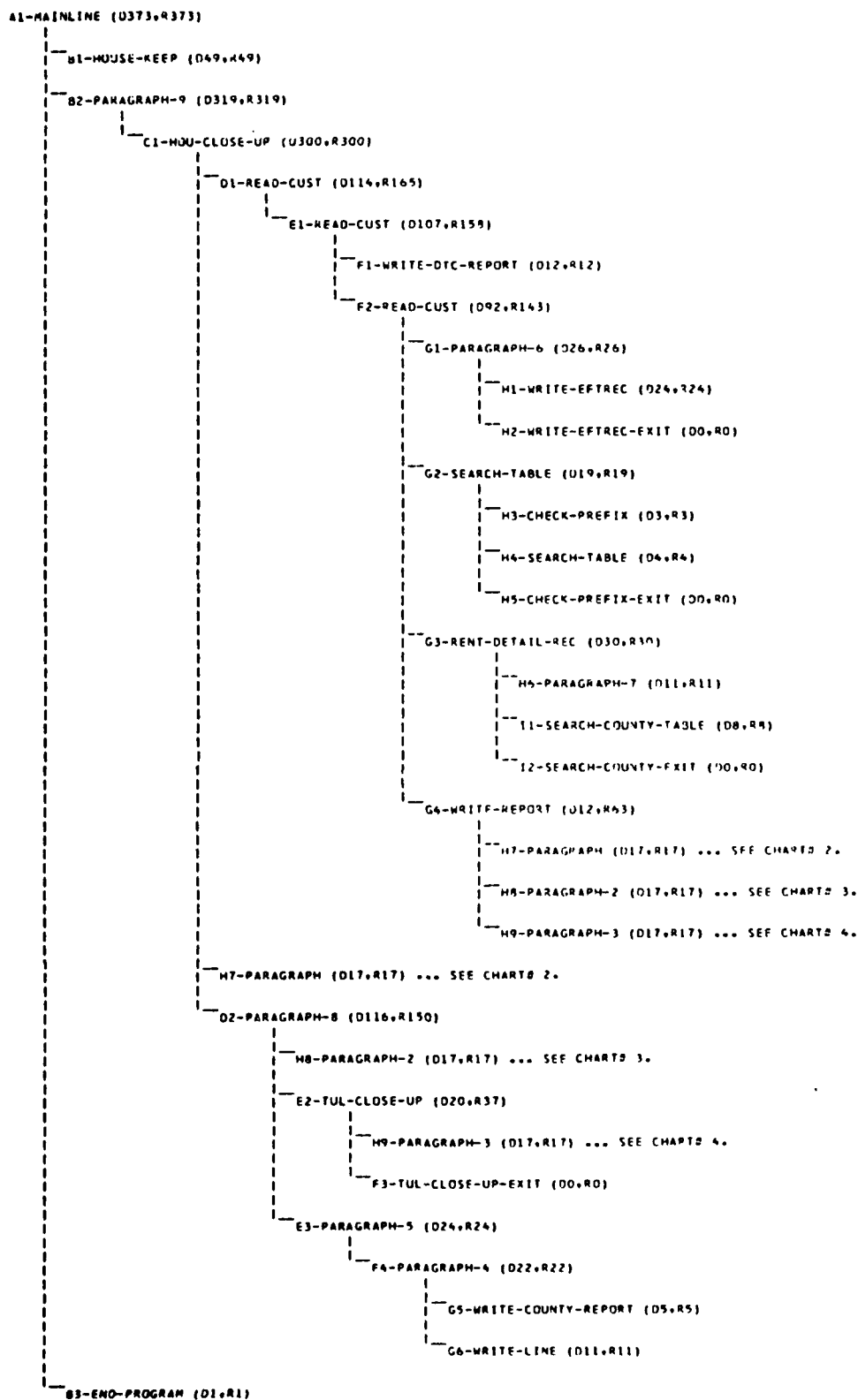


Figure 8 Sample RECODER structure chart.

Language Technology does offer a free Software Portfolio Analysis service. They will process a sample of the potential customer's COBOL source code and return it for the customer's review. This trial test is one of their marketing techniques to provide a potential client an opportunity to judge RECODER's effectiveness.

4.3. Power Tools.

The Power tools series of CASE tools are marketed by ICONIX. They consist of a complement of CASE tools that operate on the Macintosh family of microcomputers. Six products are available that can operate either individually in a standalone mode or in a tightly coupled mode that provides support for the entire software life-cycle.

4.3.1. Front-end Capabilities.

ICONIX provides the FreeFlow and FastTask packages that support the front-end design effort of the life-cycle. These two products are equivalent but each supports a different design technique. FreeFlow supports the DeMarco Structured Analysis (SA) technique and uses data flow diagrams (Figures 9 & 10) and control flow diagrams. The FastTask package supports the Ward-Mellor and Hatley real-time techniques with state transition diagrams (Figure 11), state/event and state transition matrices. Either of these packages will then pass the data through the active data dictionary to the SmartChart package which translates the functional requirements generated by the design package into an appropriate software structure. SmartChart will then create structure charts and generate pseudocode for the system. Built into SmartChart is a language sensitive editor used to process the pseudocode. Documentation is automatically produced as a byproduct of this step.

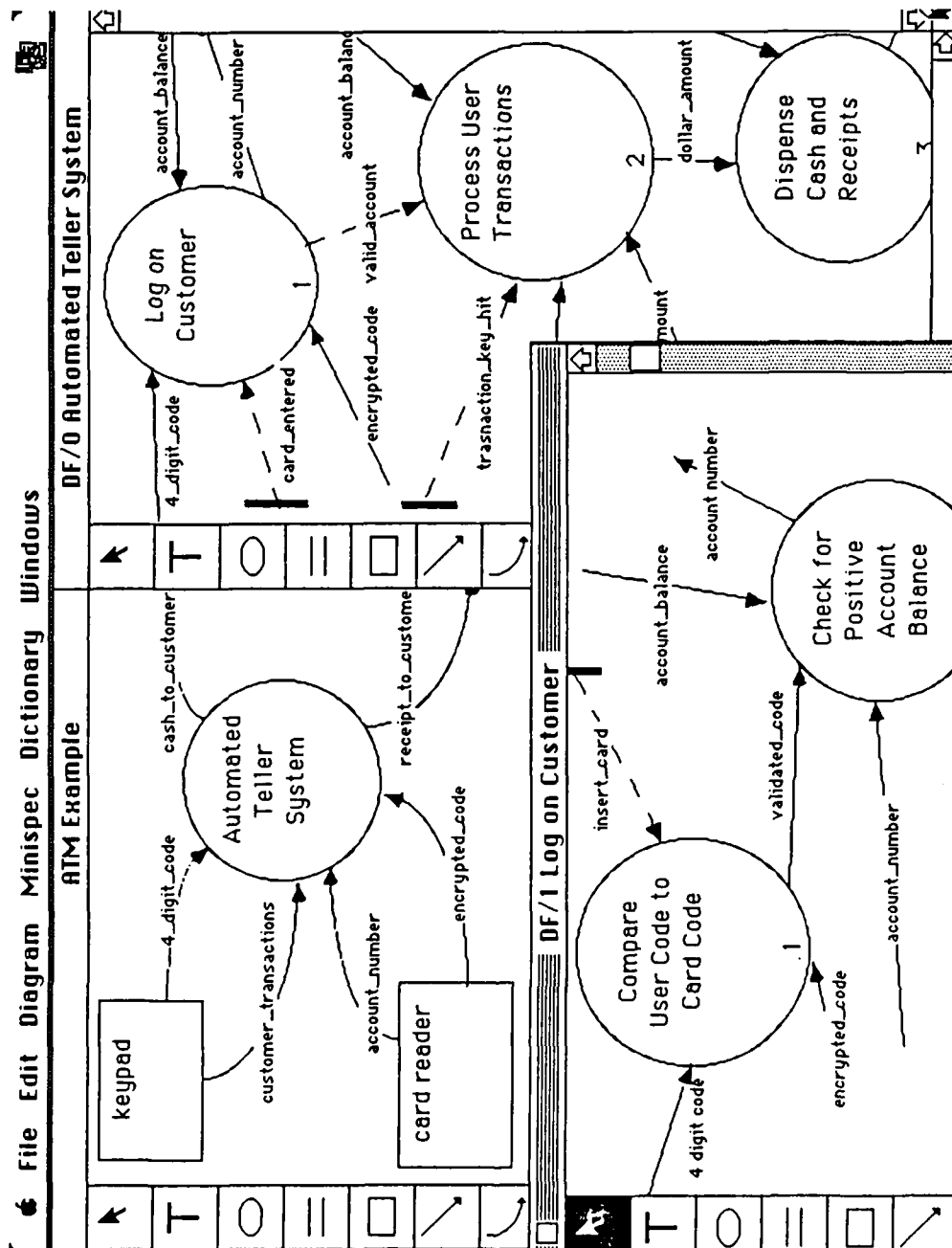


Figure 9 Sample Power Tools leveled set of data flow diagrams.

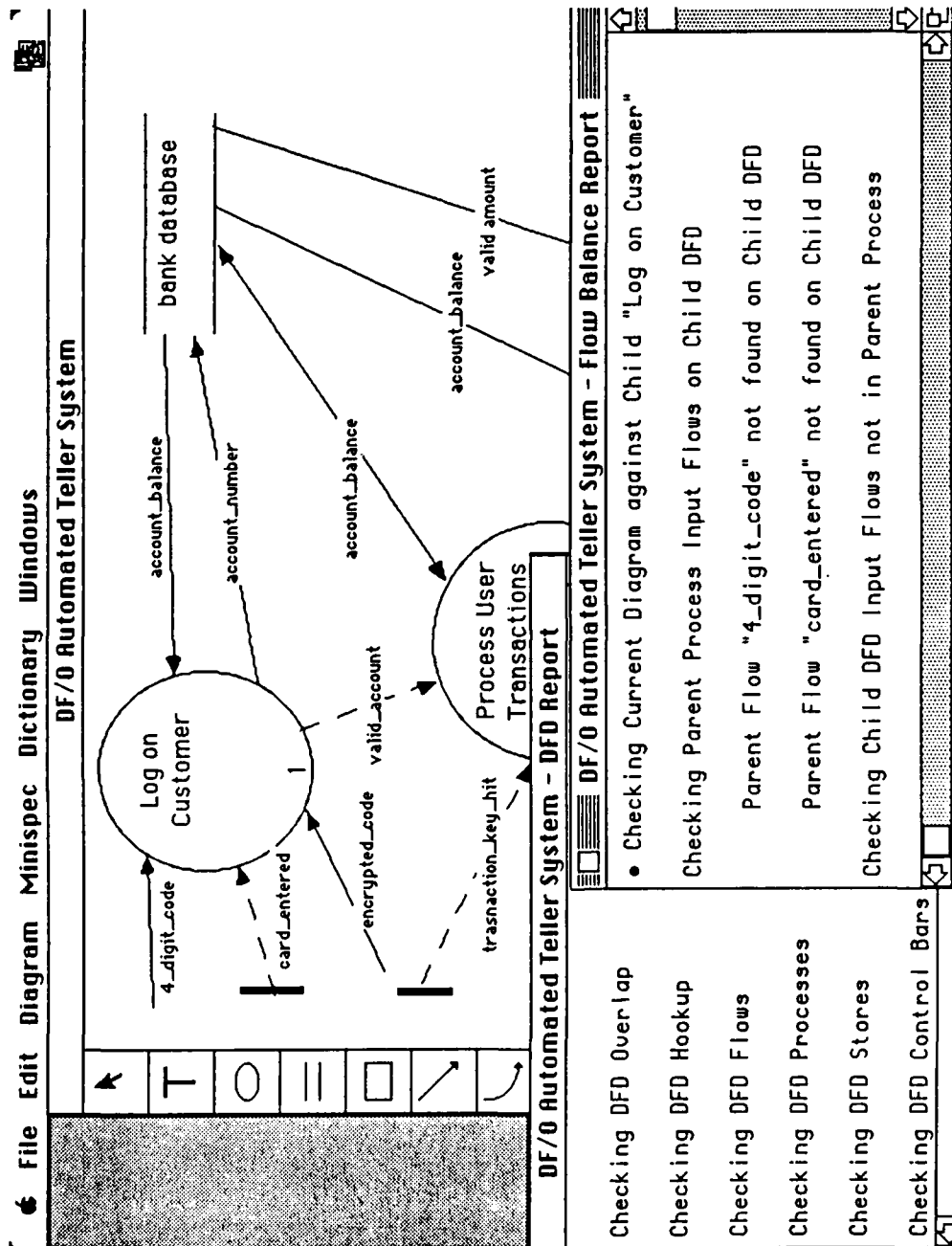


Figure 10 Sample Power Tools data flow diagram with consistency and balance reports.

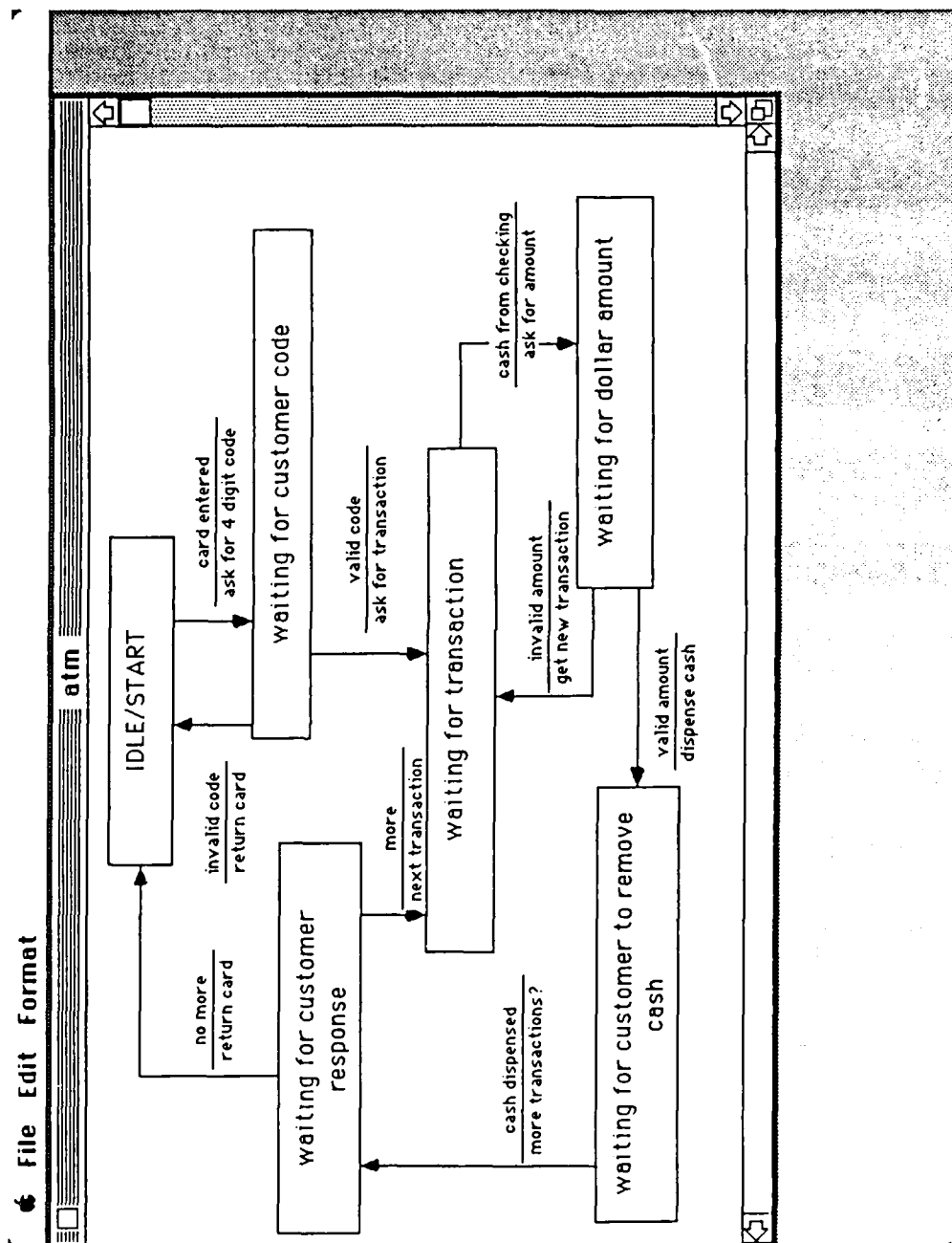


Figure 11 Sample Power Tools state transition diagram.

4.3.2. Back-end Capabilities.

Implementation is then performed by the programmers using the structure charts produced in the previous step. These charts are used as the outline and the pseudocode provides specific guidance. The analysts then write the source code using language templates furnished by Power Tools which includes ADA, C, FORTRAN, LISP, Modula-2, Pascal and Prolog. The SmartChart tool does not actually generate the source code, but is used as an intelligent assistant to the analyst.

4.3.3. Reverse Engineering/Maintenance Capabilities.

Power Tools provides for some maintenance assistance. This consists of using their Strip utility which automatically generates new PDL listings and structure charts from the source code comments. This allows maintenance programmers to use these structure charts and PDL listings as a guide to write new code. This method is helpful as long as the comments in the source are current.

4.4. CASE Products Capabilities.

Table 7 contains a matrix of the CASE products with some of their capabilities listed. It is intended to provide the reader with additional details concerning those products. Many of the details were furnished by Brown [Brow88] and Edwards [Edwa88]. Of the original 54 vendors that were requested to provide information, 33 vendors supplied sufficient data suitable for inclusion in the table. Of the other 21 vendors, some did not respond or responded with insufficient detail or else their product was not appropriate for discussion. The majority of information was obtained from the product literature. This matrix should not be used for product evaluation or purchase guidance. It is presented to provide the reader with an appreciation of some of the product capabilities. Additional information concerning the products may be obtained directly from the vendors. The addresses of the vendors are supplied in Appendix A.

Table 7 CASE products capabilities matrix.

Vendor	Tool(s)	Category	Hardware	Cost	Network	Capabilities
AGS Management Systems, Inc.	MULTI/CAM	Front-end	Mainframe Workstation	High	Yes	PT, TE (Uses Excelerator embedded with-in MULTI/CAM)
American Management Systems, Inc.	Life-cycle Productivity System	Front-end Back-end	Mainframe PC	High	Yes	PT, TE D(DFD, E/R), DD(C, B) SP, RG, SL CG(Cobol), SL, PMG, TCG DG, SCG
Applied Data Research, Inc.	DEPICTOR	Front-end	Mainframe PC	High	Yes	TE, VC D(E/R, CD) DD
Arthur Anderson & Co	Foundation Series	Front-end Back-end RE/M	Mainframe PC	High	Yes	PT, TE, B, VC D(DFD, CD), DD(C, B), CE, SP RG CG(Cobol), CSE, TDM TPA
CADRE Technologies, Inc.	Teamwork Series	Front-end	Workstation PC	Med	Yes	PT D(DFD, CFD, SC, UDI) DD(C, B), RT, CFA, DT
The CADWARE Group, Ltd.	SYLVA Series	Front-end	PC	Unk	Yes	Unk D(DFD, CD, E/R, SD, UDI) DD(C, B), PDL, DT
The Catalyst Group	PATHVU Series	RE/M	Mainframe	High	Yes	DG CR(Cobol), M, SCG, GCR T(ASM to Cobol)

Table 7 (cont'd)

Vendor	Tool(s)	Category	Hardware	Cost	Network	Capabilities
CGI Systems, Inc.	PACBASE	Front-end Back-end RE/M	Mainframe	High	Yes	Project Mgt: Front-end: B, PL, DT D(CD, UDI), DD(C, B) SP, RG, GFS, PDL CG(Cobol) GCR, DG, BC
Computer Sciences Corp	Design Generator	Front-end	PC	Med	Unk	Project Mgt: Front-end: Unk D(DFD), D(C, B)
Cortex Corp	CorVision	Front-end Back-end RE/M	Mini PC	Unk	Yes	Project Mgt: Front-end: Unk D(E/R), DD, SP, RG CG(Cobol), CSE Unk
ICONIX Software Engineering, Inc.	PowerTools Series	Front-end Back-end RE/M	Macintosh	Med	Yes	Project Mgt: Front-end: Unk D(DFD, E/R, SD, SC, CFD) DD(C, B), PDL, CE DG, SCG
IDE	Software through Pictures	Front-end	Workstation	Unk	Yes	Project Mgt: Front-end: Unk D(DFD, SC, SD, E/R, UDI) DD(C, B), PDL
Index Technology	Excelerator	Front-end	Mini PC	High	Yes	Project Mgt: Front-end: Unk D(DFD, SC, SD, E/R, UDI) DD(C, B)
Institute for Information Industry	KangaTool Series	Front-end	Workstation	Unk	Yes	Project Mgt: Front-end: PP, PT D(DFD, SC), DD(C, B)

Table 7 (cont'd)

<u>Vendor</u>	<u>Tool(s)</u>	<u>Category</u>	<u>Hardware</u>	<u>Cost</u>	<u>Network</u>	<u>Capabilities</u>
KnowledgeWare	Workstation Series	Front-end	PC	Unk	Unk	Project Mgt: Front-end: Unk D(DFD, E/R, SC), DD(C) SP, RG
Language Technology	RECORDER	RE/M	Mainframe	High	Yes	Project Mgt: RE/M: TE M, CR(Cobol), DG, PP
LBMS	AUTO-MATE PLUS	Front-end	PC	High	Unk	Project Mgt: Front-end: TE D(DFD, E/R), DD
Manager Software Products, Inc.	Manager Series	Front-end Back-end	Mainframe	Unk	Yes	Project Mgt: Front-end: Unk D(E/R, UDI), DD CG(Cobol), SL, PMG
Matterhorn, Inc.	HIBOL	Back-end	Mainframe	Unk	Unk	Project Mgt: Back-end: Unk CG(Cobol), uploads from PC
McDonnell Douglas	ProKit* Workbench	Front-end	PC	High	Yes	Project Mgt: Front-end: DT, B, PL, TE D(DFD, E/R, SC), DD, SP, RG
Mentor Graphics Corp	Mentor Graphics CASE	Front-end	Workstation	Unk	Unk	Project Mgt: Front-end: DT, AT D(DFD), RT
META Systems	PSL/PSA Series	Front-end RE/M	Mainframe Workstation PC	High	Yes	Project Mgt: Front-end: RE/M: DT D(DFD, DSSD, CD), DD(B) CR(Cobol, Fortran, AS, JCL)
NASTEC Corp	DesignAid	Front-end	Mainframe Workstation PC	Unk	Yes	Project Mgt: Front-end: DT, TE, PT D(DFD, E/R, DSSD), DD(C, B) RT, RA, SP, RG

Table 7 (cont'd)

<u>Vendor</u>	<u>Tool(s)</u>	<u>Category</u>	<u>Hardware</u>	<u>Cost</u>	<u>Network</u>	<u>Capabilities</u>
OPTIMA, Inc.	Design Vision Series	Front-end Back-end	PC	Unk	Unk	Unk Project Mgt: D(DSSD), DD Front-end: CG(Cobol) Back-end:
POLYTRON Corp	Poly Series	Back-end RE/M	Workstation PC	Low	Yes	DT, AT, PP, B, PL Project Mgt: SL, PMG Back-end: GCR
ProMod	ProMod Series	Front-end Back-end RE/M	Mini Workstation PC	Unk	Yes	DT, SA Project Mgt: D(DFD, SC), DD(C), RT, PDL Front-end: SP, RG Back-end: CG(ADA, C, Pascal), CSE RE/M: Unk
RATIONAL	RATIONAL Design Facility	Front-end	Mainframe	Unk	Yes	DT, Project Mgt: D(CD), DD(C, B), RT, PDL Front-end:
Softlab, Inc.	MAESTRO	Front-end Back-end RE/M	Mainframe	High	Yes	PT, AT, PP, VC, TE Project Mgt: D(CD), DD(C), PDL, SP, RG Front-end: CG, CSE, JG Back-end: Unk RE/M:
StarSys, Inc.	MacBubbles	Front-end	Macintosh	Low	Unk	Unk Project Mgt: D(DFD), DD(B) Front-end:
Texas Instruments	Information Engineering Facility	Front-end Back-end	Mainframe PC	High	Yes	PL Project Mgt: D(E/R, PD), DD(C), SP Front-end: CG(Cobol) Back-end:

Table 7 (cont'd)

<u>Vendor</u>	<u>Tool(s)</u>	<u>Category</u>	<u>Hardware</u>	<u>Cost</u>	<u>Network</u>	<u>Capabilities</u>
Visible Systems Corp	VISIBLE ANALYST Workbench	Front-end	PC	Med	Yes	Project Mgt: Front-end: TE D(DFD, CD, UDI), DD(C, B)
Visual Software, Inc.	vsDesigner	Front-end	PC	Unk	Yes	Project Mgt: Front-end: TE D(DFD, DSSD, E/R), DD
YOURDON, Inc.	Analyst/Designer Toolkit	Front-end	PC	Unk	Unk	DT, TE D(DFD, E/R, SD, SC, UDI, CD) DD(C)

Table 7 was completed based upon documentation received from the vendors. Capabilities were not included unless they were specifically identified in the product literature. Some of these products have capabilities that were not included in the matrix since the documentation may have stated it in such a way that it was unclear to the author. I apologize for any inaccuracies or omissions and any errors are the exclusive responsibility of the author.

5. Summary.

This chapter has provided a short discussion about the tools that are available for use by the software professional. It should be apparent to the reader that all of the tools identified within this chapter are CASE tools. They are all useful in software development and are entitled to be categorized as CASE tools. Historically, the last office in an organization to automate inexplicably seems to be the computer support office. Engineers have had their computer aided design tools for quite some time and of course the business community has dozens of versions of their commonly used applications to choose from. The software developers have only recently begun to automate their environment to assist their development efforts. Because of this, the rush is on by vendors to capture the market with their products. This CASE revolution rivals the one of 20 years ago when structured programming techniques were first introduced.

We have presented the major categories of CASE tools for the entire life-cycle. The reader should recognize that the CASE label means different things to different people. For the purposes of this research, the four categories are the front-end, back-end, reverse engineering/maintenance and the project management tools.

Having laid the the foundation of what CASE tools are in this chapter, we are now ready to explore how they may be able to respond to the small organization.

V. THE SMALL ORGANIZATION DEVELOPMENT ENVIRONMENT

Ninety percent of the time things will turn out worse than you expect. The other ten percent of the time you had no right to expect so much.

Augustine's Laws

1. Introduction.

This chapter identifies the CASE tools that are critical to the small software development environment. These requirements are different from those of a typical large organization for at least a couple of reasons. First, there is the obvious lack of personnel which causes certain problems and secondly, there are informal software development attitudes that may exist in a small shop.

We will first present a small organization model that will be used for illustrative purposes and further discussion. This model should accurately represent common small organizations and will be suitable to draw our conclusions. From this model, we will present topics of concern for the software development that are critical to the small organization. While large organizations may share these concerns to a degree, these potential problem areas are critical to the small organization. From these areas of concern, we will then identify CASE tools that should be considered of prime importance to the small organization.

2. The Small Organization Model.

It is impossible to attempt to discuss every small organization. In order to discuss the small organization, it is necessary to present an abstract model that is both understandable

and also captures the principle concepts of small organizations. This model should be germane to all small software development environments even if each of the specifics are not identical. It is interesting to note that this model is also appropriate to many large organizations as well. These large organizations usually develop software with a team concept or structure which will bear a resemblance to our model. Quite often a large organization logically consists of many small independent units.

2.1 Organization.

Our small organization model consists of up to six software development personnel. This model includes one project manager or team leader within the group. The project manager is a "working manager" and the other team members all share in the responsibility of performing the required quality assurance checks, configuration management practices, librarian functions and documentation writing and software testing. Of course all small software development organizations do not consist of the above composition but most small organizations will resemble it. As the reader can see, this team does not adhere to the programming team composition which Weinberg [Wein71] and Brooks [Broo75] identified. It is unusual for a small organization to be organized with each team member formally tasked with the specific roles of the librarian, administrator, tester or other support roles. This section will identify and explain how a small organization may be organized.

2.2. Personnel Responsibilities.

The project manager or team leader is a "working manager" who usually has two major responsibilities. Their first responsibility is to perform the necessary management tasks of ensuring that the project is on schedule, within budget, and that the software development procedures and standards are being adhered to and to supervise the software developers. In addition, the project manager is also a part-time fellow "worker bee" and

must also do his share of actual code generation. The project manager must share in the workload since there is usually more "hands on" work to be done than the management functions and also because he or she is typically an experienced software engineer and is required to assist the other team members. The project manager was typically a team member prior to being promoted to project manager. Thus the project manager will usually continue to stay actively involved with his or her previous duties.

These project managers face special management problems concerning their management of software developers. Licker [Lick85] has provided an outstanding discussion of the difficulties that modern managers of programmers (MOP) face. He explains that programmers are a special breed of workers who have special requirements. He states that they tend to be reclusive and not adept at human relationships. They also dislike paperwork and fundamentally do not like to document their programs. Dr. Licker stated

There is still the difficulty that programmers must document their programs before anyone can understand what the program is or does --- and documentation is not seen as profitable labor by most programmers. It presents no technical challenges and provides no opportunity for learning. Furthermore, no one will cheer reading documentation, especially the manager, who will probably find problems with it.

It is no small accomplishment for managers to successfully supervise programmers in a large organization. It is even more difficult to successfully manage them in a small organization where there may be a lack of formal job descriptions or clearly defined responsibilities. This research effort is not intended to address the management problems of supervising programmers but these problems do exist and will certainly impact the small organizations.

The software engineer team members will design, code and test the software. The senior or lead software engineer will usually be responsible for the system design and perform the design reviews as required. Additionally, since the organization does not have

dedicated personnel for quality assurance and other necessary support functions, these software engineers will also normally perform these duties on a part-time basis.

2.3. Functional Responsibilities.

The small organization is different from the large organization in relation to their functional responsibilities as well. By this, we mean that the large organization will have many software development teams that may be organized by hardware environment, software environment or perhaps by the applications type. The large organization will also have dedicated maintenance support units in addition to segregated quality assurance and support offices. Conversely, the small organization will not have additional support personnel and they will be required to maintain and support the software that they have previously developed. This maintenance activity grows as new software is developed until the small organization is saturated with their maintenance role and are not able to develop new software. For this reason, the small software development environment model is heavily weighted towards software maintenance activities. This model will expend 25% of its activities towards developing new applications software and 75% of its time maintaining already existing code. There are some small organizations that specialize in state of the art sophisticated software that is written once and then discarded, but the majority of the small organizations tend to develop routine software applications which are then required to be supported.

The small organization may also perform a caretaking role for a major software system. For example, they may be required to support a large software system that was previously developed by a third party organization and the small organization is tasked to keep it operational. They ensure data protection and recovery actions are performed in addition to making software enhancements. They would typically only have the opportunity to write small peripheral software programs and utilities to assist them or the customers.

3. Difficulties in a Small Organization.

As previously stated, the small organization has special difficulties that larger organizations may experience to some extent but which are critical to the small organization. Two of these difficulties will be discussed in this section along with some of the problems caused by them. The first difficulty is limited personnel resources and the other is one of the personnel having a casual attitude concerning software engineering practices.

3.1. Limited Personnel.

By definition, the small organization has a limited number of personnel. Our model has up to six members who must share responsibilities for developing the software. This may cause problems since there are not enough personnel to consistently provide adequate backup. The following problem areas are caused by a lack of personnel.

3.1.1. Limited Overlap of Functional Responsibilities.

Since there are few personnel in the software development organization, the individual programmers are required to support a broad range of functional responsibilities. This generates the problem of not being able to have *effective* backup capabilities for the software developers. The individual programmers are assigned wide ranges of functional responsibilities that may saturate or overload the individual programmer. Effectively, backup support is reduced since the backup programmer may not have very much experience in his or her secondary area.

3.1.2. Limited Depth of Technical Knowledge.

Because of the wide area of functional responsibilities that the individual programmers have in a small organization, the programmers tend to develop a broad or shallow level of knowledge about the applications software. This causes them to become generalists rather

that specialists. This "jack of all trades" syndrome inhibits the programmers from specializing or obtaining a deep understanding or highly technical expertise in a particular functional area. The programmers are just not allowed to develop specialized skills which in turn increases the amount of effort it takes to maintain the software.

3.1.3. Shared Software Engineering Responsibilities.

All of the software developers must share the software engineering practices such as quality assurance, configuration management. This is because there is a lack of dedicated personnel to perform these functions. This causes problems since whenever more than one person is responsible for an action, it may create an atmosphere where no one is responsible. These areas of responsibility should be specifically assigned to personnel in a full-time manner since they are so important to the successful completion of the software development efforts. However, this is usually not the case. Most small organizations will tend to be encouraged by the customers to "code and go". This creates future difficulties in maintaining this software. While the programmers must be intimately involved in testing the software and writing the documentation, they should not be responsible for the approval of the project. The approval actions should be removed from the programmer level and instead given to the supervisory personnel with the counsel of an independent quality assurance representative.

3.1.4. Heavy Maintenance Workload.

As previously stated, most of the small organizations are principally concerned with maintaining previously developed software. The small organization must also support the existing software environment. This may consist of merely installing new releases of the operating system and performing the necessary site adjustments. It may also include correcting identified problems with software that was purchased without maintenance support from the vendor. They may also be required to support software previously

supplied to their customers. It is expensive to maintain software since the programmers must first learn what the software is doing and understand it before they are able to modify it. If a programmer does not change a particular portion of software often, he must relearn it. If the software is difficult to understand, or does not adhere to good programming practices or standards, it will probably take the programmer longer to modify the software.

3.1.5. Backlog of Application Development.

The backlog of applications software at organizations can exceed three years. There is usually a lot of software that the customers or users would like to have developed. The small shop also have backlogs that are due in part to their heavy maintenance workload. The higher priority items are accomplished first which is usually correcting the problems with existing software before new software is developed. The cost of software engineers is quite high and therefore management would rather have work backed up than to have idle programmers.

3.2. Informal Development Attitudes.

Even more troublesome than limited personnel is the difficulty of the programmers having casual or informal attitudes concerning software engineering development practices. These attitudes are similar to those held by most programmers 20 years ago. This attitude is manifested by programmers that "don't have the time" to document until the end of a project, perform design reviews, use formal configuration management practices or other engineering practices that should be used for all development efforts. These attitudes were prevalent within the industry before the software engineering discipline was first identified and the benefits were clearly documented by Boehm [Boeh81a] and others. Alternatively, these attitudes can also be caused by informal personal relationships between the supervisors and the programmers. These casual or laissez-faire relationships may also cause poor development practices to be overlooked.

This section is not intended to be a tutorial on the proper software development practices that should be accomplished when developing software. It is instead an attempt to present some of the problems that are faced by the small organization when good software development practices are not used.

3.2.1. Poorly Enforced Development Practices.

Today, some software engineers do not believe that this is a problem in today's enlightened period of structured design and the widespread use of software engineering practices. They believe that programmers consistently utilize good software development practices. It is our contention that in fact many programmers do not use these practices and that this is common in the smaller organizations and is also a present in large organizations. A highly visible example of noncompliance with software engineering practices and noncompliance with established company policies would be the recent episodes at Apple Computer Inc. and also at Microsoft Corporation [Whit88]. Programmers at both of these large companies disregarded their company standards and developed programs that directly accessed the hardware addresses and registers rather than using the "standard" and authorized programming interfaces. In the Apple situation, this resulted in Apple software that would not properly operate on the new Macintosh II hardware which worked correctly with the previous Macintosh machines. In the case of Microsoft, the programmers exploited information about IBM microcomputers to write drivers for several of their products to take advantage of faster execution speeds. This caused some of their software products to incorrectly work with the IBM Adapter Interface which was produced by IBM that was intended to be device dependent. As the the article stated:

The most ironic aspect of all this is that Apple and Microsoft have probably been the industry's two most vocal proponents of device-independent interfaces.

These are two of the more visible examples of nonadherence to established programming development standards. While in the "short run" the software performance was improved, it caused considerable embarrassment to the two companies not to mention loss of revenues. We contend that this casual attitude is very prevalent in the small organization.

3.2.2. Uncompleted Projects.

These projects are not projects that are canceled in progress or only partially coded. These instead are the projects that are apparently complete from the user or customer perspective but in fact are incomplete in accordance with good software engineering practices.

One scenario would be when the customers casually requests a capability from the programmer. Remember that this is a small organization and the users typically have access to the software personnel. The developer tends to want to write code and do things for the users and will typically agree to do the project with little hesitation, even if it requires him to work on it during non-duty hours. There may not be a formal request form completed by the customer. Even if one is completed, it tends to be casually written. The programmer then writes the code, tests it to make sure that it works and shows it to the user. The user is pleased and requests to use the program right away and is provided it with a few instructions from the programmer.

These projects usually have not had the benefit of rigorous software practices being used. This causes problems in the future when the programmer and/or user are no longer with the company but the program is still being used and requires maintenance.

3.2.3. Poor Documentation.

The software does not have to be developed in a small organization in order to suffer from poor documentation. Examples of poor documentation are too numerous to mention.

Albeit, if the documentation was written in a small organization that does not have the rigorous quality assurance and verification procedures in place, the documentation has a far greater chance of not being suitable. We would suggest that there are many programmers that would desire to work in a highly operational environment like the one described above where they are not constantly being forced to develop good documentation. In these environments, the paperwork which most people and especially programmers do not enjoy, is often neglected.

4. CASE Tool Requirements of a Small Organization.

It would appear that all of the CASE development tools are needed in the small organization. We certainly could not disagree with that conclusion. What we will identify here is a candidate minimum set of CASE tools that should be available which responds to the previously identified needs of the small organizations. It is critical that all of the CASE tools used are easy to learn and use due to the shortage of programming personnel in the small unit. Additionally, the CASE tools must not require a significant effort to keep it operational. By this, the tools should not require dedicated resources to update it with future versions of the software or burden the developers with extensive overhead.

4.1. Front-End CASE Tools.

The small organization should have a structured design development tool for each type of software application that is developed. This tool will produce designs that are easy to update as additional design changes occur. All of the design diagrams will be automatically stored in an active data dictionary and accessible to the other CASE development tools.

4.2. Back-End CASE Tools.

An automated source code generator would be extremely helpful. However, this tool is not commonly available. When it is available, it should head the list in this category. The

available tools that should be included at the present are a syntax sensitive editor, online code debugger and test data generators.

4.3. Reverse Engineering/Maintenance CASE Tools.

A code restructuring tool and change impact tools are the premier tools in this category. The programmers must be able to "clean up" the previously written code and also determine what software and documentation will be affected by any software changes.

4.4. Project Management CASE Tools.

The most important tools in this category are used to support configuration management and the documentation text editor. These two tools are needed to ensure that all software and documentation are controlled properly throughout the development life-cycle. The text editor must be consistent through the entire life-cycle of tools and integrated so development personnel are not required to learn and use a different text editor for each individual tool.

5. Summary.

As we have discussed in this chapter, the small organization is really a microcosm of the large organization. This is reflected in the results of the survey detailed in the next chapter. The CASE tools we identified to be used by the small organizations provides relief to some of the problems these organizations currently face. The front-end and back-end tools provides structured methodologies for designing the software which improves the programmers productivity. This will then help eliminate some of the backlog of applications to be developed. These tools also provide automated or improved documentation support which helps the small organization programmers produce effective documentation. Since the software engineering responsibilities are shared by the individual team members, the CASE tools act as an embedded quality assurance representative who

enforces the appropriate software engineering disciplines. The reverse engineering and maintenance tools allow the programmers more time to expand their technical knowledge of the software. These tools provide the ability for a reduced number of programmers to be responsible for maintaining more software. This enables a meaningful overlap of the functional areas of responsibilities to be performed. The project management tools provides the additional automated support for controlling the software which is needed by the small organizations.

These small organizations appear to be at the head of the spear of software engineering. By this we mean that while the large organization will usually adhere to established software engineering practices, they have the ability to utilize their entire staff of software development resources, or the "shaft" of the software engineering spear. They therefore have built-in redundancy and backup support for difficulties that may arise. The small organization on the other hand, must solve their problems with only the "head" of the spear. This analogy provides a new meaning of being "on the leading edge of technology". It is for these reasons that it is critical to improve the development productivity and software development practices of a small organization.

To solve the problems of the small organization, the reader may suggest that the small organization merely "do their job correctly to begin with" or other simple but ineffective advice. The difficulty lies in that most software professionals would prefer do a good job but they may not be able due to institutional roadblocks. This research is an attempt to identify those currently available CASE tools that may be of use to the small organization. We will now turn our attention to what is available and the opinions that software professionals have concerning the use of the current CASE tools.

VI. INDUSTRY SURVEY

You can observe a lot by just watching.

Yogi Berra

1. Introduction.

In support of this research, software professionals were surveyed who have experience using one or more of the CASE products. We were interested in obtaining their opinions concerning the appropriateness of the use of these CASE tools by small organizations. Appendix B contains a copy of the cover letter and questionnaire that was used in this survey.

We wanted to obtain results from experienced users since they are in a better position to provide informed opinions concerning the use of these tools. We understood that our results could be biased because the respondents have already shown their endorsement of the use of CASE tools by their purchase of them. We felt that it was more important to receive knowledgeable, experienced opinions from informed users rather than requesting opinions from software professionals who have not had the opportunity to use these tools.

2. Survey Methodology.

We requested names and addresses of representative clients from the CASE vendors. Some of the vendors were unable to cooperate, while others were able to assist provided that alternative procedures were followed. Those questionnaires were mailed en masse to the vendors who in turn forwarded them to their customers. This was done in order to protect their customers confidentiality.

We chose to mail a relatively small number of questionnaires to a broad variety of different CASE product users. This was done in order to avoid responses that reflect heavily from a single CASE product. If that CASE product was insufficient, we could have received heavily unfavorable comments because of that individual product. Additionally, if we sent a large number of questionnaires to a few different vendors, we may not have received enough responses. Table 8 reflects that some of the questionnaires seemed to have been lost and were never returned. We did not want to rely upon just a few vendors for this reason. Table 8 identifies the distribution of the questionnaires. 12 different vendors participated with our survey. The reason that we received more responses than the questionnaires sent for the Nastec tool is because we received responses about the Nastec tool when we sent the questionnaires to addresses supplied by another vendor. The "direct" distribution method identified in the table was the process whereby we mailed the questionnaires directly to the customers who were identified by the vendors. The "vendor" method of distribution consisted of mailing the questionnaires to the vendors who in turn mailed them to their customers. It would appear that the questionnaires which were mailed by the vendors were not as well received as those which were mailed directly. The response rate for the direct mail was 53 out of 93 for a response rate of approximately 57% while the questionnaires which were mailed by the vendors responded with only 8 out of 59 for an abysmal 14% response rate. The "other responses" were responses received as a result of customers completing a questionnaire for additional CASE products that they were also using. The negative responses were not negative in the sense that they did not recommend the use of CASE tools, rather these figures are a result of the responses that either had not actually installed the products or were unable to complete the questionnaire. The survey produced 76 usable completed questionnaires for our research. We were quite satisfied with the response level from the customers surveyed. Many of the respondents spent considerable time and effort to explain some of their answers which provided excellent insights into their opinions and

were most helpful.

Table 8 Questionnaire distribution.

<u>Vendor</u>	<u>Distribution Method</u>	<u>Number Sent</u>	<u>Responses</u>
AGS	Vendor	10	0
CADRE	Direct	16	10
Catalyst	Vendor	10	0
ICONIX	Vendor	12	7
IDE	Vendor	12	1
Index	Direct	13	4
KnowledgeWare	Vendor	15	0
Language Technology	Direct	8	3
LBMS	Direct	14	8
McDonnell Douglas	Direct	11	11
Nastec	Direct	11	12
Promod	Direct	20	5
Other Responses	---	---	15
Total		152	76
Negative Responses		---	6

3. Results.

Appendix C contains the data from the 76 usable completed questionnaires we received. It also contains an explanation of what the data value codes represent for each question's responses. This section extracts and presents portions of the results. We included all of the data from the 76 completed respondents in Appendix C to allow the interested readers to perform data reduction of additional questions of interest.

The results of the survey can be categorized into four different areas. The first category

of information concerns the organization's demographics. This data includes the organization's size, individual group size and details concerning their use of formal development practices. The second area identifies their opinions about using the specific CASE tool in the small organization. The third area contains their opinions about using the CASE product family by the small organizations. Finally, the respondents also provided additional information about what they would like to see improved with the CASE tools. The following sections will present the results and provide graphs of some of the results.

3.1. Organization Composition.

The first area of the survey consists of the demographics of the respondents organizations and their software development environments. This section will discuss the results of the answers as they pertain to the organization and development environment questions in sections I and II.

Approximately 85% of all respondents identified themselves as managers or project leaders and only 15% as analysts or "others". Hopefully this will provide us with a "bigger" picture of the use of the current CASE tools.

3.1.1. Organization Size.

This research effort focuses on the small organization and we would prefer to have received most of the responses from representatives of the smaller organizations. Unfortunately, most users of the CASE products tend to be in larger organizations. Approximately 55% of the respondents work in organizations with over 100 software development personnel. Economics could be the principle reason for this fact since the tools tend to be relatively expensive.

We discovered that most of the software groups were small (Figure 12). 33 of the 76 responding organizations were made up of 15 or fewer software development personnel in their groups. 18 of these groups consisted of fewer than 7 personnel. This reflects that

most software development is performed by small groups within even large organizations.

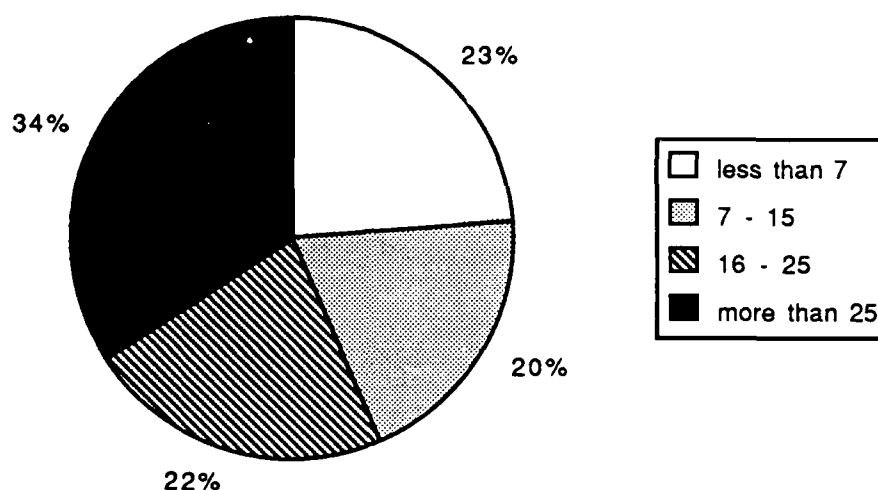


Figure 12 Reported sizes of the software development groups.

According to the respondents, the average software development group consists of nearly 30 people. Figure 13 illustrates the average composition of the software development group. While the size of this average group varies by organization, we anticipate that the ratios will be consistent. It is significant that the average for both the quality assurance and documentation/technical writers combined is only about 4% of the development group. Other personnel identified by the respondents included operations personnel, PC support, communications specialists, project control personnel and also consultants.

Another indicator that most of the software is developed by small groups are the responses to question 8 in section II. Most of the respondents (65%) stated that the software efforts are performed by less than six people. The average experience level of the programmers in the organizations that responded was quite high. 57% of the respondents stated that most of their programmers had over five years experience.

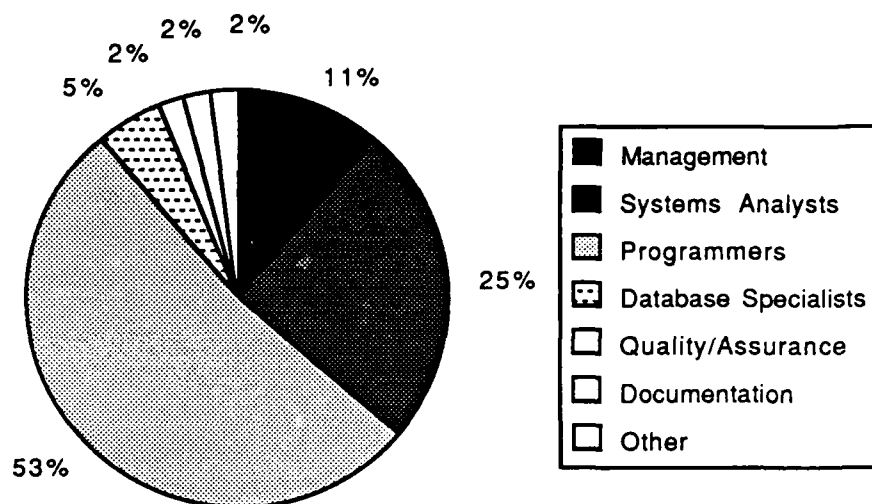


Figure 13 Average composition of the software development group.

3.1.2. Software Project Characteristics.

We received an appreciation for the types of software efforts performed by our respondents. The answers were about equally divided between being responsible for developing software or responsible for both development and maintaining software. No one stated that they were responsible only maintaining software. The technical difficulty of their projects were either technically difficult (38%) or of average difficulty (62%). None responded that they developed simple projects. Most (82%) of the projects took over six months to develop. The majority (58%) of the organizations support a combination of mainframe, minicomputer, microcomputer and network hardware suites.

3.1.3. Organization Software Development Practices.

We were interested in the existing software development practices that were in place prior to the installation of the CASE tools. We discovered that most of the respondents work in organizations that use established software development methodologies. From this we can project that they also use software engineering practices. The decision to purchase the CASE products were about evenly divided between the managers (44%) and the

technical staff (35%) with about 16% decided by a combination of the managers and technical staff together. It was interesting to note that a two of the decisions were prompted by customers of the organizations.

Most (64%) respondents stated that their organizations had company programming standards in place prior to installing the CASE tools and of those with standards already in place, 71% claimed that the standards were either always or usually followed. Formal software design methods were in place in 43% of the installations and of these, 76% stated that the methodology was either always followed or usually followed. These figures indicate that approximately 47% of the installations have programming standards that are usually or always followed and about 34% of the sites either consistently followed or usually followed a specific programming methodology. These statistics are reinforced by the fact that the existing working environment (Figure 14) was about evenly divided between being informal (46%) and somewhere in between (45%) being highly structured/formal and informal. This results in only about 9% of the organizations being highly structured and formal.

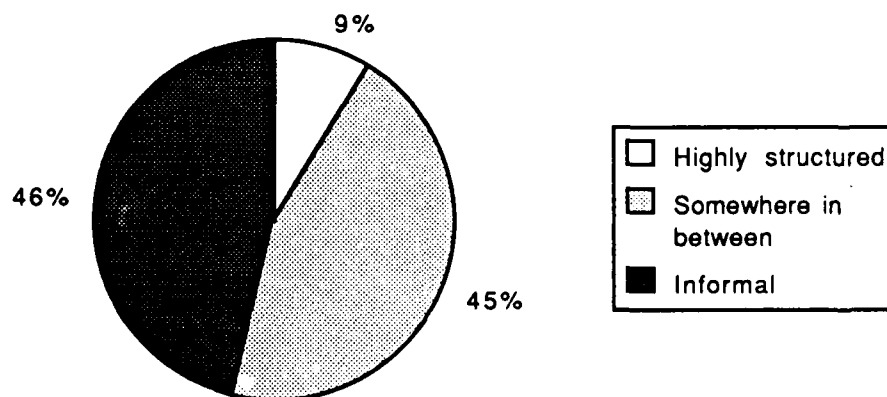


Figure 14 Level of organizational software development formalisms.

3.2. Use of CASE Tools in General.

The response to using CASE tools by small organizations was overwhelmingly positive

(Figure 15). Over 87% of the respondents recommend that CASE tools should be used by a small organization. This is a resounding endorsement of the use of CASE tools. This response was further emphasized when over 85% of the respondents stated that it was a wise decision to install these CASE tools.

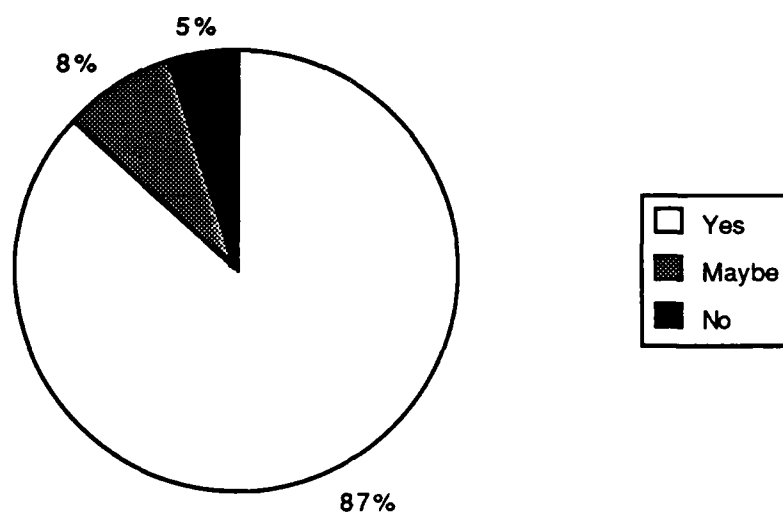


Figure 15 Recommendation for the use of CASE tools by small organizations.

We also discovered over half of all personnel in the software shops are using the CASE tools (Figure 16). It is very interesting to note that in 17% of the software shops, all of the personnel use their CASE tool. This suggests that not only are the CASE products being purchased, but once bought are being aggressively used.

There was a significant difference in the reaction to the CASE tools by the senior and junior programming personnel. 81% of the more experienced personnel favored the use of the CASE tools, while only 51% of the junior personnel favored the use of these tools. This may be a reflection that the junior personnel not fully understanding just how beneficial these products are, while the more senior personnel have had the opportunity to experience problems which could have been prevented by using these tools. Perhaps the junior personnel are more highly trained in the current software engineering disciplines already and do not see the need for these tools. Unfortunately, our questionnaire did not

address this issue in more detail and so this topic could be of interest for future research.

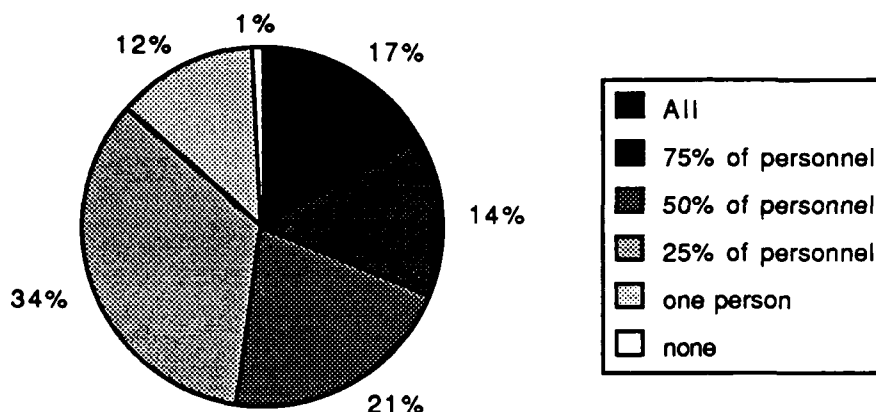


Figure 16 Amount of personnel who use the CASE tools.

We queried the users about the impact felt by their organizations when they installed their CASE tools. Only 15% of the respondents stated that extensive modifications were required (Figure 17). However, over 95% of the them stated that the level of modifications

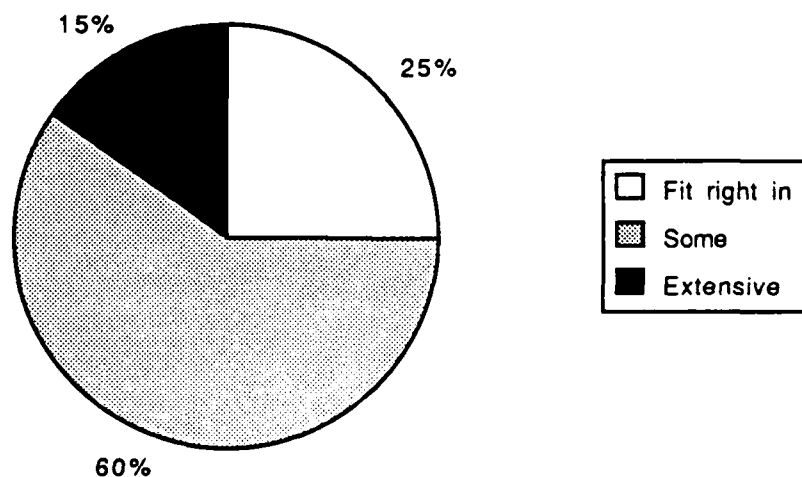


Figure 17 Modifications to the organizations caused by installing the CASE tools.

experienced were what they anticipated. It would appear that the CASE tools fit into existing methodologies fairly well. This leads us to believe that the software developers

performed product evaluations to minimize installation impacts.

We asked the respondents in which phases of the software life-cycle they use their CASE tools (Figure 18). 67% of the tools are used in the definition of requirements phase; 82% are used in the analysis phase; 74% are used in the design phase; 22% are used in the coding phase; 16% in the testing phase; and 19% are used in the maintenance phase. Additionally, the respondents also specifically identified the generation of documentation as a separate use of the CASE tools. Most of these tools contain aids to produce documentation but two of the respondents felt that this was such a significant use of the tool that they identified it as a separate and distinct use. Those totals reflect that the CASE tools are used for more than a single phase. The small percentages for the later phases reflect that most of the tools used are those for the front-end design effort.

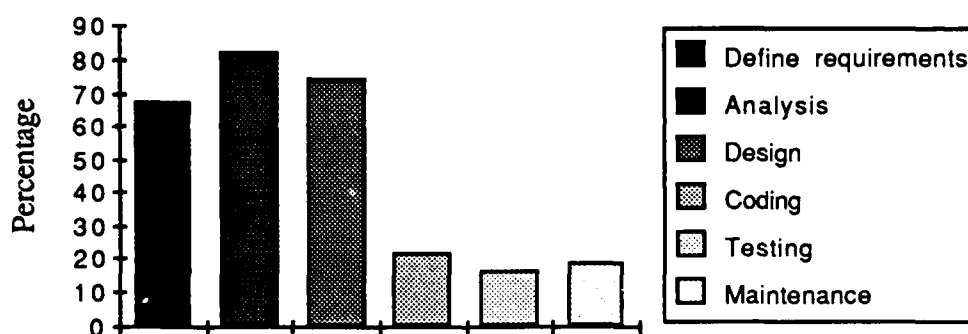


Figure 18 Phases in which the CASE tools are used.

Table 9 contains the cumulative results for all of the tools surveyed for question 12 in section III. The five columns identify the number of responses we received and the average score is also listed. We are providing these cumulative averages here for this group of questions for all of the tools surveyed. The specific totals for each tool are identified in the next section.

Table 9 Combined results of opinions provided for all CASE tools.

<u>No.</u>	<u>Question</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>Ave</u>
12a.	Ease of use	22	29	16	7	2	2.2
12b.	Helpful for requirements definition	26	24	12	5	8	2.3
12c.	Enhances productivity	27	31	14	1	3	2.0
12d.	Indispensable for analysis	22	22	21	5	5	2.3
12e.	Tool is too rigorous	1	7	18	20	27	3.9
12f.	Personnel are resistant	1	6	17	24	28	4.0
12g.	Personnel should use it	26	21	17	7	4	2.2
12h.	Improves software maintainability	28	25	15	2	4	2.0
12i.	Good economic value	26	20	17	8	5	2.3
12j.	Should use for coding	15	7	16	13	20	3.2
12k.	Good configuration management	12	19	21	10	8	2.8
12l.	Should use for testing	12	10	18	11	21	3.3
12m.	Design capabilities needed	20	30	15	4	5	2.2
12n.	Well suited for maintenance	17	21	15	10	10	2.7
12o.	Appropriate for small team	30	31	10	2	3	1.9
12p.	Other tools more appropriate	6	9	24	14	18	3.4

3.3. Use of Specific CASE Tools.

This section identifies the results of the survey for the specific CASE tools (Table 10). We have identified ten categories of the CASE tools. One for each of the nine different tools for which we received responses and a tenth category to include the other responses that we received. Additionally, we identified in Table 10 how many responses were received for each tool.

3.4. Additional Comments Received.

We received numerous insightful comments from our respondents that provided clear visibility into their use of their CASE tools and their software development environments. These comments ran the gamut from CASE tools being precisely what was needed to one comment that explained that they were dissatisfied with their particular CASE tool but were

Table 10 Response averages for the individual CASE tools.

Question Number	Automate Plus	Design Aid	Excelerator	Power Tools	Prokit*Workbench	ProMod	Recorder	Software thru Pictures	Teamwork	Extra
Number of Responses	8	12	4	7	11	5	3	1	10	15
12a.	2.1	2.4	3.3	1.6	2.5	3.0	1.3	2.0	1.5	2.2
12b.	1.8	1.9	1.8	2.9	1.8	2.0	5.0	2.0	2.4	2.7
12c.	1.6	2.3	3.0	1.6	2.0	2.6	1.7	3.0	2.2	1.5
12d.	1.5	2.2	3.7	1.9	2.0	2.6	3.7	2.0	2.0	2.9
12e.	4.0	4.0	3.0	4.1	3.7	3.2	4.0	4.0	3.7	4.3
12f.	4.1	4.1	4.0	4.3	3.7	3.2	4.0	3.0	4.2	3.9
12g.	2.0	2.5	3.8	2.1	1.7	2.4	1.3	3.0	2.7	1.9
12h.	1.6	2.3	2.3	2.0	1.7	3.6	3.7	3.0	2.1	1.3
12i.	2.1	2.2	3.5	1.7	1.9	3.8	3.0	4.0	2.7	1.7
12j.	4.1	2.6	3.5	2.8	3.8	4.2	2.0	3.0	3.5	1.6
12k.	2.4	2.9	3.3	3.1	2.2	4.4	2.7	3.0	3.3	2.0
12l.	3.8	3.5	3.5	3.8	3.4	4.6	1.3	3.0	3.2	2.5
12m.	1.5	2.3	2.3	1.9	2.5	3.0	3.3	2.0	2.5	1.9
12n.	3.1	2.6	2.0	3.0	2.9	3.4	1.0	3.0	2.7	2.3
12o.	1.4	2.1	2.3	1.4	2.1	2.6	2.3	2.0	2.0	1.7
12p.	4.4	3.0	2.3	3.4	3.3	2.4	3.7	3.0	4.1	3.4

going to continue to pursue the use of CASE tools since they were convinced that they were effective. This section will attempt to identify and quantify comments received and list those areas and capabilities of CASE tools that were found beneficial or are desired by the respondents. These results were principally obtained from questions 9a, 9b and 11 in section III.

The respondents identified several major types of capabilities that are important to them. These capabilities were either identified as being an existing beneficial capability of the CASE tool or a capability that they would like the tool to have. The following sections explain the major categories of capabilities listed. Appendix C contains the comprehensive list of the features identified.

3.4.1. Overall System Capabilities.

The respondents identified five areas within the overall system that are of interest to them. First, the overall operation speed of the tool is important. They were concerned with execution speed, physical security and overall general improvements. Secondly, they were concerned with the tool's interfaces. They desire good user interfaces, seamless integration with other CASE tools, and effective interfaces with existing systems and databases. Third, the data dictionary is very important to them. Nearly half of the respondents said that the data dictionary was one of the most useful features of the tool. They also identified additional data dictionary formats and more powerful data dictionary functions as being desirable. Fourth, they are interested in database capabilities. They expressed interest in data normalization aids and additional database design capabilities. Finally, they expressed their desires for enforced formalisms and structured methodologies. They feel that it is important that the tools provide these disciplines.

3.4.2. Project Management Capabilities.

The respondents identified several areas of interest. The topics receiving the most

interest included better word processing capabilities, report generation, automated documentation generation, project management capabilities in general, configuration management and also requirements tracking. Additionally, they mentioned project tracking, estimation and measurement aids.

3.4.3. Front-end Capabilities.

The graphics features were by far the most frequently identified front-end capabilities mentioned. Over half of the respondents listed graphics as one of the most useful features of their CASE tool. Several other features also received repeated attention. These included error checking, diagramming capabilities and consistency checking. Many other features received mention which included ADA, prototyping, and real-time support.

3.4.4. Back-end Capabilities.

The principle capability identified in this category was a code generator. Two respondents thought it was the most useful existing feature (obviously they were using a back-end CASE tool), and nine others desired a code generator in their tool. Others that were mentioned included a context sensitive editor, structured testing aid, test data generator, and ADA support.

3.4.5. Reverse Engineering/Maintenance Capabilities.

The respondents identified two major areas in this category. They listed the general reverse engineering capability and also identified aids to ease the burden of maintaining existing software. There was a definite lack of specific comments provided for this category. We believe that a contributing cause for this is the fact that most of the respondents use front-end design tools and therefore are unaware of the available capabilities of this category of CASE tools.

VII. CONCLUSIONS

For every problem there is one solution which is simple, neat, and wrong.

H. L. Mencken

This thesis has sought to review some of the commercially available CASE products for use by small software development organizations with less than seven personnel. We performed this research in three phases. The first consisted of reviewing pertinent literature relating to software engineering and software development methods and practices. We then reviewed a selection of CASE product literature and observed product demonstrations. Finally, we performed a survey of CASE tool users within the industry. This survey, while displaying a lack of rigorous validity, provides us with an appreciation of the opinions held by these software professionals concerning the use of CASE tools.

1. Findings.

The survey provided us with an overwhelming endorsement for the use of CASE tools by the small software development organization. It was surprising to us to see the strength of their conviction. Over 87% of the survey respondents recommended their use and 8% of the them stated that the CASE tools should be used under certain conditions. Only 5% or 3 of the 76 respondents stated that they should not be used by the small organization.

The CASE tools provide graphic support that is highly desirable to the users. These graphic representations are the heart of this generation of CASE tools. They provide graphic illustrations of the system and software design that are quick and relatively easy to

update. This alone has provided extensive productivity gains to the software developers. The use of active data dictionaries has provided mechanical support to the programmers to identify dangling or unidentified functions, execution paths or variables that in the past may have not been discovered until testing or user execution. This has provided for cleaner and more accurate code.

Our research of the literature has also endorsed the use of these CASE tools. Most of the tools appear to adhere to and encourage the use of the software engineering disciplines that are recommended by leaders in the field. It appears that the vendors are for the most part providing the software development structure and discipline desired.

This first generation of CASE tools appear to satisfy the special concerns of the small software development organization. These tools assist the small shops in overcoming the difficulties caused by their shortage of personnel. These tools improve programmer productivity thereby enabling them to produce and maintain more software. These tools also provide support for the more structured software development environment that is critical to the small organizations. These tools provide development methodologies to properly implement good software engineering practices. In this respect, these CASE tools act as an embedded quality assurance inspector which guides the software development in a real-time mode. While the tools are not perfect, they provide solutions to many of the problems faced by the small organization.

We have also discovered that the CASE tool industry is currently in a relatively early stage of evolution desperately trying to mature and stabilize. For example, CASE tools have reached the point of evolution where they are recognized and seminars are held throughout the country about them. The industry is continually hosting professional seminars to discuss relevant CASE issues. Of significance is the fact that vendors are now attempting to establish industry standards for their tools to interface with each other. They are just starting this effort, and it is encouraging to observe this process. This apparent cooperation should assist in improving the integration between the different CASE tools

and help vendors who are developing new products.

2. A Comment Concerning Future CASE Tools.

Based upon the responses received and literature researched, we would like to offer a comment concerning future CASE tools. First, the tools of the future must be user friendly beyond marketing claims. They truly need to be easy to learn and operate. The software engineer's time is limited even in the larger organizations. Secondly, the vendors must continue to pursue better integration between each of their individual tools and with other vendors tools. Hopefully in the future, the software personnel will be able to pick the most appropriate design tool from his inventory for an individual application and use it without concern whether it will interface properly with any of his back-end support tools. This mix and match approach needs to be aggressively pursued since there is no one "perfect" design method suitable for every application. If this is not accomplished, programmers will be forced to either go without or force a tool to be used inappropriately. Finally, we encourage all software professionals to aggressively pursue the use of reverse engineering and maintenance tools. These tools show exceptional promise and appear to offer tremendous benefits in maintaining software. We are being overrun with software applications and unless we are able to maintain the previously developed software, we may discover that we are developing disposable software. That is, software that is used one time and then discarded.

3. Future Research.

We were obviously limited in both time and resources for our research and were not able to accomplish all that we would have preferred. Additionally, we discovered interesting ideas after we had begun the effort that could not be included. We now offer a few suggestions for future research.

First, the survey questionnaire used was our first effort at discovering opinions of the

software industry and can certainly be improved. The data that we received was excellent and provided a tremendous amount of information. In fact, several of the answers suggested additional areas that would be interesting. For example, why did the junior programming personnel have a more unfavorable opinion about the CASE tools? It would also be enlightening to obtain more information about the existing company standards and programming methodologies both before and after the CASE tools were installed.

Secondly, we feel it would be constructive to perform a rigorous experiment with several programming groups to discover the impact of using CASE tools. This would be similar to Boehm's [Boeh81b] excellent effort where he had two different groups of graduate students develop a software project where one used the large-scale software engineering procedures and the other group did not. It would be beneficial to observe and quantify improvements that are attributable to using CASE development tools.

Finally, we made no effort to evaluate the economic impact of using CASE tools. We felt that since the CASE industry is so young that it would not be possible to adequately comment. We think that as the industry matures and stabilizes, that an economic review will be possible if experiments are performed which are similar to the one we referred to in the last paragraph.

Our research has discovered an overwhelming endorsement for the use of Computer Aided Software Engineering (CASE) tools by the small software development organization. The literature review and the opinions expressed by the 76 respondents to our survey indicate that we as computer scientists should aggressively and actively advance future research and utilize this newest generation of software development productivity tools. We leave as our challenge to the readers to obtain additional information concerning these tools and to integrate them into their development efforts.

LITERATURE CITED

- [Boeh73] Boehm, Barry W., "Software and its Impact: A Quantitative Assessment," Datamation, 19, 5, (May 1973), pp 48-59.
- [Boeh76] Boehm, Barry W., "Software Engineering," IEEE Transactions on Computers, C-25, 12, (December 1976), pp 1226-1241.
- [Boeh81a] Boehm, Barry W., Software Engineering Economics, Prentice-Hall, Inc., Englewood Cliffs, NJ, (1981).
- [Boeh81b] Boehm, Barry W., "An Experiment in Small Scale Application Software Engineering," IEEE Transactions on Software Engineering, SE7,5, (September 1981) pp 482-493.
- [Booc87] Booch, Grady, Software Engineering with Ada, 2ed, Benjamin/Cummings Publishing Company Inc., Menlo Park, CA, (1987).
- [Broo82] Brooks, Frederick P. Jr., "The Mythical Man-Month. Essays on Software Engineering," Addison-Wesley Publishing Company, Reading, MA, (1982).
- [Broo85] Brookshear, Glenn J., Computer Science: An Overview, Benjamin/Cummings Publishing Company Inc., Menlo Park, CA, (1985).
- [Brow88] Brown, Alice C., "Review of the Availability of CASE Tools for the PC and Workstations," IEEE & ACM Professional Development Seminar, Tampa, FL, (June 4, 1988).
- [Char86] Charette, Robert N., Software Engineering Environments: Concepts and Technology, McGraw Hill, Inc., New York, NY, (1986).
- [Dema79] DeMarco, Tom, Structured Analysis and Specification, Prentice-Hall, Inc., Englewood Cliffs, NJ, (1979).
- [Dijk65] Dijkstra, Edsger W., "Programming Considered as a Human Activity," in Proceedings of the 1965 International Federation of Information Processing Congress, North Holland Publishing Company, Amsterdam, Netherlands, (1965) pp 213-217.
- [Dijk72] Dijkstra, Edsger W., "The Humble Programmer," Communications of the ACM, 15, 10, (October 1972), pp 859-866.
- [DOD88] Department of Defense. Defense System Software Development, DOD-STD-2167A, Department of Defense, Washington D.C., (1988).

- [Edwa88] Edwards, William W. II, "A Methodology for CASE Tool Selecection," in Proceedings of CASE Studies 1988. Ninth Annual Conference on Applications of Computer Aided Software Engineering Tools, (May 23-27 1988), Section C8801, pp1-35.
- [Gibs88] Gibson, Michael L., "A Guide to Selecting CASE Tools," Datamation, 34, 13, (July 1, 1988), pp 65-66.
- [Glas82] Glass, Robert L., "Recommended: A Minimum Standard Software Toolset," ACM Software Engineering Notes, 7,4, (October 1982), pp 3-13.
- [Hoar81] Hoare, C. A. R., "The Emperor's Old Clothes," Communications of the ACM, 24,2, (February, 1981), pp 75-83.
- [Jack83] Jackson, M.A., System Development, Prentice-Hall, Inc., Englewood Cliffs, NJ, (1983).
- [Knut73] Knuth, Donald E. The Art of Computer Programming. Vol 1/ Fundlemental Algorithms, 2ed, Addison-Wesley Publishing Company, Reading, MA, (1973).
- [Lick85] Licker, Paul S., The Art of Managing Software Development People, John Wiley & Sons, New York, NY, (1985).
- [Merl88] Merlyn, Vaughn P., "CASE - Today and in the Future," in Proceedings of CASE Studies 1988. Ninth Annual Conference on Applications of Computer Aided Software Engineering Tools, (May 23-27, 1988), Section C8820, pp1-12.
- [Naur76] Naur, Peter, Randell, Brian, Buxton, J.N., Software Engineering Concepts and Techniques: Proceedings of the NATO Conferences, Petrocelli/Charter, New York, NY, (1976).
- [Orr77] Orr, K.T., Structured Systems Development, Yourdon Press, New York, NY, (1977).
- [Parn72] Parnas, D. L., "On Criteria to be used in Decomposing Systems into Modules," Communications of the ACM, 15, 12, (December 1972), pp 1053-1058.
- [Pres87] Pressman, Roger S., Software Engineering: A Practitioner's Approach, 2ed, McGraw-Hill Book Company, New York, NY, (1987).
- [Stay76] Stay, J.F., "HIPO and Ingrated Program Design," IBM Systems Journal, 15, 2 (1976), pp 143-154.
- [Voel88] Voelcker, John, "Automating Software: Proceed with Caution," IEEE Spectrum, 25, 7, (July 1988), pp 25-27.
- [Ward86] Ward, Frank, "Keynote Address", Proceedings: Workshop on Future Directions in Computer Architecture and Software, (Dharma P. Agrawal, ed.), 5-7 May 1986, pp 1-15.

- [Warn74] Warnier, Jean D., Logical Construction of Programs, 3ed., Van Nostrand Reinhold Company, New York, NY, (1974).
- [Wass82] Wasserman, Antony, I., "Automated Tools in the Information System Development Environment," in Automated Tools for Information Systems Design, (Hans-Jochen Schneider & Antony I Wasserman, eds.), North Holland Publishing Company, Amsterdam, The Netherlands, (1982), pp 1-9.
- [Whit88] Whitmore, Sam, "Programming Shortcuts are Not Time Savers in Long Run," PC Week, (June 28 1988), pg 32.
- [Wein71] Weinberg, Gerald M., The Psychology of Computer Programming, Van Nostrand Reinhold Company, New York, NY, (1971).
- [Wirt71] Wirth, Niklaus, "Program Development by Stepwise Refinement," Communications of the ACM, 14, 4, (April 1971), pp 221-227.
- [Your79] Yourdon, Edward N., and Constantine, Larry L., Structured Design: Fundamentals of a Discipline of Computer Program and System Design, Prentice-Hall, Inc., Englewood Cliffs, NJ (1979).

APPENDIXES

**APPENDIX A.
VENDOR CONTACTS**

<u>ADDRESS</u>	<u>PRODUCT(S)</u>	<u>CONTACT</u>
AGS Management Systems, Inc. 880 First Ave King of Prussia, PA 19406	MULTI/CAM	Mr. Robin Wheeler (404)-952-9093
American Management Sys, Inc 9800 Centre Parkway, Suite 950 Houston, TX 77036	Life-cycle Productivity System	
Applied Data Research, Inc. Route 206 and Orchard Rd CN-8 Princeton, NJ 08543	Depictor	
Arthur Anderson & Co. 101 East Kennedy Blvd Tampa, FL 33602	Foundation Series	Mr. Robert Ebaugh (813)-222-4600
CADRE Technologies, Inc. 222 Richmond St Providence, RI 02903	Teamwork	Ms. Jackie Harness (401)-351-5950
The CADWARE Group, Ltd 869 Whalley Ave New Haven, CT 06515	SYLVA Series	
The Catalyst Group Peat Marwick Main & Co. 303 East Wacker Dr Chicago, IL 60601	PATHVU et al.	Mr. James Peterson (800)-323-3059
CGI Systems, Inc. 8200 Greensboro Dr, Suite 1010 McLean, VA 22102	PACBASE	Mr. Larry Claussen (703)-448-8181
Computer Sciences Corp 3160 Fairview Park Dr Falls Church, VA 22042	Design Generator	
Cortex Corp 138 Technology Dr Waltham, MA 02154	CorVision	Mr. Bob Howatt (617)-894-7000

Appendix A. (cont'd)

<u>ADDRESS</u>	<u>PRODUCT(S)</u>	<u>CONTACT</u>
ICONIX Software Engineering, Inc 2800 Twenty Eighth St, Suite 320 Santa Clara, CA 90405	PowerTools Series	Mr. Doug Rosenberg (213)-458-0092
IDE 8300 Boone Blvd, Suite 500 Vienna, VA 22180	Software through Pictures	Mr. Doug Whall (703)-848-8808
Index Technology Corp One Main St Cambridge, MA 02142	Excelerator	Mr. William Agee (404)-992-2910
Institute for Information Industry 8th Floor, 106 Ho-Ping E. Rd. Taipei, Taiwan, R.O.C.	KangaTool Series	
KnowledgeWare 3340 Peachtree Rd., NE Suite 1100 Atlanta, GA 30026	KnowledgeWare Workstation Series	Mr. Steve Kahan (800)-338-4130
Language Technology 27 Congress St Salem, MA 01970	RECORDER	Mr. Bob Freedman (800)-732-6337
Learmonth & Burchett ManagementSystems Inc. 2900 North Loop West, Suite 800 Houston, TX 77092	AUTO-MATE PLUS	Mr. Ed Hall (800)-231-7515
Manager Software Products, Inc. 131 Hartwell Ave Lexington, MA 02173	Manager Series	
Matterhorn, Inc. 6207 Bury Dr Eden Prairie, MN 55344	HIBOL	
McDonnell Douglas P.O. Box 516, L861-302-1E St. Louis, MO 63166	ProKit*WORKBENCH	Ms. Bev Deshazer (800)-822-7337
Mentor Graphics Corp 8500 S.W. Creekside Place Beaverton, OR 97005 (formerly Tektronix CASE Division)	Mentor Graphics CASE	
META Systems 315 E. Eisenhower Pkwy, Suite 200 Ann Arbor, MI 48104	PSL/PSA et al.	Ms. Rebecca Sizemore (313)-633-6027

Appendix A. (cont'd)

<u>ADDRESS</u>	<u>PRODUCT(S)</u>	<u>CONTACT</u>
NASTEC Corp 24681 Northwestern Highway Southfield, MI 48075	CASE 2000 DesignAid	Mr. Darrell Trimble (703)-556-9401
OPTIMA, Inc. 1300 Woodfield Rd, Suite 400 Schaumburg, IL 60173	Design Vision, et al.	Mr. Mike LeSage (800)-633-6303
POLYTRON Corp 1700 NW 167th Place Beaverton, OR 97006	Poly Series	
Promod, Inc. 23685 Birtcher Dr Lake Forest, CA 92630	ProMod Series	Mr. Tom Scott (800)-255-2689
RATIONAL 3320 Scott Blvd Santa Clara, CA 95054	RATIONAL Design Facility et al.	
Softlab, Inc. 188 The Embarcadero Bayside Plaza, 7th Floor San Francisco, CA 94105	MAESTRO, et al.	Mr. Dennis Crow (415)-957-9175
StarSys, Inc. 11113 Norlec Dr Silver Spring, MD 20902	MacBubbles	
Texas Instruments, Inc. 6500 Chase Oaks Blvd P.O. Box 869305 Plano, TX 75086	Information Engineering Facility	Mr. Steve Thomas (703)-849-1469
Visible Systems Corp 49 Lexington St Newton, MA 02165	VISIBLE ANALYST Workbench	
Visual Software, Inc. 3945 Freedom Cir, Suite 540 Santa Clara, CA 95054	vsDesigner	
YOURDON, Inc. 1501 Broadway New York, NY 10036	Analyst/Designer Toolkit	

APPENDIX B.
SAMPLE SURVEY QUESTIONNAIRE

Appendix B.



DEPARTMENT OF THE AIR FORCE
DETACHMENT 158, AIR FORCE ROTC (ATC)
UNIVERSITY OF SOUTH FLORIDA, TAMPA, FL 33620-8250

REPLY TO
ATTN OF:

SUBJECT:

19 July 1988

to: Dear Computer Professional,

I am currently sponsored by the Air Force for my graduate studies at the University of South Florida. For my thesis, I am researching the use of CASE tools in the small software group environment. Your CASE vendor supplied your name as someone who might be able to assist me.

I appreciate you completing this questionnaire; it should take less than 20 minutes. A pre-paid postage envelope is enclosed. If you would like a copy of the results, please let me know.

If possible, I would like the project or team leader responsible for the development/maintenance projects to complete the questionnaire. Answers should be provided as they pertain to **your software group** that is using the CASE tool(s) and **not** the entire company.

Please answer the questions as they apply to you with that in mind. If there is more than a single response to any question, please check all that apply. Conversely, if information is unavailable for a question, please leave that question blank.

Additional comments, are of course encouraged. Please include them on the reverse side of this cover letter.

Thank you,

Marc L. Sims

Marc L. Sims, Captain, USAF

MLS/mtf

SECTION I. Personal Information:Position: ☐ Manager ☐ Project Leader ☐ Analyst ☐ OtherMay I contact you for additional information? ☐ Yes ☐ No

If yes, Name _____ Telephone _____

• Would you recommend the use of your current CASE development tool(s) for a software group with *less than seven* people?

☐ Yes ☐ No ☐ It depends _____**SECTION II. Your Software Development Environment Prior to the Installation of Your First CASE Tool(s):**

1a. Approximate number of Data Processing personnel in the entire company:

Management _____ Systems designers/analysts _____

Programmers _____ Database Specialists _____

Q/A Testers _____ Documentation/Tech Writers _____

Others: _____

Note: for the remaining questionnaire, please answer the questions as they pertain to your software group and not the company.

1b. Approximate number of Data Processing personnel in your group:

Management _____ Systems designers/analysts _____

Programmers _____ Database Specialists _____

Q/A Testers _____ Documentation/Tech Writers _____

Others: _____

2. Average experience level of personnel in your group?

☐ Greater than 5 years ☐ 3-5 years ☐ Less than 3 years

3. What type of efforts are supported?

☐ Business ☐ Education ☐ Government ☐ Other _____

4. What environment(s) do you support?

☐ Mainframe ☐ Mini ☐ Micro ☐ Networks

- 2

SECTION III. Environment Since Installation of Your CASE Product(s):

If you have installed more than one CASE product, please complete a separate copy of this section for each tool. We will pay for any additional postage.

Tool Name/Vendor: _____

1. When was the tool installed? _____ Month _____ Year
- 2a. Did you receive training from the vendor? ☐ Yes ☐ No
- 2b. Was it appropriate? ☐ Yes ☐ No _____
3. What type of on-going training is being used?
☐ None ☐ Formal Classes ☐ On-the-job training
4. How many people use this tool?
☐ All ☐ 75% ☐ 50% ☐ 25% ☐ 1 person ☐ None
- 5a. What has the response been from your most experienced personnel?
☐ Favorable ☐ No opinion ☐ Unfavorable
- 5b. What has the response been from your least experienced personnel?
☐ Favorable ☐ No opinion ☐ Unfavorable
6. During which phases of the software development cycle do you use this tool?
Please check each block that applies.
 - ☐ Definition of requirements
 - ☐ Analysis
 - ☐ Design
 - ☐ Coding
 - ☐ Testing
 - ☐ Maintenance
If other, please specify _____
- 7a. Did you modify your existing methods to use this CASE tool?
☐ No it fit right in ☐ Some ☐ Extensive modifications
- 7b. Is this what you anticipated? ☐ Yes ☐ No
8. In your opinion, was the installation of this tool a wise decision?
☐ Yes ☐ No _____

Appendix B. (cont'd)

- 9a. What is the tool's most useful feature(s)? (e.g., graphics, data dictionary, error checking, etc.) _____
- 9b. What feature(s) would you like this tool to have? _____
10. Has the addition of this tool and the reaction of your group made it easier or harder to introduce a similar tool in the future?
☐ Yes ☐ No _____
11. What additional CASE tool(s) would you like to have? _____
12. Based upon your experience, please give your opinions concerning the use of this CASE tool by a software group of *fewer than seven* people.
- ("1" means *strongly agree*, "5" means *strongly disagree*)
- | | 1 | 2 | 3 | 4 | 5 |
|--|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| a. This tool is easy to use. | ☐ | ☐ | ☐ | ☐ | ☐ |
| b. This tool is helpful for the Requirements Definition phase. | ☐ | ☐ | ☐ | ☐ | ☐ |
| c. This tool enhances productivity in a small group. | ☐ | ☐ | ☐ | ☐ | ☐ |
| d. This tool is indispensable for the Analysis phase. | ☐ | ☐ | ☐ | ☐ | ☐ |
| e. This tool is too rigorous in enforcing its methodologies and does not allow programmer flexibility. | ☐ | ☐ | ☐ | ☐ | ☐ |
| f. Personnel are resistant to using this tool and will cause management/personnel problems in a small group. | ☐ | ☐ | ☐ | ☐ | ☐ |
| g. Personnel should be required to use this tool. | ☐ | ☐ | ☐ | ☐ | ☐ |
| h. Using this tool for the Design phase greatly improves the maintainability of the software. | ☐ | ☐ | ☐ | ☐ | ☐ |
| i. Economically, this tool is a good value for a small group. | ☐ | ☐ | ☐ | ☐ | ☐ |
| j. This tool should be used during the Coding phase. | ☐ | ☐ | ☐ | ☐ | ☐ |
| k. This tool provides good configuration management practices that are needed by a small group. | ☐ | ☐ | ☐ | ☐ | ☐ |
| l. This tool should be used during the Testing phase. | ☐ | ☐ | ☐ | ☐ | ☐ |
| m. The design capabilities of this tool are particularly needed by a small group. | ☐ | ☐ | ☐ | ☐ | ☐ |
| n. This tool is well suited for the Maintenance phase. | ☐ | ☐ | ☐ | ☐ | ☐ |
| o. This tool is appropriate for a small software team. | ☐ | ☐ | ☐ | ☐ | ☐ |
| p. Other tool(s) would be more appropriate than this one. | ☐ | ☐ | ☐ | ☐ | ☐ |

APPENDIX C. SURVEY DATA

This appendix contains the data received from the 76 respondents to our survey. The data for section I is in the first five pages and the last five pages has the data for section III. Below are the explanations used for this appendix.

SECTION I.

<u>Number</u>	<u>Explanation</u>
Position	1 = manager; 2 = project manager; 3 = analyst; 4 = other
Recommend	1 = yes; 2 = no; 3 = it depends
	the x means that this response is an extra one from the same organization regarding a different tool

SECTION II.

<u>Number</u>	<u>Explanation</u>
1a. & 1b.	the number of personnel; the L means that the respondent did not know how people they had but it was a very large organization other type of personnel: 1 = data entry clerks; 2 = PC support; 3 = s/w maint; 4 = comm specialist; 5 = operations; 6 = tech support 7 = sys s/w; 8 = consultants; 9 = functional analysts; 10 = proj control 11 = configuration mgt
2.	1 = greater 5 yrs; 2 = 3-5 yrs; 3 = less than 3 yrs
3.	1 = business; 2 = education; 3 = gov't
4.	1 = mainframe; 2 = mini; 3 = micro; 4 = net; 5 = combo
5.	1 = applications; 2 = systems; 3 = database; 4 = telecom; 5 = engineering 6 = real-time; 7 = R & D;
6.	1 = difficult; 2 = ave; 3 = not tech difficult
7.	1 = dev new programs; 2 = maint; 3 = both
8.	1 = 1 or 2; 2 = 3 or 5; 3 = greater than 5
9.	1 = less than 1; 2 = less than 6; 3 = less than 12; 4 = more than 12
10a.	1 = yes; 2 = no
10b.	1 = yes; 2 = usually; 3 = seldom; 4 = no
11a.	1 = yes; 2 = no
11b.	1 = in-house; 2 = Warnier-Orr; 3 = Yourdon; 4 = Method1 (Arthur Anderson); 5 = Stradis; 6 = Maurice
11c.	1 = yes; 2 = ususally; 3 = seldom; 4 = no
12.	1 = mgt; 2 = technical staff; 3 = comb; 4 = QA; 5 = strategic planning; 6 = marketing; 7 = respondent; 8 = internal customers; 9 = customer
13.	1 = highly structured; 2 = informal; 3 = in between

Appendix C. (cont'd)

SECTION III.

<u>Number</u>	<u>Explanation</u>
Tool	1 = Prokit*Workbench; 2 = DesignAid; 3 = Automate Plus; 4 = Excelerator; 5 = Recoder; 6 = Teamwork; 7 = ProMod; 8 = Software through Pictures; 9 = Power Tools; 20 = LINC (UNISYS) 21 = Telon (Pansophic); 22 = ADADL; 23 = Apollo (DSEE); 24 = Brackets (Orr); 25 = Oracle; 26 = PACBASE (CGI); 27 = SEQUENT UNIX Prog workbench; 28 = Knowledgeware; 29 = DEC; 30 = Architect; 31 = Yourdon S/W Workbench
1.	year installed
2a.	1 = yes; 2 = no
2b.	1 = yes; 2 = no
3.	1 = none; 2 = formal classes; 3 = OJT
4.	1 = all; 2 = 75%; 3 = 50%; 4 = 25%; 5 = 1 person; 6 = none
5a.	1 = favorable; 2 = no opinion; 3 = unfavorable
5b.	1 = favorable; 2 = no opinion; 3 = unfavorable
6.	1 indicates that the tool is used in these phases
6.	other uses are 1 = data admin - data planning; 2 = documentation; 3 = rapid prototyping of DB functions; 4 = SCM
7a.	1 = fit right in; 2 = some; 3 = extensive
7b.	1 = yes; 2 = no
8.	1 = yes; 2 = no
9a/b.	These capabilities and features are listed below. 1 = data dictionary 2 = reusability of data dictionary objects 3 = balancing capabilities 4 = data normalization aid 5 = graphics 6 = forcing formalisms 7 = reverse engineering 8 = automatic documentation generation 9 = code generator 10 = prototyping 11 = good user interface 12 = structured methodology 13 = petrinets for real-time process 14 = ADA code generator 15 = windowing 16 = development envir for windows 17 = reports 18 = project mgt 19 = interface with DBMS system 20 = interface with SQL/DS schema creation 21 = Networks 22 = data modeling 23 = program specifications 24 = test data generator 25 = better word processing 26 = better interface with developed system

Appendix C. (cont'd)

- 27 = error checking
- 28 = simplified logic
- 29 = automatic generate system flow diagrams
- 31 = speed
- 32 = estimator
- 33 = scheduler
- 34 = project tracking
- 35 = more rigorous data modeling
- 36 = E/R tool
- 37 = regeneration test facility
- 38 = ease of maintaining existing systems
- 39 = less custom code requirements
- 40 = enterprize modeling
- 41 = diagraming capabilities
- 42 = overall general improvements
- 43 = screen painting
- 44 = consistancy checking
- 45 = better naming
- 46 = better interfaces
- 47 = database design
- 48 = generate 4GL code to natural language
- 49 = generate 4GL
- 50 = seamless integration with other tools
- 51 = strategic planning
- 52 = ADA support
- 53 = requirements tracking
- 54 = more systems engineering support
- 55 = configuration management
- 56 = context sensitive editor
- 57 = PDL tool
- 58 = better physical security
- 59 = metrics
- 60 = measurement
- 61 = disk roll out when memory is full
- 62 = structure charts
- 63 = generate structured cobol
- 64 = flexibility
- 65 = additional integration with coding
- 66 = structured testing
- 67 = more front-end design
- 68 = graphic outline processor
- 69 = documentation templates
- 70 = real-time aids
- 71 = additional dictionary formats
- 72 = much more powerful dictionary
- 73 = requirements analysis

- 10. 1 = yes; 2 = no; since this question was poorly worded, this item was left blank unless the respondent clearly indicated their answer
- 11. same as for question 9
- 12. self explanatory

Appendix C. (cont'd)

RESPONDENTS

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

SECTION I.

Position	1	1	4	4	1	x	2	1	1	1	3	2	x	1	3
Recommend use	1	1	3	1	1	x		2	1	1	1	3	x	1	1

SECTION II.

1a. Company

Management				2	20	20	30		20	L	5	4	4	100	3
Systems analysts				2	25	25	15		17	L	7	10	10	100	4
Programmers				2	100	100	50		50	L	24	5	5	500	2
Database specialists				2	15	15	5		9	L	2	1	1	20	0
Q/A testers				0	25	25	25		0	L	1	0	0	50	0
Doc/tech writers				0	25	25	5		5	L	0	0	0	10	0
Others				0	0	0	0		0		3	7	7	0	1
other type											2	5	5		4

1b. Group

Management	1	15	3	0	3	3	0	8	2	4	0	0	0	1	3
Systems analysts	0	30	2	2	3	3	1	15	1	4	7	3	3	2	4
Programmers	0	30	18	0	13	13	2	20	4	0	0	0	0	5	2
Database specialists	2	0	1	0	2	2	0	3	0	1	0	0	0	3	0
Q/A testers	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
Doc/tech writers	0	4	0	0	1	1	1	2	0	0	0	0	0	1	0
Others	0	0	2	0	0	0	0	3	0	0	0	0	0	0	1
other type			1					3							

2. Experience level	1	1	2	1	2	2	2	1	1	1	1	1	1	1	1
3. Type effort	1	1	1	1	3	3	3	5	2	1	1	1	1	5	3
4. Environment	1	5	5	5	5	5	5	5	1	1	5	1	1	5	5
5. Application 1	3	1	1	1	1	1	1	1	3	1	1	1	1	1	1
5. Application 2	10		3	7	3	3	6	3	7	3		5	5	3	6
5. Application 3								4						4	
6. Software difficulty	2	2	2	2	2	2	2	1	2	2	2	2	2	2	2
7. Responsible for				1	3	3	1	3	1	1	1	1	1	3	3
8. Persons per project	1	2	1	1	2	2	2	2	1	3	3	1	1	3	1
9. Person months	2	2	2	4	4	4	4	3	2	4	4	3	3	3	2
10a. Standards in place	1	2	1	1	1	1	2	1		2	1	1	1	1	2
10b. Observed	3		3	2	2	2		2			3	1	1	1	
11a. Method in place	2	2	2	1	2	2	2	2		2	2	1	1	1	2
11b. Which method				2								1	1	1	
11c. Observed				2						2		1	1	1	
12. Who decided	3	4	1	1	2	2	2	2	1	1	5	1	1	2	1
13. Work environment	2	2	2	3	3	3	2	3	2	2	2	3	3	1	2

Appendix C. (cont'd)

RESPONDENTS

16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

SECTION I.

Position	1	2	2	x	1	1	1	1	2	1	3	2	1	1	
Recommend use	2	1	1	x	1	1	1	1	1	1	1	1	1	1	3

SECTION II.

1a. Company

Management	2	30	100	100	50	L	14	4	5	10	100	3	100	L	100
Systems analysts	3	30	300	300	50	L	25	4	12	30	1000	10	250	L	700
Programmers	8	90	400	400	200	L	45	6	20	20	1000	25	1000	L	800
Database specialists	0	0	30	30	40	L	6	0	3	1	30	5	50	L	50
Q/A testers	0	0	6	6	25	L	0	0	0	0	10	0	20	L	30
Doc/tech writers	1	0	1	1	10	L	0	0	2	5	10	0	50	L	70
Others	4	0	100	100	0		0	0	35	10	1000	0	0		0
other type									5	6					

1b. Group

Management	1	10	0	0	4	1	4	1	0	5	5	0	10	6	2
Systems analysts	4	15	4	4	10	3	15	2	2	30	20	2	10	31	6
Programmers	0	80	2	2	0	13	25	2	4	20	10	3	30	0	6
Database specialists	0	0	0	0	15	0	6	0	0	1	2	0	10	6	3
Q/A testers	0	0	0	0	0	4	0	0	0	0	0	0	3	6	1
Doc/tech writers	0	0	0	0	2	0	0	0	0	0	0	0	5	1	1
Others	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
other type															

2. Experience level	1	2	1	1	1	1	1	2	2	2	2	1	2	1	1
3. Type effort	1	1	1	1	1	3	1	4	2	1	1	1	1	3	5
4. Environment	1	1	1	1	5	5	1	1	1	1	5	5	1	1	5
5. Application 1	1	1	1	1	1	1	1	5	1	1	1	1	1	1	1
5. Application 2			3	3	3	6	3			3	3		3	3	
5. Application 3					4		4			6	4		5	11	
6. Software difficulty	2	2	2	2	2	1	2	2	2	1	2	2	1	1	1
7. Responsible for	3	3	1	1	1	1	1	1	3	3	3	1	3	3	3
8. Persons per project	1	3	2	2	3	3	2	2	2	2	2	2	3	2	3
9. Persons months		3	4	4	3	4	4	3	2	3	4	3	4	4	4
10a. Standards in place	1	1	1	1	1	1	1	2	1	2	1	2	1	2	1
10b. Observed	3	3	3	3	2	2	2		3	1	2		3		2
11a. Method in place	2	2	1	1	1	1	2	2	2	2	2	2	1	1	1
11b. Which method			1	1	3	1							1		1
11c. Observed			3	3	2	2							3		2
12. Who decided	2	3	3	3	1	3	3	2	3	3	3	2	3	2	3
13. Work environment	2	3	3	3	1	3	3	2	3	3	3	2	3	2	3

Appendix C. (cont'd)

RESPONDENTS

31 32 33 34 35 36 37 38 39 40 41 42 43 44 45

SECTION I.

Position	1	2	4	1	1	2	1	3	2	3	2	1	3	1	1
Recommend use	3	1	1	1		1	1	1	1	1	1	1	1	1	1

SECTION II.

1a. Company

Management				2	1000	4	70	L			10	100	7	20
Systems analysts				4	3000	30	50	L			5	70	11	60
Programmers				12	3000	60	250	L			30	200	10	140
Database specialists				1	500	20	15	L			0	10	0	0
Q/A testers				2	300	0	20	L			1	5	0	8
Others				0	0	0	0				0	400	8	150
other type												5	5	5

1b. Group

Management	1	11	1	2	2	5	1	1	6	1	1	1	1	2	3
Systems analysts	3	30	1	4	7	50	5	6	0	3	4	1	2	6	4
Programmers	4	5	1	12	0	100	20	0	140	5	6	3	0	3	8
Database specialists	0	3	0	1	0	5	0	2	0	2	1	0	0	0	2
Q/A testers	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
Doc/tech writers	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
Others	0	6	0	0	0	0	0	0	0	10	0	0	0	0	0
other type		7								8					

2. Experience level	1	1	1	1	2	1	2	1	1	1	1	1	1	2	2
3. Type effort	1	3	6	3	5	5	1	1	5	5	3	3	1	3	1
4. Environment	5	5	2	2	5	5	3	5	4	5	5	2	1	1	1
5. Application 1	5	1	6	2	1	1	5	1	1	1	1	6	1	1	1
5. Application 2	6	5		5	5	2	6	2	3	2	2				
5. Application 3	7	6		6	6	4		3	6	3	3				
6. Software difficulty	1	2	2	1	1	1	1	1	2	1	1	1	1	2	2
7. Responsible for	1	3	1	1	1	3	1	1	3	1	3	1		3	3
8. Persons per project	3	1	2	3	1	3	1	3		2	3	1	2	1	2
9. Person months	4	3	4	4	4	4	3	4	4	3	4	4	2	1	3
10a. Standards in place	2	1	1	1	2	1	2	2	2	1	2	2	1	1	1
10b. Observed		2	2	1		2				2			2	2	2
11a. Method in place	2	2	1	1	2	2	2	1	2	2	1	2	2	2	2
11b. Which method			3	1		3		3			1		4		
11c. Observed			3	1		2		3			2		2		
12. Who decided	3	3	2	1	2	3	2	2	2	2	3	2	3	2	3
13. Work environment	3	3	2	1	2	3	2	2	2	2	3	2	3	2	3

Appendix C. (cont'd)

RESPONDENTS

	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
SECTION I.															
Position	1	2	2	2	x	1	x	1	1	2	1	1	2	x	x
Recommend use	1		1	2	x	1	x	1	1	1		1	1	x	x
SECTION II.															
1a. Company															
Management	L	20	30	300	300	L	L	10	40	1	4	200	120	120	120
Systems analysts	L	75	200	500	500	L	L	12	20	1	0	200	60	60	60
Programmers	L	150	200	500	500	L	L	56	100	5	20	400	1000	1000	1000
Database specialists	L	10	7	50	50	L	L	3	5	0	2	150	35	35	35
Q/A testers	L		30	100	100	L	L	0	5	0	0	0	20	20	20
Doc/tech writers	L	5	70	200	200	L	L	1	0	0	7	50	18	18	18
Others		0	0	0	0			0	0	0	0	200	30	30	30
other type												9	10	10	10
1b. Group															
Management	1	1	6	2	2	1.5	6	1	1	1	1	4	10	10	10
Systems analysts	7	8	12	10	10	0	0	7	6	1	5	10	8	8	8
Programmers	0	8	7	10	10	2.5	140	8	10	2	7	0	50	50	50
Database specialists	0	0	2	2	2	0	0	0	1	0	0	5	2	2	2
Q/A testers	0	0	3	0	0	0	0	0	0	0	0	0	3	3	3
Doc/tech writers	0	0	2	0	0	0	0	1	0	0	0	0	0	0	0
Others	0	0	0	0	0	0	0	0	0	0	0	20	3	3	3
other type												9	10	10	10
2. Experience level	1	2	1	1	1	2	1	2	2	3	2	2	2	2	2
3. Type effort	3	1	3	3	3	1	3	1	3	3	1	3	3	3	3
4. Environment	5	4	5	5	5	5	4	1	5	1	5	5	5	5	5
5. Application 1	1	3	2	1	1	6	1	1	1	2	2	1	1	1	1
5. Application 2	5	4	3	2	2		3	5	3	2	5	2	3	3	3
5. Application 3	6	6		3	3		6			3	6	3	4	4	4
6. Software difficulty	1	2	1	1	1	1	2	2	2	2	2	1	2	2	2
7. Responsible for	1	1	1	3	3	3	3	1	3	3	3	1	1	1	1
8. Persons per project	3	3	3	3	3	2	2	2	2	2	1	3	2	2	2
9. Person months	4	4	4	4	4	4	4	2	3	3	2	4	4	4	4
10a. Standards in place	1	1	2	2	2	2	2	1	2	1	2	1	1	1	1
10b. Observed	1	2					2	2		2		4	2	2	2
11a. Method in place	2	2	2	2	2	2	2	1	2	1	2	1	1	1	1
11b. Which method								2		1		1	1	1	1
11c. Observed								2		2		3	2	2	2
12. Who decided	3	3	2	2	2	2	2	3	2	2	2	2	1	1	1
13. Work environment	3	3	2	2	2	2	2	3	2	2	2	2	1	1	1

Appendix C. (cont'd)

RESPONDENTS

	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76
SECTION I.																
Position	1	2	1	1	4	2	1	1	1	1	1	2	x	1	1	1
Recommend use	1	3	1	1	1	1	1	1	1	1	1	1	x	1	1	1
SECTION II.																
1a. Company																
Management	25	100	2	4	4		1	3	1000	12	4	400	400	20	300	
Systems analysts	100	75	4	11	25	300	2	8	1500	75	0	300	300	20		
Programmers	25	500	13	35	15	200	2	10	3500	75	35	1500	1500	100	2500	
Database specialists	0	100	1	2	5	50	0	2	500	8	0	20	20	9		
Q/A testers	0	50	2	0	1	30	0	6	2000	12	0	200	200	20		
Doc/tech writers	0	75	2	0	5	100	0	2	1000	12	0	40	40	2		
Others	0	0	0	0	3	0	0	0	0	12	0	0	0	0		
other type					11					12						
1b. Group																
Management	7	7	1	1	2	2	1	3	20	4	1	3	3	3	1	1
Systems analysts	18	10	4	0	4	2	2	8	20	25	0	2	2	2	1	0
Programmers	18	5	11	9	2	9	2	10	40	0	12	20	20	15	13	2
Database specialists	0	1	1	1	0	2	0	2	10	4	0	0	0	1	1	0
Q/A testers	0	2	1	0	0	4	0	6	5	0	1	0	0	4	0	0
Doc/tech writers	0	2	2	0	2	1	0	2	5	1	0	0	0	0.5	1	0
Others	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
other type																
2. Experience level	2	2	2	1	1	2	1	1	1	2	2	2	2	1	1	3
3. Type effort	7	3	8	5	3	8	1	3	1	3	3	1	1	1	6	4
4. Environment	5	5	5	5	5	5	2	5	5	5	5	1	1	5	1	3
5. Application 1	1	5	1	4	1	1	1	1	1	3	6	1	1	1	5	5
5. Application 2	2	6	2	6	6	2		3	2	5		3	3			
5. Application 3	3		7			3			3	6						
6. Software difficulty	2	1	1	1	2	1	2	2	2	2	1	2	2	2	1	1
7. Responsible for	3	1	3	1	3	1	3	1	1	3	1	3	3	3	3	1
8. Persons per proj	2	3	2	3	3	3	1	3	3	2	2	2	2	3	2	2
9. Person months	4	4	4	4	4	4	2	4	3	3	4	2	2	3	4	4
10a. Standards	1	2		1	1	2	1	2	2	1	1	1	1	1	1	2
10b. Observed	1	4		2	2		1	4		2	3	3	3	2	2	
11a. Method in place	2	1	1	1	2	2	1	1	2	1	1	1	1	1	1	2
11b. Which method		1	3	3			3	5		3	1	2	2	6	3	
11c. Observed		2	2	2			2	1		2	2	3	3	1	2	
12. Who decided	2	2	1	3	2	3	3	3	3	3	3	2	2	3	3	2
13. Work environ	2	2	1	3	2	3	3	3	3	3	3	2	2	3	3	2

Appendix C. (cont'd)

	RESPONDENTS														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SECTION III.															
Tool	1	1	1	1	1	4	1	1	1	1	1	2	20	2	2
1. Date installed	87	88	88	88	88	87	88	88	86	87	87	87	86	85	88
2a. Training received	1	2	2	2	1	1	2	1	2	1	2	1	1	2	1
2b. Appropriate	1				1	1		1		1		1	1	2	1
3. On-going training	3	3	3	3	4	4	1	4	3	2	4	3	4	3	1
4. People use it	5	5	5	4	2	3	2	2	4	3	5	1	1	4	3
5a. Exp response	1	1	1	1	1	1	1	1	1	1		3	1	1	1
5b. Junior response		2	2	1	1	1	1	1	2	3		2	1	2	2
6. Definition req	1	1		1	1	1	1	1	1	1	1	1		1	1
6. Analysis		1	1	1	1			1	1	1	1	1	1	1	1
6. Design				1	1			1	1		1	1	1	1	1
6. Coding					1								1		
6. Testing					1								1		
6. Maintenance					1			1	1				1		
6. Other use	1														
7a. Modify	2	2	2	2	2	2	1	2	1	2	3	1	2	1	2
7b. Anticipated	1	1		1	1	1		1		1	1	1	1	1	1
8. Wise decision	1	1	1	1	1	2	1	1	1	1		2	1	1	1
9a. Useful cap #1	1	1	6	5	11	12	10	5		5	10	5	27	5	1
9a. Useful cap #2	2	5	5	10	1	11		10		1	1		28		
9a. Useful cap #3	3		1	1				17		3			4		
9b. Desired cap #1	4		7	9			13	18	20	4	9	5	8	1	
9b. Desired cap #2			8							9	23	25	29		
9c. Desired cap #3			9							7	24	26			
10. Easier	1			1	1	2	1	1			2				1
11. Addl tool #1				9	9	9	14		7						
11. Addl tool #2							15		19						
11. Addl tool #3							16								
12a. Easy to use	2	4	4	3	2	5	3	2	1	1	2	3	2	2	2
12b. Help RD	1	3	4	2	1	2	2	2	1	1	1	5	5	1	2
12c. Enhance prod	1	3	3	2	1	5	2	4	1	1	1	5	1	3	3
12d. Analysis	1	2	4	3	2	5	3	2	1	1	1	3	2	2	3
12e. Too rigorous	3	3	2	4	5	3	4	3	5	5	4	3	5	4	5
12f. Resistant to it	3	3	3	4	5	5	4	3	5	5	2	5	5	4	5
12g. Required use	1	3	1	2	2	5	2	3	1	1	1	5	3	3	2
12h. Design phase	1	3	2	3	1	2	2	2	1	1	1	5	1	2	1
12i. Economic value	1	2	1	3	1	4	2	5	1	1	1	5	1	2	2
12j. Coding phase		2	4	5	3	4	4	2	5	5	5	5	1	2	3
12k. Config mgt	1	3	2		2	3	4	2	1		3	5	1	3	1
12l. Testing phase	5	3	4	5	3	3	3	2	1	3	5	5	2	2	4
12m. Design cap	2	3	2	3	2	3	3	5	1	3	2	5	1	2	2
12n. Maint phase	4	2	5	4	2	2	4	2	1	1	5	5	1	3	1
12o. Approp for small	2	2	1	2	2	2	2	5	1	1	2	5	1	2	1
12p. Others better	4	3	1		4	1	2		5	5	3	3	5	3	

Appendix C. (cont'd)

	RESPONDENTS														
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
SECTION III.															
Tool	2	2	2	21	2	2	3	3	3	3	3	3	3	3	2
1. Date installed	87	88	86	87	84	87	87	88	85	88	87	88	85		
2a. Training received	1	1	2	1	1	1	1	1	1	2	1	1	1	1	2
2b. Appropriate	2	1		1	1	1	1	1	1		1	1	1	1	1
3. On-going training	3	4	3	2	4	4	2	3	2	1	4	3	2	4	1
4. People use it	1	4	3	3	1	4	4	2	3	5	3	4	4	2	6
5a. Exp response	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3
5b. Junior response	2	1	1	1	1	1	1	1	2	2	1	1	1	1	2
6. Definition req	1	1	1		1	1	1	1	1		1	1	1	1	1
6. Analysis		1	1		1	1	1	1	1	1	1	1	1	1	1
6. Design		1	1		1	1	1	1	1		1	1	1	1	
6. Coding				1		1			1						
6. Testing				1		1									
6. Maintenance				1		1								1	
6. Other use						2						1			
7a. Modify	1	3	2	1	1	2	2	3	2	2	2	3	2	2	1
7b. Anticipated	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1
8. Wise decision	2	1	1	1	1	1	1	1	1		1	1	1	1	2
9a. Useful cap #1		5	5	37	1	1	4	5	41	5	5	47	50	6	1
9a. Useful cap #2		3		38	5	5	6				44		8		27
9a. Useful cap #3						27	1								
9b. Desired cap #1		31	35	39	22	25	10		42	43	46	48	9	11	19
9b. Desired cap #2										10	45				
9c. Desired cap #3															
10. Easier			1	1	1	1	1		1	2	1	1	1		
11. Addl tool #1	30	18	36	7	40		9			9		49	51	11	
11. Addl tool #2		24					32								
11. Addl tool #3															
12a. Easy to use	4	3	1	2	2	1	2	3	2	2	3	2	2	1	2
12b. Help RD	2	1	1	3	3	1	2	2	1	3	1	3	1	1	1
12c. Enhance prod	3	2	1	1	3	1	2	1	2	2	1	2	2	1	1
12d. Analysis	2	2	1	5	2	1	1	1	2	4	1	1	1	1	3
12e. Too rigorous	2	5	5	5	5	5	4	5	2	3	4		5	5	2
12f. Resistant to it	3	3	5	5	5	5	4	5	3	5	4	4	3	5	3
12g. Required use	2	2	5	1	1	1	1	3		4	1	3	1	1	1
12h. Design phase	1	2	3	1	3	1	2	1	2	2	1	3	1	1	2
12i. Economic value	3	2	1	4	1	1	2	4	3	1	2	2	2	1	3
12j. Coding phase	5	3	5	1	4	1	5	4	4	5	3	5	3		4
12k. Config mgt	2		3	2	2		2	1	3	3	4	3	2	1	2
12l. Testing phase	3	5	5	1	2	1	5	4	5	4	4	5	2	1	4
12m. Design cap	2	1	1	4	2	3	2	1	2	2	1	2	1	1	4
12n. Maint phase	3	1	2	4	4	3	2	3	3	4	5	5	2	1	3
12o. Approp for small	3	2	1	5	1	1	2	1	2	1	1	1	2	1	3
12p. Others better	3	4	5	5	3	3	5	5	2	3	5	5	5	5	1

Appendix C. (cont'd)

RESPONDENTS

	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
SECTION III.															
Tool	4	4	6	6	6	6	6	6	6	6	6	6	5	5	5
1. Date installed	87	87		86	87	87	86	88	87	87	88	88		87	87
2a. Training received	2	1	2	2	2	1	2	1	1	2	2	2	1	1	1
2b. Appropriate		1			2		1	1	1				1	1	1
3. On-going training	3	3	1	4	3	3	3	1	3	3	2	3	3	3	4
4. People use it	3	2	5	3	4	4	4	4	3	4	4	4	4	4	5
5a. Exp response	3	1	2	1	1	1	1	1	1	1	1	1	1	1	3
5b. Junior response	3	1	2	1	2	1	3	2	1	2	1	3	1	1	1
6. Definition req	1	1	1	1			1	1	1			1			
6. Analysis		1	1	1	1	1	1	1	1	1	1	1			
6. Design		1		1	1	1	1	1	1	1	1	1			
6. Coding															
6. Testing															
6. Maintenance									1				1	1	1
6. Other use															
7a. Modify	2	2	3	2	2	2	2	2	3	2	2	2	1	1	2
7b. Anticipated	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
8. Wise decision		1	1	1	1	1	1	1	1	1	1	1	1	1	1
9a. Useful cap #1	1	1	27	23	27	27	5	5	5	5	27	5	7	62	63
9a. Useful cap #2	5	5			5	5	1	15		27	1	1			8
9a. Useful cap #3							27	21			50				
9b. Desired cap #1				52		53	52	58	53	9		61			7
9b. Desired cap #2				54						7					
9c. Desired cap #3															
10. Easier	2	1				1	2	1	1	1	1	1	1		1
11. Addl tool #1		9		52		55	56	1	24	59	9	50			
11. Addl tool #2				18		18	57			60	57				
11. Addl tool #3						24									
12a. Easy to use	3	3	2	1	1	1	1	1	1	2	2	3	2	1	1
12b. Help RD	2	1	2	2	3	2	2	1	1	3	5	3	5		5
12c. Enhance prod	3	2	2	2	3	2	3	2	1	2	2	3	2	2	1
12d. Analysis		3	3	1	3	3	2	1	1	2	3	1	3	3	5
12e. Too rigorous		4	4	2	3	5	3	5	5	4	3	3	4	5	3
12f. Resistant to it	3	4	5	4	4	5	2	3	5	4	5	5	5	5	2
12g. Required use	3	3	2	2	5	2	3	3	1	4	2	3	1	2	1
12h. Design phase		2	1	2	5	2	1	2	2	2	2	2	4	3	4
12i. Economic value	4	3	3	2	3	2	3	3	2	3	3	3	2	2	5
12j. Coding phase		3	2	5	5	5	2	2	4	4	5	1	3	2	1
12k. Config mgt		3	3	3		3	2	4	4	2	4	5	3	2	3
12l. Testing phase		4	2	4	5	3	2	3	3	4	5	1	1	1	2
12m. Design cap		2	2	2	5	2	3	1	3	3	2	2	3	3	4
12n. Maint phase	2	2	2	4	5	4	2	1	2	2	2	3	1	1	1
12o. Approp for small	3	2	2	3	3	1	2	1	2	2	2	2	1	2	4
12p. Others better	2	4	3	4	5	5	3	5	5	4	4	3	3	5	3

Appendix C. (cont'd)

	RESPONDENTS															
	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	
SECTION III.																
Tool	7	7	7	7	22	8	23	24	25	26	27	28	2	4	9	
1. Date installed	87	87	85	87	87	88	87	86	88	86	87	88	86	86	88	
2a. Training received	1	1	1	1	1	2	1	2	1	1	2	1	1	1	2	
2b. Appropriate	1	1	1	1	1		1		2	1		1	1	1	1	
3. On-going training	1	1	3	3	3	4	4	4	3	4	3	3	3	3	3	
4. People use it	1	1	3	5	2	1	1	4	3	2	1	1	4	4	3	
5a. Exp response	3	1	1	3	3	1	1	1	1	1	1	1	3	1	1	
5b. Junior response	2	2	3	3	3	2	1	1	2	1	1	1	2	2	3	
6. Definition req	1			1	1	1	1			1		1		1	1	
6. Analysis	1	1	1	1	1	1	1			1		1	1	1	1	
6. Design		1	1	1	1	1	1	1	1	1		1	1	1	1	
6. Coding			1			1	1	1	1	1	1					
6. Testing							1	1		1	1					
6. Maintenance							1	1		1	1					
6. Other use										2						
7a. Modify	1	1	3	2	2	3	1	1	1	2	3	2	1	2	2	
7b. Anticipated	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
8. Wise decision	1	2	1	2	1	1	1	1	1	1	1	1	1	1	1	
9a. Useful cap #1	5	5	6	5	59	1		5	17	9	55	5	5	5	1	
9a. Useful cap #2	1	27				5		9	43			1	3	1	5	
9a. Useful cap #3	23	1				41						36				
9b. Desired cap #1	53	25		11	64	65	59	67	44				42	50	64	
9b. Desired cap #2																
9c. Desired cap #3																
10. Easier	2	1		2	2			1	1		1		1	1	1	
11. Addl tool #1	52					66			18			9			68	
11. Addl tool #2																
11. Addl tool #3																
12a. Easy to use	3	1	2	5	1	2	3	2	1	3	3	1	4	2	2	
12b. Help RD	1	2	2	3	5	2	1	5	2	1	4	1	2	2	2	
12c. Enhance prod	2	3	1	5	1	3	2	2	1	1	2	1	2	2	1	
12d. Analysis	2	3	1	5	5	2	3	4	3	3	4	1	3	3	2	
12e. Too rigorous	4	3	5	1	4	4		5	3	4	4	4	2	2	3	
12f. Resistant to it	3	4	4	1	5	3	3	4	4	2	5	5	3	4	4	
12g. Required use	1	3	1	4	2	3	4	2	4	1	1	1	3	4	2	
12h. Design phase	3	5	3	5	2	3	1	1	1	1	3	1	3	3		
12i. Economic value	4	5	2	5	1	4	3	1	1	1	1	1	2	3	2	
12j. Coding phase	5	5	1	5	3	3	1	1	1	1	1	3	4			
12k. Config mgt	3	5	4	5	5	3	1	3	4	1	1	2	4	4	4	
12l. Testing phase	5	5	3	5	5	3	1	2	4	1	1	5				
12m. Design cap	2	5	1	5	3	2		3	1	1	1	2	2	2	3	
12n. Maint phase	3	5	2	5	5	3	1	2	1	1	1	4				
12o. Approp for small	2	3	2	4	3	2	1	1	1	1	1	1	3	2	2	
12p. Others better	3	3	3	1	1	3			1	5	4	4	2	2	3	

Appendix C. (cont'd)

RESPONDENTS

61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76

SECTION III.

Tool	9	9	9	9	29	7	9	1	30	2	2	26	24	26	9	31
1. Date installed	88	88	87	88		88	87	86	87	88	87	88	87			87
2a. Training received	2	1	2	2	1	2	2	1	2	1	1	1	1	1	2	2
2b. Appropriate	1	1			1			1	1	1		1	1	1	1	
3. On-going training	4	4	4	3	3	3	3	3	3	3	3	2	3	4	1	3
4. People use it	1	2	2	1	3	1	3	4	3	4	4	2	4	4	4	5
5a. Exp response			1	1	1	3	1	1	1	1	1	1	1	1	3	1
5b. Junior response	1	1	3	1	1	3	1	2	1	1	2	3	2	1	1	2
6. Definition req	1	1		1			1	1		1			1	1		1
6. Analysis			1	1	1	1	1	1	1	1	1	1	1	1	1	1
6. Design			1	1	1	1		1	1	1	1	1	1	1	1	1
6. Coding					1				1			1		1		1
6. Testing					1				1			1		1		
6. Maintenance			1							1		1		1		
6. Other use					4											
7a. Modify	2	2	2	2	2	2	2	3	2	1	2	2	1	3	1	3
7b. Anticipated	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1
8. Wise decision	1	1	1	1	1	1	1	1	1	1		1	1	1		1
9a. Useful cap #1	5	5	44	1	1	57	5	1	41	5	1	64	6	1	5	1
9a. Useful cap #2	12	1		3	55		27		4	1	5	8	5	67	1	17
9a. Useful cap #3		44		41	53				5	3					17	44
9b. Desired cap #1	69	70	41	21	5	11	72	9				73	9	7	62	
9b. Desired cap #2		71	25	8					9	41						
9c. Desired cap #3																
10. Easier	1	1	1	1	1		1		1	1	1	1	1	1	1	1
11. Addl tool #1	55		50	53			36								7	
11. Addl tool #2															55	
11. Addl tool #3															6	
12a. Easy to use	1	2	2	1	2	4	2	3	1	1	4	4	3	3	1	2
12b. Help RD	3	4	5	1	3	2	1	2	2	1	3	4	1	1	4	2
12c. Enhance prod	2	2	1	2	3	2	1	3	1	1	2	2	1	1	2	2
12d. Analysis	2	2	1	2	3	2	2	2	2	1	3	4	1	1	2	3
12e. Too rigorous	5	3	5	5	3	3	4	3	4	5	5	5	5	5	4	4
12f. Resistant to it	5	4	5	5	2	4	5	4	4	4	4	3	4	3	2	4
12g. Required use	1	3	1	4	1	3	2	2	2	2	3	2	1	1	2	2
12h. Design phase	1	1	2	2	2	2	3	2	1	1	3	2	1	1	3	1
12i. Economic value	2	4	1	1	2	3	1	3	1	1	3	4	2	1	1	1
12j. Coding phase	5	3	4	5	1	5	3	3	3	3	4	1	4	1	3	1
12k. Config mgt	3	2	2	3	1	5	3	2	2	2	5	1	2	3	5	1
12l. Testing phase	5	2	3	5	1	5	3	3	3	3	5	1	4	3	5	3
12m. Design cap	1	1	1	2	2	2	3	2	1	1	2	4	1	1	2	2
12n. Maint phase	2	2	3	3	3	2	3	2	4	1	3	1	2	1	5	3
12o. Approp for small	2	2	1	1	2	2	1	3	2	1	2	3	1	1	1	1
12p. Others better	3	3	4	4	2	2	3	3	3	4	2	2	4	5	4	3