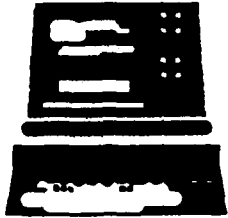


4



USAISEC

*US Army Information Systems Engineering Command
Fort Huachuca, AZ 85613-5300*

U.S. ARMY INSTITUTE FOR RESEARCH
IN MANAGEMENT INFORMATION,
COMMUNICATIONS, AND COMPUTER SCIENCES
(AIRMICS)

AD-A216 909

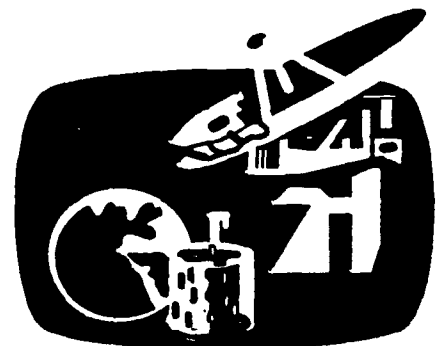
POTENTIAL APPLICATIONS OF ARTIFICIAL INTELLIGENCE TO THE FIELD OF SOFTWARE ENGINEERING

ASQBG-I-89-003

October, 1988

DTIC
ELECTE
JAN 18 1990
S B D

AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0800



DISTRIBUTION STATEMENT A

Approved for public release
Distribution Unlimited

90 01 17 0 29

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704--188
Exp. Date: Jun 30, 1986

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS NONE		
2a. SECURITY CLASSIFICATION AUTHORITY N/A			3. DISTRIBUTION / AVAILABILITY OF REPORT N/A		
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE N/A					
4. PERFORMING ORGANIZATION REPORT NUMBER(S)			5. MONITORING ORGANIZATION REPORT NUMBER(S) N/A		
6a. NAME OF PERFORMING ORGANIZATION Intelligent Systems Group, Data Systems Research Oakridge National Labs		6b. OFFICE SYMBOL (if applicable)	7a. NAME OF MONITORING ORGANIZATION N/A		
6c. ADDRESS (City, State, and ZIP Code) Oakridge, TN 37831-6100			7b. ADDRESS (City, State, and Zip Code) N/A		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION U.S. Army Institute for Research in Mgmt. Information, Communications & Computer Sci.		8b. OFFICE SYMBOL (if applicable) ASBG-I	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER Interagency Agreement DOE No. 1662-1662-A1		
8c. ADDRESS (City, State, and ZIP Code) 115 O'Keefe Bldg., Georgia Institute of Technology Atlanta, GA 30332-0800			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
			DY10-02	05	01
11. TITLE (Include Security Classification) Potential Applications of Artificial Intelligence to the Field of Software Engineering (UNCLASSIFIED)					
12. PERSONAL AUTHOR(S) Emrich, M.; Agarwal, A.; Jairam, B; Murthy, N.					
13a. TYPE OF REPORT FINAL		13b. TIME COVERED FROM 87/02 TO 87/12		14. DATE OF REPORT (Year, Month, Day) 88-03-13	15. PAGE COUNT 144
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Artificial Intelligence, Software Engineering, Knowledge-based Systems, Knowledge-based programming tools.		
19. ABSTRACT (Continue on reverse if necessary and identify by block number) The software crisis initiated a major change in the perspective of software engineering. While conventional methodologies may have met software development requirements a decade ago, the present scale of programming has made automation of the development process imperative. Recent research has focused on the application of artificial intelligence (AI) techniques to software engineering. The ultimate goal is the automation of the entire software development life cycle. An overview of the software development life cycle is presented. The feasibility of incorporating AI methods for automating the traditional and prototyping approaches to software development is explored. A number of current research projects which apply AI to software engineering tasks, including a knowledge-based software project manager are discussed. Future research directions are highlighted. (KT)					
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED / UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL Major George E. Thurmond II			22b. TELEPHONE (Include Area Code) (404) 8 94-3110		22c. OFFICE SYMBOL ASBG - I

This work was done under Interagency Agreement DOE number 1662-1662-A1 for the United States Army Institute for Research in Management Information, Communications, and Computer Sciences (AIRMICS), the RDTE organization of the United States Army Information Systems Engineering Command (USAISEC). This report is not to be construed as an official Army position, unless so designated by other authorized documents. The material included herein is approved for public release, distribution unlimited. Not protected by copyright laws.

THIS REPORT HAS BEEN REVIEWED AND IS APPROVED.



s/ Glenn E. Racine
Glenn E. Racine, Chief
Computer and Information Systems Division

s/ John R. Mitchell
John R. Mitchell
Director, AIRMICS

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

POTENTIAL APPLICATIONS OF ARTIFICIAL INTELLIGENCE TO THE FIELD OF SOFTWARE ENGINEERING

M. Emrich, A. Agarwal, B. Jairam, N. Murthy

**Intelligent Systems Group
Data Systems Research and Development Program
Oak Ridge Reservation
Oak Ridge, TN 37831-6100**

**Martin Marietta Energy Systems, Inc
for the
U.S. Dept. of Energy
Under Contract No. DE-AC05-84OR21400**

Prepared for

**U. S. ARMY INSTITUTE FOR RESEARCH IN MANAGEMENT INFORMATION,
COMMUNICATIONS, AND COMPUTER SCIENCES
(AIRMICS)**

under Interagency Agreement DOE No. 1662-1662-A1

October, 1988

TABLE OF CONTENTS

	<u>Page No.</u>
ABSTRACT	vii
ACKNOWLEDGEMENTS	ix
1. INTRODUCTION	1
2. OVERVIEW OF THE TRADITIONAL SOFTWARE DEVELOPMENT LIFE CYCLE	9
2.1 REQUIREMENTS PHASE	9
2.2 SPECIFICATIONS PHASE	9
2.3 DESIGN PHASE	11
2.4 CODING PHASE	11
2.5 IMPLEMENTATION AND VERIFICATION PHASE	12
2.6 MAINTENANCE PHASE	12
2.7 DISADVANTAGES OF THE TRADITIONAL APPROACH	12
2.7.1 Error Correction	13
2.7.2 Missed Deadlines	13
2.7.3 Elimination of Alternatives	13
2.7.4 Redundancy	13
3. OVERVIEW OF PROTOTYPING APPROACH TO SOFTWARE DEVELOPMENT	15
3.1 ADVANTAGES OF PROTOTYPING	17
3.1.1 Reduced Specification and Design Errors	17
3.1.2 Reduced Time Required to Produce Systems	17
3.1.3 Increased User Satisfaction	17
3.1.4 Increased Design Options	18
3.2 RESOURCES ASSOCIATED WITH PROTOTYPING	18
3.2.1 On-line Interactive Systems	18
3.2.2 Fourth Generation Languages	18
3.2.3 Generalized Software	19

TABLE OF CONTENTS (cont.)

	<u>Page No.</u>
3.3 COMPARISON OF RESOURCES NEEDED FOR THE TWO METHODOLOGIES	19
3.3.1 Development Cost and Time	19
3.3.2 Resource Cost	20
3.3.3 Opportunity Cost	20
3.3.4 Operating Cost	20
3.3.5 Maintenance Cost	21
3.4 CIRCUMSTANCES WHICH MAKE PROTOTYPING INEFFICIENT	21
3.4.1 Well-Defined Goals	21
3.4.2 Limited User Time	21
3.4.3 High Resource Acquisition Cost	21
4. AI AND SOFTWARE ENGINEERING	23
5. AI/SOFTWARE ENGINEERING CROSSOVER	27
6. KNOWLEDGE-BASED SOFTWARE ASSISTANT (KBSA)	29
7. PROGRAMMER'S APPRENTICE (PA)	31
8. KNOWLEDGE BASED PROGRAMMING ASSISTANT (KBPA)	33
9. GLITTER	35
10. SOFTWARE MANAGER APPRENTICE	37
10.1 RELEVANT RESEARCH	37
10.2 CURRENT EFFORT	38
10.3 SOFTMAN SYSTEM	41
10.3.1 Modules 1 and 2	43
10.3.2 Module 3	44
10.4 CURRENT STATUS	44
10.5 ENHANCEMENTS	46
11. DISCUSSION	51
12. REFERENCES	53

TABLE OF CONTENTS (cont.)

	<u>Page No.</u>
13. ANNOTATED BIBLIOGRAPHY	59
13.1 GENERAL	59
13.2 PROTOTYPING	72
APPENDIX A: SOFTMAN SCREENS	75
APPENDIX B: SAMPLE CONSULTATION	109
APPENDIX C: GLOSSARY OF RELEVANT TERMS	139

LIST OF FIGURES

	<u>Page No.</u>
Figure 1. Traditional Approach to Software Development	10
Figure 2. Prototypical Approach to Software Development	16
Figure 3. Sample SOFTMAN Rules	39
Figure 4. Sample SOFTMAN Frame	40
Figure 5. SOFTMAN Structure	42

LIST OF TABLES

Table 1. AI/Software Engineering Systems: Status . .	3
Table 2. AI/Software Engineering Systems: Functions .	6

ABSTRACT

The software crisis initiated a major change in the perspective of software engineering. While conventional methodologies may have met software development requirements a decade ago, the present scale of programming has made automation of the development process imperative. Recent research focusses on the application of artificial intelligence (AI) techniques to software engineering. The ultimate goal is the automation of the entire software development life cycle.

An overview of the software development life cycle is presented. The feasibility of incorporating AI methods for automating the traditional and prototyping approaches to software development is explored. A number of current research projects which apply AI to software engineering tasks, including a knowledge-based software project manager, are discussed. Future research areas are highlighted.

ACKNOWLEDGEMENTS

The authors acknowledge the information assistance provided by Dr. Charles Rich of the Massachusetts Institute of Technology; Dr. Mehdi Harandi of the University of Illinois, Urbana; Dr. Stephen Fickas of the University of Oregon, Eugene; and Dr. Cordell Green of the Kestrel Institute, Palo Alto, California. Special thanks are also due to Drs. Rich and Harandi for their participation in a briefing at Ft. Belvoir, Virginia. The authors acknowledge assistance from the Central Research Library Staff, especially Judith Booth, Opal Russell, and Kendra Albright Jones for reference support. Additionally, appreciation goes to Julie Williams, Andrew M. Rochat, and Teresa Ladd for administrative and clerical support.

1. INTRODUCTION

Software engineering has emerged as an important area in computer science. It focusses on the development and implementation of large software systems (Zelkowitz, 1979). The field seeks to systematize and formalize the various activities involved in the science of programming and systems development. In the early seventies, the need for such formalization arose when conventional development methods failed to adequately meet the challenges posed by system design. This period was characterized by tremendous improvements in hardware technology. As a result of these advances, computer systems with high power and large capacity became feasible. Such systems led to the need for very large and extremely complex programming tasks. This phenomenon is designated as the software crisis (Sommerville, 1985).

While software engineering has developed into a practical methodology to overcome the software crisis, another important area in computer science that has emerged is artificial intelligence (AI). Although it is not the intent of this paper to precisely and formally define AI, its general meaning can be intuitively described as the science of making computers "intelligent." Intelligent in the sense that they are capable of performing actions that they have not been explicitly programmed to do (Barr, 1981; Barr, 1982). An illustrative case is the ability to reason and infer based on incomplete knowledge and to

evaluate alternatives using heuristics. Machines that can "learn" based upon their past experiences can also be included in this category.

The investigation presented in this report emphasizes integrating the two fields for achieving software development automation. The ultimate goal of this integration is to develop systems that generate reliable machine executable programs starting with the requirements definition phase (Partridge, 1986). Although considerable progress remains to be made before this goal is attained, current research has succeeded in automating some aspects of the software development life cycle (SDLC).

Several current research projects attempt to automate the various SDLC stages (Table 1). Some address more than one stage. The Knowledge Based Software Assistant (KBSA) project at the Rome Air Development Center (KBSA, 1987) is an attempt to develop a comprehensive, intelligent software development environment. The Programmer's Apprentice (PA) project at the Massachusetts Institute of Technology focusses on developing an intelligent system that emulates a human assistant. The ultimate objective is to continually gather problem-solving heuristics until the system approximates the expert's knowledge. The GLITTER project at the University of Oregon, Eugene seeks to formalize and automate the specifications process. The Knowledge Based Programming Assistant (KBPA) project at the University of

Table 1. AI/Software Engineering Systems: Status

Project	Theoretical stage	Demo/ Prototype	Research Under way	In Daily Use and / or Commercially Available	Current Status and Future trends
KRT	•••••	•••••	•••••	•••••	System has been completed; Commercially available; constant upgrade and refinement.
Programmer's Apprentice	•••••	•••••	•••••		Code generator module complete; address other stages within next two years
CHI	•••••				Project stopped; provided basis for REFINE at Kestrel Institute
PSI	•••••				Project Stopped
GLITTER	•••••	•••••	•••••		Prototype is being developed for specification phase; currently diversifying into analysis area with a system called KATE
Design Aid System/KBPA	•••••	•••••	•••••		Prototypes of coding unit and debugging unit are ready and are being used for educational purposes in the undergraduate curriculum; design and testing modules will be addressed during next eight years.
Intermetrics Code Generator	•••••	•••••		•••••	Retargetable code generator; commercially available from Intermetrics; being enhanced to generate code for different IBM mainframes.

Table 1. (continued)

Project	Theoretical stage	Demo/ Prototype	Research Under way	In Daily Use and / or Commercially Available	Current Status and Future trends
DARTS	•••••	•••••		•••••	Program synthesis tool; primarily in-house use; minor modifications for improved performance
Arrowsmith-P	•••••				Project stopped; M.S. thesis work.
SQAM	•••••	•••••	•••••		Going on-line Fall 87; modifying to operate under Unix and PC environments.
TIMM/Tuner	•••••	•••••	•••••	•••••	Being used in-house at GRC; available commercially from GRC.
ES/AG	•••••				Project Stopped.
ESB	•••••	•••••			Prototype is up and running
KBSA	•••••		•••••		Work on the facets going on at Kestrel, Honeywell, Sanders and USCLA
ACES	•••••	•••••	•••••		Cost estimation system for Ada projects Demonstrational prototype is ready. Future enhancements include development of a more friendly user interface, incorporation of a cost to completion feature and graphic output capabilities.

Illinois, Urbana focusses on developing an intelligent design, coding, testing, and debugging tool.

Table 2 lists several other projects that attempt to automate the various stages of the SDLC. While most of these focus on activities such as requirements analysis, specifications, design, and code generation, the effort has not been toward applying AI techniques for improving rapid prototyping. Rapid prototyping is an iterative process of developing programs from incomplete specifications. Since AI involves solutions to problems with incomplete knowledge, it can be useful for this purpose.

Another important aspect of the integration is automated software management support. Efforts like Time Line (Breakthrough Software Corporation, Novato, California) and Harvard Project Manager (Software Publishing Corporation, Mountain View, California) were not knowledge-based. In addition, little work has been done in software management using metric-based measurements.

Metric-based software measurement is a branch of software management which uses past performance to make predictions about current projects. Early research in metrics was nearly abandoned since the results obtained from metrics were sometimes controversial and substantially different from actuals. Another reason for the lack of interest was the difficulty in collecting data for the metrics. However, in recent years, software quality has become increasingly important. This has led to the revival

Table 2. AI/Software Engineering Systems : Functions

Project	Specifications and Analysis	Design	Code Generation	Testing / Debugging	Maintenance	Management	Integration of SDLC	Organization
KRT	••	••	••					McDonnell Douglas Houston Astronautics
Programmer's Apprentice	••	••	••		••		••	MIT
CHI	••	••	••					Kestrel Institute
GLITTER		••						Univ of Oregon, Eugene
Design Aid System/KBPA		••	••	••	••			Univ of Illinois, Urbana, Champaign
Intermetrics Code Generator			••					Intermetrics
DARTS		••	••					General Dynamics San Diego CA
Arrowsmith-P						••		Univ of Maryland
SQAM				••		••		Army AMCCOM and TDC

Table 2. (continued) Table 2. AI/Software Engineering Applications : Functions

Project	Specifications and Analysis	Design	Code Generation	Testing / Debugging	Maintenance	Management	Integration of SDLC	Organization
TIMM/Tuner					••	••		General Research Corp., Santa Barbara CA.
ES/AG	••							Bell Labs, NJ
ESB		••	••					Milton S. Eisenhower Research Center
KBSA	••	••	••	••	••	••	••	Kestrel Institute Palo Alto CA.
ACES	••							Institute of System Analysis, Bethesda, MD.

of metrics measures. Research has generated reliable metrics which can be used as project management guidelines. In addition, developments in hardware and software have made it easier to automatically collect metrics data (e.g., run time and source lines of code). A project is underway to develop a knowledge-based system, SOFTMAN Apprentice, for performing automated project management. The system derives its basic concept from metrics research performed at the University of Maryland (Basili, 1985).

2. OVERVIEW OF THE TRADITIONAL DEVELOPMENT LIFE CYCLE

An important concept in software engineering is that of the SDLC. As the name implies, it describes the process of developing a software system from conception through implementation and maintenance (Birrell, 1985). This process proceeds through several stages (Beregi, 1984), during which the system is successively transformed from high-level, natural language specifications to machine executable code (Figure 1).

2.1 REQUIREMENTS PHASE

The first stage in the SDLC is requirements analysis. In this phase, system analysts work to understand the existing software and hardware environment of the user. The user provides the problem description and specification to the analysts. The analysts identify potential solutions to any problems and rank them in order of certain parameters (e.g., cost and performance). Finally, they define the problem in terms of its functions and constraints. An acceptable solution to the problem and a statement of resources are determined. A document is written communicating requirements to the development team.

2.2 SPECIFICATIONS PHASE

From user specifications, software developers produce the architectural specifications. This framework identifies the interfaces and interrelationships between various system components, as well as the data flow between the components.

PHASES:

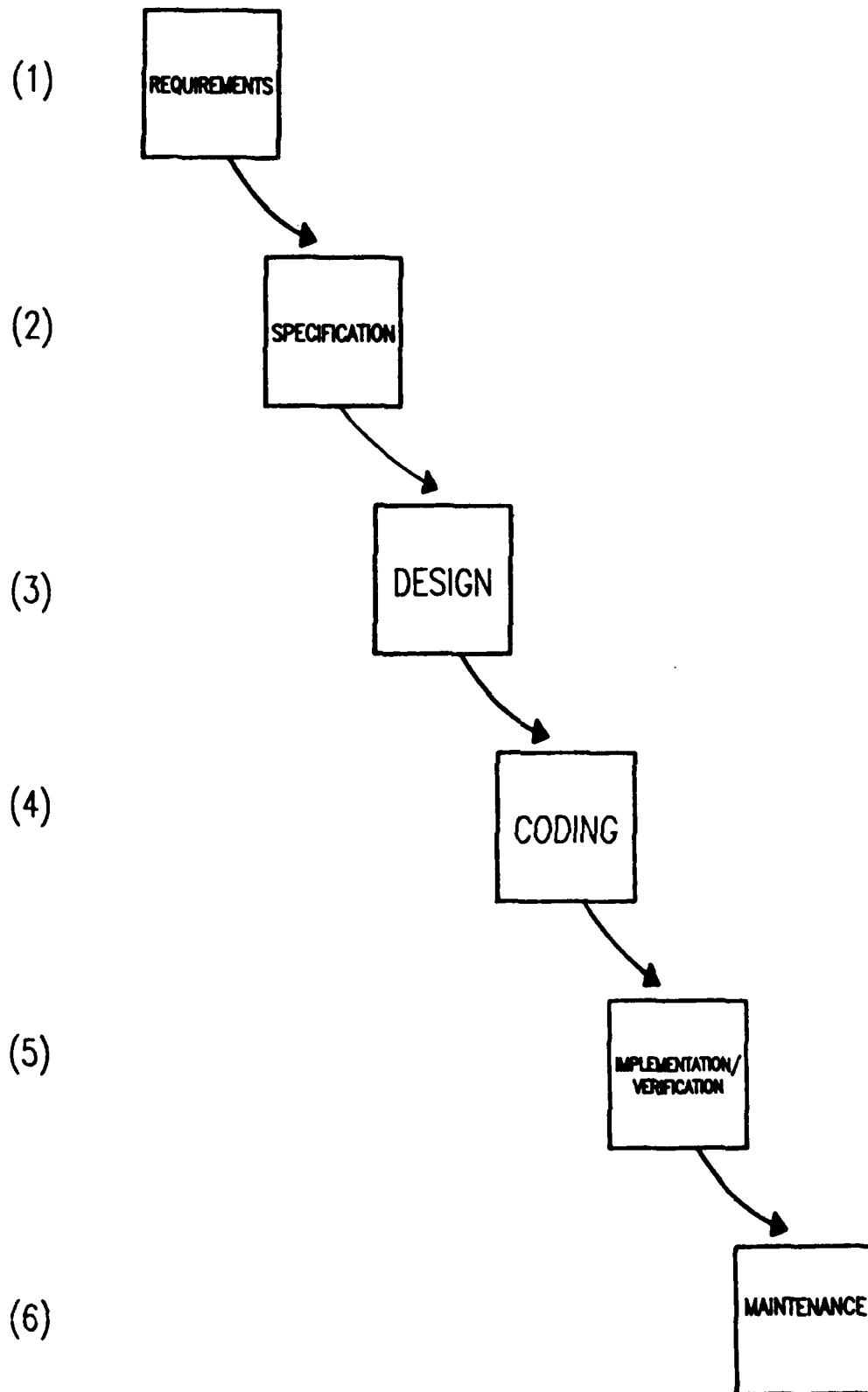


Figure 1. Traditional Approach to Software Development

Internal details of components are deferred to a later stage. The specifications are expressed in formal or semi-formal language. In addition, flowcharts or diagrams (e.g., HIPO charts) which pictorially depict the relationships between components may be developed. One statement at this level may expand to approximately one hundred lines of code.

2.3 DESIGN PHASE

It is in this phase that internal details of components are written using pseudo-code, flowcharts, decision tables, etc. One statement may expand to fifteen lines of code in the final product. Design defects are removed by manual inspections; undetected errors usually will not surface until the testing phase.

2.4 CODING PHASE

The algorithms for how the computer will solve the problem are developed during the design phase. After the design of the system is complete, coding begins. Herein, the abstract design is transformed via a programming language (e.g., PL/I, COBOL, and Pascal) into a compilable program.

Coding is followed by system testing which essentially consists of activities such as verification and validation. During verification, the correctness of the system is checked. Validation is the process of checking whether the system performs its intended duties and solves its intended problem. In other words, validation seeks to prove the system's correspondence

(Blum, 1986). The subtle point to be noted is that while a program might be correct (i.e., true to its specifications), it might not be the solution that is sought if the original specifications were wrong.

2.5 IMPLEMENTATION AND VERIFICATION PHASE

Programmers code individual modules in high-level languages and test each module. These modules are then integrated, and tested for performance, functionality, and reliability. Design errors not detected until this stage may cost as much as 75-80% more to correct (Martin, 1985).

2.6 MAINTENANCE PHASE

This phase coexists with the usage of the software product. It begins when the product goes on-line and involves correction of errors detected during product usage. Such modifications may involve major changes in the software which could be expensive in terms of time and money. The maintenance phase constitutes about 60-80% of the software life cycle (Sommerville, 1985; Spies, 1983).

2.7 DISADVANTAGES OF THE TRADITIONAL APPROACH

The traditional method suffers from various shortcomings when used for large scale software development. Errors may be costly to detect and correct. Due to the length of time between specification and implementation phases, deadlines may be missed. Exploration of alternative designs are not feasible. In addition, most existing reusable code is not utilized.

2.7.1 Error Correction

It may not be possible for the user to precisely define the product's operational requirements during the requirements phase. While operational parameters like performance and ease-of-use are more readily detected after the product has been implemented, specification inconsistencies are hard to detect. However, since they are identified late in the SDLC, such errors may lead to considerable changes in the original design itself.

2.7.2 Missed Deadlines

In large software projects, the intervening period between the requirements and verification phases is so long that product specifications often change by the time the product is implemented. Users may be dissatisfied with the final product. Changes are required to make the software acceptable to users. This can be time-consuming and expensive, resulting in missed deadlines.

2.7.3 Elimination of Alternatives

Since it is rather expensive to build a working model in this approach, neither users nor developers can explore the effects of alternative designs on system performance. Hence, the design may not be an optimal one. However, it is important to have inexpensive, throw-away models for examining these aspects.

2.7.4 Redundancy

The traditional approach does not take advantage of existing generalized software. Such software is usually available for the maintenance of databases, security, and data integrity (Naumann,

1982). These utilities are often rewritten explicitly for the user in the traditional approach. The same is true of various generalized input-output software.

The traditional approach to software development entails many modifications to the software. If user specifications are poorly defined, this process becomes inefficient. An approach which combines lengthy and sequential SDLC stages into a single, short activity would be more effective. The prototyping approach seeks to achieve this goal.

3. OVERVIEW OF THE PROTOTYPING APPROACH TO SOFTWARE DEVELOPMENT

Section 2 presented a short overview of the traditional approach to software engineering. An alternative to the "waterfall" methodology is rapid prototyping. A prototype is a working model which is built cheaply and quickly to test the validity of initial specifications and requirements. The user checks the product after every refinement to verify expectations and uncover inconsistencies (e.g., ease-of-use). The product is modified iteratively until it becomes acceptable to the user (Lipp, 1986).

Prototyping does require some initial resources (e.g., fourth generation languages) to enable fast development. The approach implicitly assumes that because human resources are most expensive, they are the most important. Therefore, one must risk other resources (e.g., hardware) prior to product development (Gremillion, 1983).

The basic motivation behind this approach is the observation that when a system is developed, the initial specifications are rarely complete and correct. The result is that the shortcomings of the system become apparent only after it has been developed; therefore, the design either needs to be modified or restructured. However, through prototyping an elementary design is quickly developed and implemented. Therefore, if a design is inadequate or faulty, it can be discarded and a new one developed. This process is iterated until all requirements of the system are met (Figure 2).

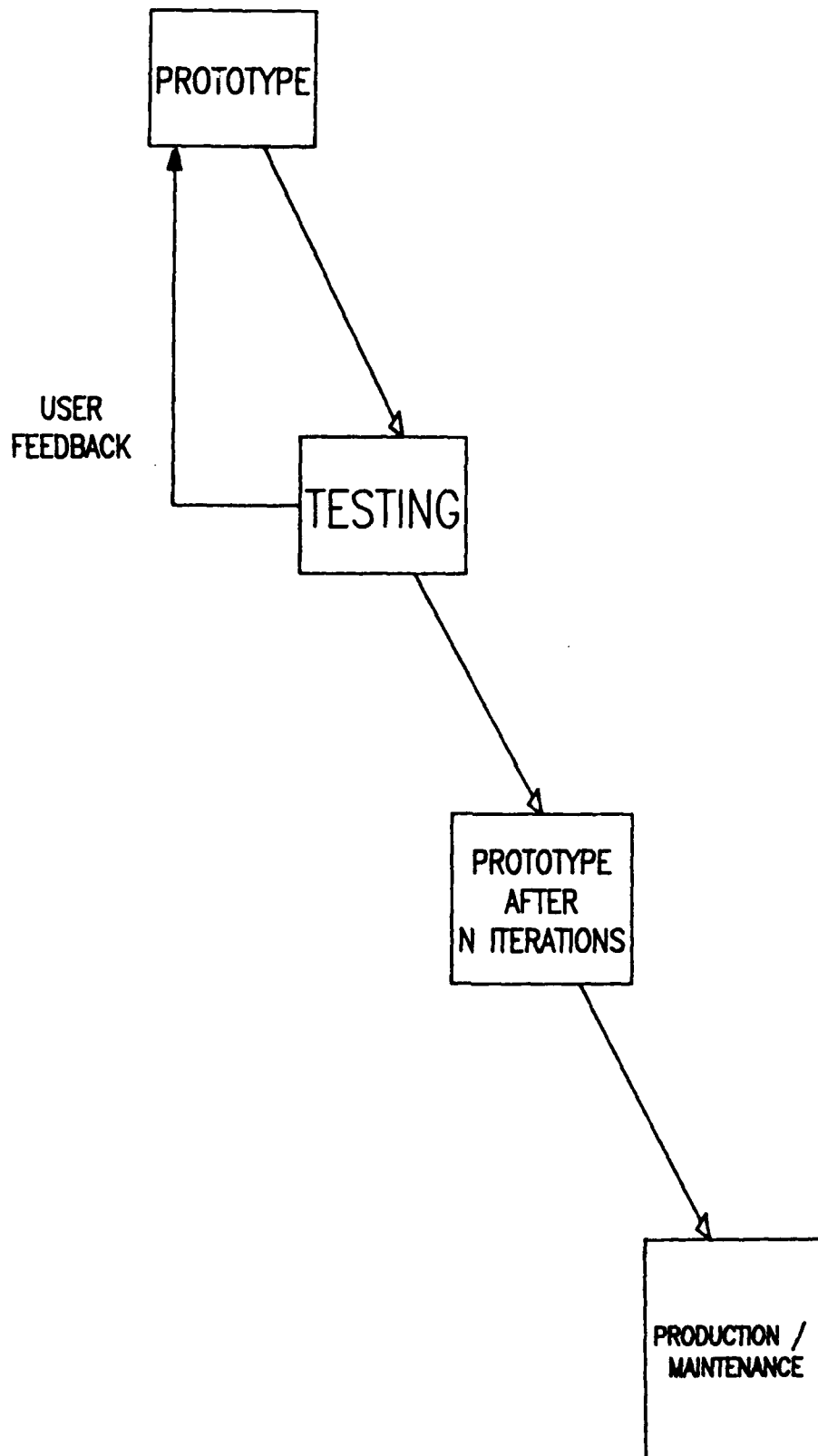


Figure 2. Prototypical Approach to Software Development

3.1 ADVANTAGES OF PROTOTYPING

3.1.1 Reduced Specification and Design Errors

Since this approach involves continuous interaction between the user and the developer, prototyping reduces specification errors. Because the first version of the product is available during early development (e.g., typically in days or weeks), design errors can be addressed quickly. Consequently, the final product has fewer inconsistencies when compared to the traditional approach.

3.1.2 Reduced Time Required to Produce Systems

Software is rarely rewritten; existing tools (e.g. databases, query languages, and report generators) are used. Often, the product is built using a fourth generation language which relieves the user of data representation and procedural details. Since errors are discovered quickly, less time is spent correcting them. For example, John Deere reports that inexperienced programmers could rewrite existing COBOL programs using IBM's ADF (Application Development Facility). A productivity twice that of the COBOL team was achieved (Haltz, 1980).

3.1.3 Increased User Satisfaction

The traditional approach usually does not involve the end user until the implementation phase. This disassociates the user from much of the software development process. However, rapid prototyping involves the user early in the development process. The user and developer participate in various sessions, thereby,

enabling the development of a close rapport. Appleton (1973) exemplifies prototyping in a functional applications system which was initially developed using the traditional life-cycle approach. The system was redeveloped using prototyping. It was reported that the new product eliminated user dissatisfaction and responded better to the users' dynamic environment.

3.1.4 Increased Design Options

For most systems, one particular solution cannot be designated as the best. In the traditional development method, the goal is to produce a feasible system the first time. Conversely, that may not be possible in a single-pass approach. However, in the prototyping method, when a trial-and-error approach is used, designs can be iteratively produced and discarded until an optimal solution is found.

3.2 RESOURCES ASSOCIATED WITH PROTOTYPING

3.2.1 On-line Interactive Systems

The philosophy behind the prototyping approach is to develop systems rapidly. Batch systems are not suitable for this purpose. However, on-line interactive systems are able to respond more rapidly to user needs.

3.2.2 Fourth Generation Languages

Fourth generation languages, also known as VHLL's (Very High-Level Languages), are program development tools which relieve the user of the data representation and procedural details of conventional languages (e.g., COBOL and PL/1). Fourth generation languages are primarily interpretive with features of

high-level coding incorporated. Such capabilities decrease applications development time. For example, a benchmark version of a management report system was developed at Heublein, Inc. Using COBOL, the effort required six months. The same program was created in half a day with Information Builders' fourth generation package, FOCUS (McCracken, 1980).

3.2.3 Generalized Software

Generalized software provides database creation and update capabilities without the development of complex programs. Various packages are available for editing the database and producing reports. In addition, security features are inherent in such software. The burden of programming details, not directly related to the project, is shifted from the programmer.

3.3 COMPARISON OF RESOURCES NEEDED FOR THE TWO METHODOLOGIES

3.3.1 Development Cost and Time

Evidence suggests that both the development cost and time in the prototyping approach are significantly less than that required in the traditional approach (Naumann, 1982). Scott (1978) describes a system that was estimated to cost \$350,000; however, using the prototyping approach, its development costs totalled \$35,000. In addition, other researchers (Read, 1981; Mason, 1982) report productivity gains with reduction in human resources.

Mason (1982) describes a rapid prototyping tool called ACT/1. This tool provides specifications using scenarios which are essentially user-system dialogues. These prototypes are translated by ACT/1 for production use.

3.3.2 Resource Cost

When high-level tools are involved, considerable resource acquisition costs may be associated with the prototyping approach. Since the cost is only for the initial installation, it may not be a major cost component if distributed over many projects. The traditional approach does not involve this cost factor.

3.3.3 Opportunity Cost

Opportunity cost must be taken into account when comparing the two development approaches. The prototyping approach produces a working system much faster. In the traditional method, the system is not available for a long period (i.e., from the start to completion). Therefore, there is an opportunity cost associated with the delayed availability of the product.

3.3.4 Operating Cost

Costs are involved during system operation. For example, generalized software may produce inefficient code with increased run-time. If coding in the traditional method were by expert programmers, operating costs may be higher for systems developed using the prototype model. However, because software teams routinely do not utilize expert programmers, the operating cost for the prototyping approach may not be significantly different.

3.3.5 Maintenance Cost

Maintenance costs are largely due to the various design and specification errors discovered after the product is in operation. With the prototyping approach, errors are more likely to be discovered early in the development process. Because changes are made early and at a higher level, this results in a lower maintenance cost for prototyped systems.

3.4 CIRCUMSTANCES WHICH MAKE PROTOTYPING INEFFICIENT

3.4.1 Well-Defined Goals

Some projects may have well-defined, stringent requirements. It may be possible to accurately define the specifications before the product is developed. In such situations, it is appropriate to develop the product using the traditional approach. This may also result in more efficient code in a lower-level language.

3.4.2 Limited User Time

An underlying assumption in the prototyping method is availability of user time. This may not always be possible or convenient. In such situations, prototyping may not offer any significant advantage over the traditional method.

3.4.3 High Resource Acquisition Cost

Initial resource costs are usually high for prototyping (e.g., fourth generation languages and generalized software). These costs are not cumulative; they form a minor fraction of the development cost if amortized over several projects. However, resource costs may increase on a per-project basis if fewer projects share the initial resource acquisition costs.

Due to its inherent development methodology, prototyping yields several benefits which cannot be envisioned within the traditional model framework. Disadvantages of prototyping related to inefficient code can be overcome by developing throw-away prototypes. Such prototypes are used to incrementally capture specifications. AI techniques can be used effectively to build throw-away prototypes from incomplete specifications.

4. AI AND SOFTWARE ENGINEERING

One solution suggested for the software crisis is the development of software using AI techniques. Advantages of such an approach are greater formalism and increased abstraction in the entities and operations involved. This can be seen as an intermediate step in developing methods that would allow increased automation of software development activities. One goal is to achieve higher programmer productivity.

The process of software development is an activity that is inherently susceptible to errors. Errors can occur at all phases of the SDLC. Their effects are amplified because they cascade. Errors in requirements analysis can lead to errors in specifications which, in turn, can lead to a faulty design, etc. On the other hand, even if the specifications are correct, it is possible for faulty code to be developed during the coding phase. Therefore, a program being erroneous either logically or syntactically, or both, is not an unusual phenomenon. Several means have been suggested and tried for proving software correctness (e.g., mathematical proofs, validation through metrics collection and analysis, and error seeding). However, none guarantee a completely bug-free and reliable program. Application of AI to software engineering tasks can provide a means of testing and verifying the correctness of programs.

Specification languages (e.g., the VHLL in the REFINE environment, GIST, and RML) can express software requirements in executable form. They can be enhanced incrementally (e.g., at each stage the new specifications are executable). In addition, maintenance can be effected by changing system specifications rather than applying patches to the source code (Goldberg, 1986). With maintenance tasks requiring up to 80% of the software development costs (Martin, 1983), this could result in a substantial savings.

Rapid prototyping has been strongly advocated as an alternative to the traditional software development method. In one research effort by Tavendale (1985), a prototype is developed from initial specifications and iteratively refined before the formal design phase begins. The prototype is generated in Prolog directly from the specifications. In this manner, an operational model can be used to verify initial specifications before system development proceeds. In other work, Fischer (1984) reports a rapid prototyping approach which focussed on supporting both specification and implementation stages of the SDLC.

Although the idea of rapid prototyping is not new, the feasibility of its practical implementation is fairly limited in the conventional software development environment. Whereas fourth generation languages lend themselves to prototyping, some authors advocate the use of AI techniques (AI, 1984; Wess, 1984). AI involves solutions to problems, knowledge about which is not complete prior to implementation. This knowledge may be the

specification of the problem or the data sets needed to reach the solution (Loomis, 1986). The inference engine of an AI system incorporates the search and reasoning methods to address such situations. It uses existing knowledge to develop partial solutions which may be further used to expand the knowledge base, thereby, reducing the abstraction level of the specifications.

Rapid prototyping involves incremental development of user programs based on incomplete user specifications. Such programs must be flexible enough to be changed. Therefore, AI methods can be used to prototype systems whose specifications and data are initially ill defined. The inference engine saves the programmer the burden of creating the detailed architecture for building such a program. Since it is independent of the prototype that it generates, the same inference engine can be used to develop different prototypes. AI methods also permit knowledge base tracing to locate incorrect specifications and to provide system documentation. With the use of AI, rapid prototyping becomes feasible as a practical system development methodology.

Utilizing AI techniques, human expertise can be captured in a "corporate memory" knowledge base. Such expertise can be made more readily available for use by company personnel. Additionally, the risk of losing information and knowledge when a person leaves the organization is reduced.

5. AI/SOFTWARE ENGINEERING CROSSOVER

The major areas into which AI applications can be divided include: robotics, computer vision, voice/speech recognition, machine learning, natural language processing, and expert (i.e., knowledge-based) systems (Mishkoff, 1985). The crossover between software engineering and the first area, robotics, is quite an unlikely prospect. However, computer vision research holds promise of software engineering (e.g., software design using flow charts and data flow diagrams) utilizing the concepts of machine vision and pattern recognition. In addition, voice and speech recognition can contribute extensively as an input medium for specifications and design implementations. Currently, little research is being conducted in these areas although such efforts could become more feasible as the technology develops.

The last two areas mentioned are the most promising for AI/software engineering interaction. Natural language processing is an area of active research (Schank, 1984). Easy-to-use front ends to expert systems that carry out software engineering tasks are the focus. One area where this is being applied is in the specifications phase. Natural language processors are used to gather the specifications and transform them into formal or semi-formal representations (Harandi, 1988). These can be further processed by automated programming tools (e.g., code generators).

The area that provides the widest scope for interaction between AI and software engineering is expert systems. As previously stated, the long-term objective of introducing AI into software engineering is to automate the SDLC. Currently, expert systems have been developed that perform requirements and specifications analysis, code generation, testing, documentation, etc. The following sections present major research efforts in these areas.

6. KNOWLEDGE-BASED SOFTWARE ASSISTANT (KBSA)

The KBSA project is a long-term effort being undertaken by the Rome Air Development Center (RADC) to provide an automated software development environment (Benner, 1987). The essential difference between the current software engineering paradigm and the one proposed by KBSA is that KBSA imposes more formalism on every SDLC activity. It gathers the evolutionary history of the system from conception to implementation. Therefore, it is able to provide a complete scenario of the implementation strategy, the decision making that went into a system, the rationale behind the decisions, the interfaces between the various units of the system, and the constraints imposed.

KBSA is characterized by the following features: a wide spectrum VHLL; an incremental, executable, and formal specification mechanism; a formal implementation scheme capable of validating and evaluating design decisions; and a maintenance facility at the specification level. In KBSA, implementation starts with a high-level abstract specification and proceeds through a series of correctness preserving transformations. KBSA can be perceived as an integrated system composed of a framework (Huseth, 1987) and the following five facets: Project Management Assistant (Jullig, 1986) which performs project definition, project monitoring, and user interface; Requirements Assistant (Harris, 1987) which deals with the informal user requirements; Specifications Assistant (Johnson, 1987) which formalizes requirements, validates them against user intentions and makes

them executable, and also provides a natural language front-end paraphraser; Performance Assistant (Goldberg, 1987) which does a performance analysis on the design decisions and evaluates them at all levels of the SDLC; and Development Assistant which derives an implementation from a completed specification.

Currently, work on four of the five facets has begun. KBSA is being investigated at four main institutions. Kestrel Institute (Palo Alto, California) is developing the Project Management Assistant and the Performance Assistant. Sanders Associates (Nashua, New Hampshire) is developing the Requirements Assistant. Work on the Specifications Assistant is being performed by the University of Southern California - Information Sciences Institute. The Development Assistant contract will be awarded during the fiscal year 1988. The framework is being developed by Honeywell Systems and Research Center (Minneapolis, Minnesota).

7. PROGRAMMER'S APPRENTICE (PA)

This MIT project follows the approach of duplicating a human expert's software development and problem-solving skills (Rich, 1987). The near-term goal is to develop an intelligent system which provides assistance during the different SDLC stages. PA uses a formalism called Plan Calculus to represent programs and programming knowledge. This scheme is a combination of the representational properties of flowcharts, data flow diagrams, and abstract data types. PA uses a library of several hundred plans; plans contain information regarding implementation methods and program forms.

The PA is composed of three parts: Requirements Apprentice (Rich, 1986a) provides assistance during the requirements analysis and specifications phase; Synthesis Apprentice (Rich, 1986b) aids in validation of the specifications, detection of inconsistencies, and other design decisions; and the Implementation Apprentice. There is a considerable overlap between the Implementation Apprentice and the Synthesis Apprentice. The main difference is the increased reasoning capabilities of the Synthesis Apprentice. While the Synthesis Apprentice provides support during several design phases, the Implementation Apprentice provides support only during implementation stages (e.g., code generation, editing, and program modification and maintenance).

Currently in PA, the Implementation Apprentice has been developed to a prototypical stage. A knowledge-based editor,

KBEmacs, facilitates program creation by allowing algorithmic fragments to be retrieved from a library (Waters, 1985; Waters, 1986). Prototypes of other components of the PA project (e.g., Synthesis Apprentice and Requirements Apprentice) are under development.

8. KNOWLEDGE BASED PROGRAMMING ASSISTANT (KBPA)

KBPA is under development at the University of Illinois (Urbana, Champaign). It is a knowledge-based support tool for software development (Harandi, 1986). It assists the programmer in the process of software development using knowledge-based techniques. KBPA is composed of four modules: design aid, coding aid, debugging aid, and testing aid. Each unit uses domain specific knowledge which is also a part of the global knowledge base. Such a structure facilitates the use of the modules as standalones or as integrated units.

The design aid module interacts with the user and obtains the high-level specifications (i.e., major components, inputs, and outputs) of the system. This is accomplished with the aid of data flow diagrams. Such diagrams not only describe a program in terms of the data that flows through it, but also the way that data is processed.

The coding module consists of a program editor and a design coder. This unit aids the programmer in identifying poor programming practices and advises the user in designing data structures. The design coder builds templates (i.e., abstract program plans); the editor transforms them into code.

The debugging module incorporates various features of intelligent debugging. One such model is the shallow model. It locates the cause of errors by having an "intuitive" idea of the specific program being debugged.

In the Spring of 1986, prototypes of coding and debugging modules had been implemented on the SUN and IBM RT. Currently, the debugging unit is in an advanced stage of development. It is used in debugging PASCAL programs as part of the University of Illinois' undergraduate curriculum. The design and testing modules will be the last in the series to be implemented.

9. GLITTER

GLITTER was developed at the University of Oregon (Eugene, Oregon). It is implemented in HEARSAY III (Balzer, 1980). The system is used in-house at the University to automate the requirements analysis process (Fickas, 1985a).

GLITTER is based on Balzer's (1980) Transformational Implementation (TI) model. The model starts with a formal specification of the problem. It applies a sequence of correctness preserving transformations until a specification conforming to the implementation conditions is reached. The original TI model suffered from a lack of automation and a formal scheme for representing goals, strategies, and design decisions.

The GLITTER system is used to overcome such shortcomings. It is an interactive transformation system that uses problem-solving techniques to automatically generate many of the transformation application steps. It provides a means for formalizing goals, strategies, and design decisions by specifying a language that allows their expression and manipulation. The syntax for specifying a goal consists of the keyword "GOAL" and a set of typed arguments.

Formalism in specifying and cataloging strategies is achieved through the use of "methods." Each method consists of a goal slot, a filter slot, and an action slot. The goal slot specifies the goal that needs to be achieved. The filter slot checks for the appropriateness of the method given the context. The action slot performs the operations needed to achieve the

goal. In addition, a method of conflict resolution between two strategies that can be applied simultaneously is provided. This is accomplished by the use of a "selection rule" mechanism. It is similar to the IF/THEN rule construct and provides the conflict resolution heuristic; this is usually a weight value that is used to evaluate the suitability of the strategy given the context.

One final advantage of GLITTER is that it provides a documentation of the problem-solving process. This is done by tracing the optimization sequence which led to the current state of the problem. In other words, a history of the problem-solving steps is provided. This aids in tracing the logic flow for future maintenance.

Current research focusses on the development of an automated requirements analysis system. The goal is to produce a complete and correct requirements definition from sketchy, informal, and incomplete user requirements (Fickas, 1985b). GLITTER is used extensively in this effort.

10. SOFTWARE MANAGER APPRENTICE

As the size of a computer system increases, the complexity of the software multiplies. The system becomes increasingly difficult to monitor and manage. Therefore, it is useful to investigate techniques for automating management decisions. In a common situation, more than one programmer is writing code, with one project manager controlling all activities. The manager must ensure that the project is completed on time, within budget, and is appropriate for the user's needs. Such decisions are usually based on the manager's past experience with handling similar development efforts.

10.1 RELEVANT RESEARCH

Little work has been done in the area of software management support which includes metric-based software measurement. One reason for this is the difficulty in collecting the data for making judgements based on metrics. In addition, there is debate regarding the accuracy of exclusively metric-based judgements. Most research done in the field of metrics has been in software complexity measurement (Belady, 1979; Curtis, 1979; Storm, 1979).

One research effort that does address metric-based software management is ARROWSMITH-P, a demonstration prototype developed for a Master's thesis at the University of Maryland (Basili, 1985). It was written using both rule-based and frame-based approaches. This dual effort demonstrated that both techniques

could produce comparably close results. The research project indicated that enough knowledge about software management could be input to obtain valid results from the system. ARROWSMITH-P required the user to make judgements about the software project (i.e., whether the number of lines of code were high or average, whether the CPU time was high or average, etc.). Therefore, the user had to be a semi-expert, while ARROWSMITH-P acted only as an assistant.

10.2 CURRENT EFFORT

The Software Manager Apprentice (SOFTMAN) is a software management system developed by the Center for Intelligent Systems (Oak Ridge, Tennessee) for the Army Institute for Research in Management Information, Communications, and Computer Science (AIRMICS) in Atlanta, Georgia. SOFTMAN does not require the user to have expert knowledge. Instead, the user provides quantitative inputs for the measurement metrics; SOFTMAN makes the qualitative judgements. In its present development stage, the software demonstrates the feasibility (i.e., a proof of concept) of using expert systems as a technique for aiding software management. One advantage of using the expert system approach to project management is that knowledge about metrics can be readily stored in rule sets (Figure 3). Moreover, I/O can be accomplished using frame-bases (Figure 4). When in a tutor mode, expert systems can also provide "how" and "why" explanations about decisions (Emrich, 1985).

File: C:\IC\ICPASCAL\SOFTMAN.BWD

Intelligence/Compiler

[F10 for Menu]
Block Search Go to Window File Object Mas-scope b-Tree Quit Compile
[F9 marks verb]

```
execute
if pascal init and cls and
do get com 'com'
until last com 'com' ;

last com 'com'
if 'com' = 7 and no-backtrack
or cls and
assert mmenu is 'com' and invoke rset1
and clean and cls and no-backtrack and fail ;

get com 'com'
if pascal mmenu, 'com' and no-backtrack ;

err
if do get err 'c'
until last err 'c' ;

get err 'c'
if pascal wrerr, 'c' and assert error is 'c' and no-backtrack ;
```

Figure 3. Sample SOFTMAN Rules

File: C:\IC\ICPASCAL\RESULT.FRM

Intelligence/Compiler

Block Search Go to Window File Object Mas-scope b-Tree Quit Compile

[F10 for Menu]

[F9 marks verb]

Frame: frm

Parent: Thing

Slot: Time Schedule Value:

Slot: Code Length Value:

Slot: Productivity Value:

Slot: Errors Value:

Slot: CPU Time Value:

Slot: Computer Runs Value:

Figure 4. Sample SOFTMAN Frame

The SOFTMAN system differs in several essential ways from ARROWSMITH-P. Since ARROWSMITH-P required the user to make qualitative judgements, there was no mathematics of metrics involved. In contrast, the SOFTMAN system will calculate metric values and make judgements based on the ranges in which the calculated metric value falls. In SOFTMAN, uniform metrics are not assumed throughout the development stage. Coding is divided into early, middle, and late stages. Different metric judgements are used for each stage. In addition, differences due to language, productivity levels, and design considerations are used to test a project's "health."

A primary aim of the system is to monitor the progress of a time-bound software project. It checks for unexpected behavior based on historical data collected from the environment. If anomalies are detected, SOFTMAN will issue warnings. The manager can then go deeper into the system to locate the problem and to receive suggestions regarding corrective actions. Another use of the system is as a tutor for training new personnel. In this role, SOFTMAN can assist future managers with project parameter estimation.

10.3 SOFTMAN SYSTEM

SOFTMAN is written using an expert system development tool, Intelligence/Compiler (IntelligenceWare, 1986) and Turbo Pascal (Borland, 1986). The overall system structure is shown in Figure 5. The consultation option consists of three modules. DETERMINER (i.e., Module 1) determines if there is a problem in

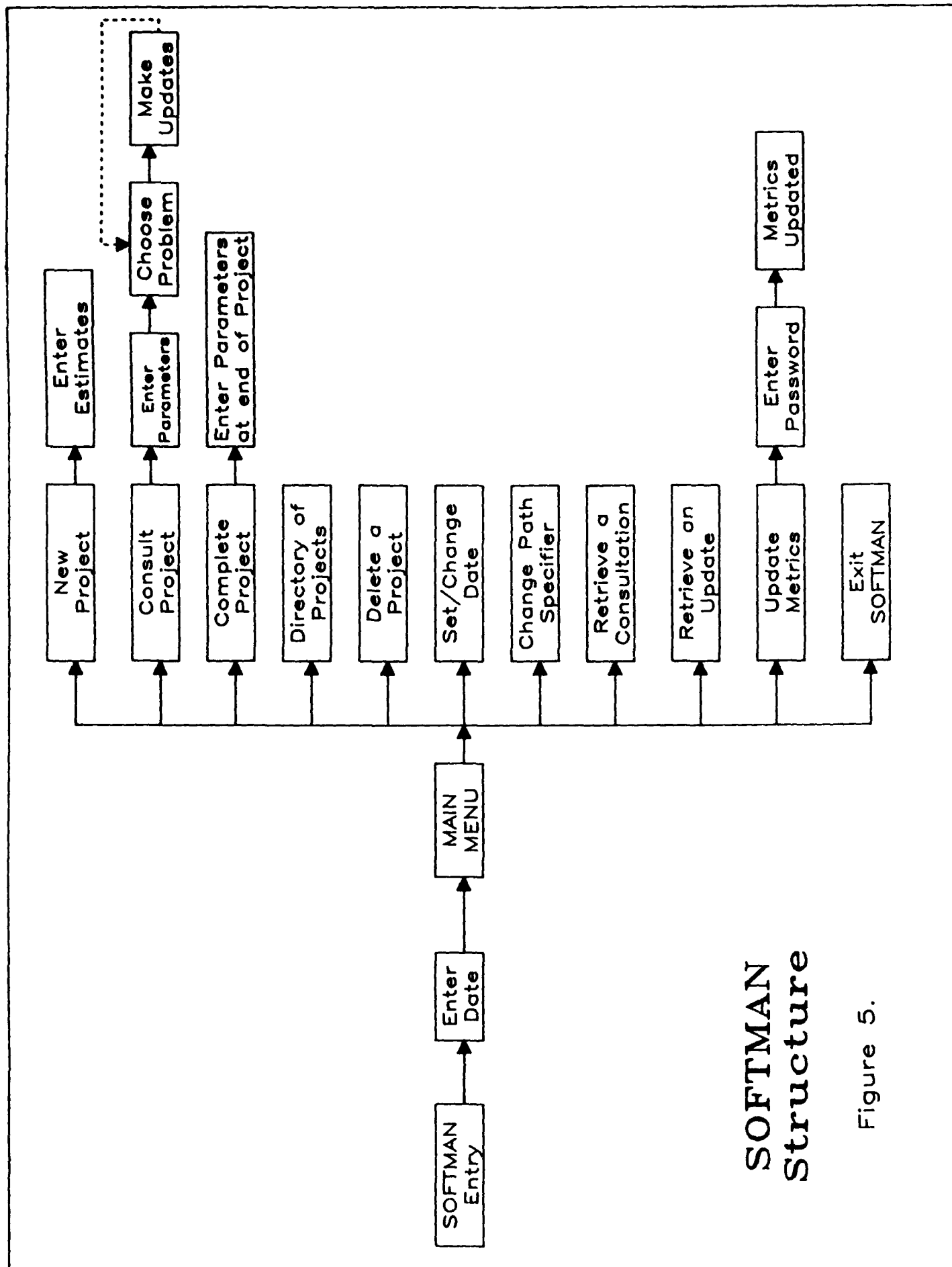


Figure 5.

the software project. IDENTIFIER (i.e., Module 2) identifies what the problem is and where it is located. REMOVER (Module 3) recommends appropriate actions to be taken. The system is menu driven (Appendix A). It guides the user through question-and-answer sessions which primarily require short, numeric answers, or menu solutions.

10.3.1 Modules 1 and 2

Upon initial entry into the system, the user is prompted to enter the current date. This information is used to make judgements about timely project completion. The "Main Menu" lists available utilities (e.g., deleting projects, retrieving consultations, and retrieving project updates).

When a new project is assigned to a manager, the "New Project" option is used to enter the project parameters (e.g., a unique project name, manager's name, estimated code size, number of coders, date of start of coding, and estimated time for project completion). Each parameter is input as a response to a question. Once the project has been initiated, the manager can intermittently consult SOFTMAN about the project's progress at a given coding stage. Based on user responses and project parameters, SOFTMAN will respond with a list of anomalies.

Context sensitive help for each question is provided at the bottom of the screen. Help messages are displayed in a different color. Additional help is available and can be called by pressing <Enter> in place of a numeric response.

10.3.2 Module 3

In the REMOVER module, the manager obtains a list of possible causes for a problem that has been detected by Module 2. To investigate these causes further, the manager will be asked more specific questions about the parameters. Based on the answers, SOFTMAN will suggest possible solutions. Some of these will require the manager to update project estimates that were initially entered. These updates will be stored and can be retrieved for later viewing.

By asking the manager questions regarding the quality of system design, the REMOVER module will provide links to the design and testing phases. It is possible that SOFTMAN will suggest that the manager backtrack and reevaluate the design before proceeding further in the coding stage. Since the solutions for the problems require qualitative judgements, Module 3 will be knowledge intensive (i.e., rule based).

10.4 CURRENT STATUS

The present system has five input development parameters gathered in Modules 1 and 2:

- 1) Number of lines of code.
- 2) Number of programmer hours.
- 3) Number of computer runs.
- 4) Number of software changes.
- 5) Amount of CPU time used.

Based on these parameters, Module 2 makes judgements on six outputs (i.e., project schedule, productivity, code length, cpu time, changes, and CPU runs) and identifies them as having a problem or not. From this list, the user can analyze one problem at a time. When a problem is selected, a color-bar comparison of the "Optimal Value" for that output parameter and the "Actual Value" from the project is provided. In addition, a brief text is displayed to explain the problem. Currently, further help in problem solution is provided for some output parameters (e.g., "Project Time Schedule").

After a consultation, the user may elect to store the results for later retrieval. Such data can be used for guiding new estimates. If the user elects not to save the session, a message is displayed warning that all recent inputs and outputs will be lost.

The user is given an opportunity to revise initial project parameters. Via the "Main Menu," each update made to a project is stored. Updates can be retrieved for future consultation.

Since the data for the metrics are environment-dependent, the Main Menu has an option for updating the metrics data files. Data for this utility is collected from all projects that have been run through the SOFTMAN "Complete" utility. Upon completion of each project, the manager should run this utility and enter the final values of all parameters. The "Metrics Update" utility can then be run to ensure that the metrics are adjusted by the data from the completed project. This utility is password

* protected so that unauthorized persons cannot change the data files. It is important to note that the changes made in this utility are permanent.

The current prototype performs some error checking functions. It differentiates between numbers and strings. It does not permit illegal choices. It displays menus for options and uses graphics for improved I/O operations. Additionally, if the user does not understand a particular question, help is provided.

10.5 ENHANCEMENTS

SOFTMAN is under development as a working prototype. Its purpose is to show the feasibility of applying expert systems to software project management. An expanded system could have several capabilities.

One major enhancement is to make the metrics dynamic. The data used to make judgements will be dynamically collected and updated in the data files for a particular environment. Although user inputs will influence the data files, all such operations will be transparent to the user. Currently, only the language-dependent environment parameters are dynamic (i.e., they are read from the disk when SOFTMAN is first initialized). The user can change this data in the ENV.DAT file. In the future, each time data for a new project is entered, the system will automatically update the metrics in the data files.

A utility to retrieve previous projects relevant to a particular search (e.g., manager name or date of project start) is planned. This facility can aid managers when estimating new projects. Since projects can be recalled and consulted, such capability can also be used to train new personnel.

Enhanced help facilities can aid decision making and improve manager judgements. At present, such capabilities are limited. Since it is quite common for users to misunderstand questions and feed erroneous data, an expanded narrative could improve user inputs. In addition, minimal capabilities are provided by Intelligence/Compiler to include "how" and "why" explanations. These capabilities must be built using the Turbo Pascal interface.

Future plans include a separate utility for each development parameter input (e.g., number of lines of code). These utilities can be used to calculate parameter values. In addition, external Pascal programs could be called from within SOFTMAN to return a parameter's value.

A utility to monitor a manager's performance with regard to estimate accuracy can be developed. When the manager uses the "New Project" utility and enters estimates, the system will consult past standards for similar projects. Therefore, a guide for current judgements and individual manager performance will be provided.

The present research projects (both SOFTMAN and ARROWSMITH-P) assume that lines of code is an adequate and useful

parameter for gauging a project's "health." However, a more accurate measure may be the number of functional modules (e.g., menu routines and report generators). This will be investigated further in SOFTMAN.

At present, SOFTMAN addresses only the coding stage of the SDLC. There are no links to the design phase or to the maintenance phase. However, the system could include design considerations in the REMOVER module asking the manager to evaluate the quality of the system design. It has been reported (Boehm, 1975; Hamilton, 1976) that approximately 65-75% of all errors occur due to faulty design. Since a faulty or weak design often leads to coding problems, this capability could increase the overall management efficiency.

One of the important differences between SOFTMAN and ARROWSMITH-P is that SOFTMAN uses different metric standards in each coding stage. To inform the manager of the current coding stage and advise regarding the appropriate coding stage, the individual stages need to be clearly defined. To determine such differences, temporal environment-dependent data will be gathered.

In summary, SOFTMAN has shown that software management is a candidate area for automated support. By 1990, it is estimated that the shortfall of software engineers and analysts will reach one million in the aerospace/defense industry alone, (Vosburgh, 1987). Since it is anticipated that this lack of technical personnel will not be fulfilled, it must be compensated by

providing intelligent support tools. Better and more efficient management of the available resources will enhance the ability to build reliable systems. Sufficient interest has been generated in this area to encourage future research.

11. DISCUSSION

The application of AI to software engineering can qualitatively and quantitatively improve the software development process. Such an application is most feasible through expert systems. This is due to the nature of software design and development. It is an activity that requires knowledge of not only programming techniques, but also of the application domain. Since expert systems can incorporate both kinds of knowledge through knowledge bases, they offer a good prospect for introducing AI in software engineering. An investigation of current research projects corroborates this notion. In addition, it has provided an insight into the path such research has taken and an indication of future direction.

Extensive research has been conducted in automatic code generation and specification languages. However, little effort has been devoted to automating the requirements analysis phase. Application of AI concepts such as natural language processing to automate this phase can lead to increased speed and ease in system development.

The prototyping approach can lead to a substantial increase in productivity. However, conventional software development procedures do not allow the adaption of this approach as a feasible system development methodology. The application of AI can make prototyping a viable alternative to the traditional SDLC. Therefore, research efforts to increase automation in prototyping should be encouraged.

Automated software management support is another area that has not generated much interest in the past. SOFTMAN and ARROWSMITH-P which are both knowledge-based have demonstrated the feasibility of using AI techniques for this process. Although the validity of metric-based measurements has been in debate, SOFTMAN has illustrated that metrics can provide sufficient information to judge project "health."

Current research has succeeded, to a limited extent, in automating certain SDLC phases. Research efforts such as the ones highlighted in this report focus on automating several of these phases. In the near-term, more expert systems addressing specific SDLC activities will emerge. However, a fully integrated, automated software development environment is a long-term goal.

12. REFERENCES

- "AI Environment Speeds Software Development," Systems and Software, 3(8), 111-118 (1984).
- Appleton, D. S. "System 2000 Database Management Systems," Boston, Massachusetts, November 1-2, 1973.
- Balzer, R., "Transformational Implementation: An Example," IEEE Transactions on Software Engineering, SE-7(4), 1981.
- Balzer, R., L. D. Erman, P. London, and C. Williams, HEARSAY-III: A Domain-Independent Framework for Expert Systems, pp. 108-110, in Proceedings of the First Annual National Conference on Artificial Intelligence, August 18-21, 1980, Stanford, California, William Kaufmann, Inc., Los Altos, California, 1980.
- Barr, A. and E. A. Feigenbaum, Handbook on Artificial Intelligence: Vol. 1, Kaufman, Los Altos, California, 1981.
- Barr, A. and E. A. Feigenbaum, Handbook on Artificial Intelligence: Vol. 2, Kaufman, Los Altos, California, 1982.
- Basili, V. and C. Ramsey, "Arrowsmith-P: A Prototype Expert System for Software Engineering Management," pp. 252-264 in Expert Systems in Government Symposium, October 24-25, 1985, McLean, Virginia, IEEE Computer Society Press, Washington, D. C., 1985.
- Belady, L. A., "An Anti-Complexity Experiment," pp. 128-129 in Workshop on Quantitative Software Models, October 9-11, 1979, Kiamesha Lake, New York, IEEE Computer Society Press, Washington, D. C., 1979.
- Benner, K. M. and D. A. White, "The Knowledge-Based Software Assistant: Overview," in Proceedings of the 2nd Annual Knowledge-Based Software Assistant Conference, August 18-20, 1987, Utica, New York, Rome Air Development Center, Griffiss Air Force Base, New York, 1987.
- Beregi, W. E., "Architecture Prototyping in the Software Engineering Environment," IBM Systems Journal, 23(1), 4-17 (1984).
- Birrell, N. D. and M. A. Ould, A Practical Handbook for Software Development, Cambridge University Press, New York, New York, 1985.

- Blum, B. I. and V. G. Sigillito, "An Expert System for Designing Information Systems," Johns Hopkins APL Technical Digest, 7(1), 23-30 (1986).
- Boehm, B. R., R. McClean, and D. Urfrig, "Some Experience with Automated Aids to the Design of Large Scale Reliable Software," pp. 105-113 in: International Conference on Reliable Software, April 21-23, 1975, Los Angeles, California, IEEE, New York, New York, 1975.
- Borland, Turbo Pascal, Version 3.01, Borland International, Inc., Scotts Valley, California, 1986.
- Curtis, B., "In Search of Software Complexity," pp. 95-106 in Workshop on Quantitative Software Models, October 9-11, 1979, Kiamesha Lake, New York, IEEE Computer Society Press, Washington, D. C., 1979.
- Emrich, M. L., Expert Systems Tools and Techniques, ORNL/TM-9555, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 1985.
- Fickas, S. F., "Automating the Transformational Development of Software," IEEE Transactions on Software Engineering, SE-11(11), 1268-1277 (1985a).
- Fickas, S. F., A Knowledge-Based Approach to Specification Acquisition and Construction, CIS-TR 85-13, University of Oregon, Eugene, Oregon, 1985b.
- Fischer, G. and M. Schneider, "Knowledge-Based Communication Processes in Software Engineering," pp. 358-368 in Seventh International Conference on Software Engineering, March 26-29, 1984, Orlando, Florida, IEEE Computer Society Press, Washington, D.C., 1984.
- Goldberg, A. T., "Knowledge-Based Programming: A Survey of Program Design and Construction Techniques," IEEE Transactions on Software Engineering, SE-12(7), 752-768 (1986).
- Goldberg, A. T. and D. R. Smith, "Performance Estimation for a Knowledge-Based Software Assistant," in Proceedings of the 2nd Annual Knowledge-Based Software Assistant Conference, August 18-20, 1987, Utica, New York, Rome Air Development Center, Griffiss Air Force Base, New York, 1987.
- Gremillion, L. L. and P. Pyburn, "Breaking the Systems Development Bottleneck," Harvard Business Review, 61(2), 130-137 (1983).

- Haltz, D. H., "ADF Experiences at John Deere," D303-SHARE, 50 (1980).
- Hamilton, M. and S. Zeldin, "Higher Order Software: A Methodology for Defining Software," IEEE Transactions on Software Engineering, SE-2(1), 9-32 (1976).
- Harandi, M. T. and M. D. Lubars, "A Design Environment for Software Systems," in Proceedings of the Conference on Expert Systems Technology in ADP Environment, November 1-3, 1987, Washington, D. C., Oak Ridge National Laboratory, Oak Ridge, Tennessee, 1988.
- Harandi, M. T., "Applying Knowledge-Based Techniques to Software Development," Perspective in Computing, 6(1), 14-21 (1986).
- Harris, D. R., "An Overview of the Knowledge-Based Requirements Assistant," in Proceedings of the 2nd Annual Knowledge-Based Software Assistant Conference, August 18-20, 1987, Utica, New York, Rome Air Development Center, Griffiss Air Force Base, New York, 1987.
- Huseth, S. and T. King, "A Common Framework for Knowledge-Based Programming," in Proceedings of the 2nd Annual Knowledge-Based Software Assistant Conference, August 18-20, 1987, Utica, New York, Rome Air Development Center, Griffiss Air Force Base, New York, 1987.
- IntelligenceWare, Intelligence/Compiler Manual, IntelligenceWare, Inc., Los Angeles, California, 1986.
- Johnson, J. R., "A Prototypical Success Story," Datamation, 29(11), 251-256 (1983).
- Johnson, W. L., "Overview of the Knowledge-Based Specification Assistant," in Proceedings of the 2nd Annual Knowledge-Based Software Assistant Conference, August 18-20, 1987, Utica, New York, Rome Air Development Center, Griffiss Air Force Base, New York, 1987.
- Jullig, R., KBSA-PMA Technical Report, Rome Air Development Center, Griffiss Air Force Base, New York, 1986.
- KBSA, 2nd Annual Knowledge-Based Software Assistant Conference, August 18-20, 1987, Utica, New York, Rome Air Development Center, Griffiss Air Force Base, New York, 1987.
- Lipp, M. E. (ed.), Prototyping: State of the Art Report, Pergamon Infotech, Maidenhead, Berkshire, England, 1986.

- Loomis, M. E. S. and T. P. Loomis, "Prototyping and Artificial Intelligence," pp. 65-73 in M. E. Lipp (ed.) Prototyping: State of the Art Report, Pergamon Infotech, Maidenhead, Berkshire, England, 1986.
- Martin, J., Fourth Generation Languages, Vol. 1, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985.
- Martin, J. and C. McClure, Software Maintenance: The Problem and Its Solution, Prentice-Hall, Inc., London, England, 1983.
- Mason, R. E. A., T. T. Carey, and A. Benjamin, "A Tool for Information Systems Prototyping," ACM Sigsoft Software Engineering Notes, 7(5), 120-125 (1982).
- McCracken, D. D., "Software Systems in the 80's: An Overview," Computerworld Extra, 14(38), 5-10 (1980).
- Mishkoff, H. C., Understanding Artificial Intelligence, Texas Instruments, Dallas, Texas, 1985.
- Naumann, J. D., "Prototyping: The New Paradigm for Systems Development," MIS Quarterly, 6(3), 29-44 (1982).
- Partridge, D., Artificial Intelligence: Applications in the Future of Software Engineering, Ellis Horwood Limited, Chichester, West Sussex, England, 1986.
- Read, N. S. and D. L. Harmon, "Assuring MIS Success," Datamation, 27(2), 109-120 (1981).
- Rich, C. and H. E. Shrobe, "Design of a Programmers Apprentice," pp. 138-173 in P. H. Winston and R. H. Brown (eds.) AI: An MIT Perspective, Vol. 1, MIT Press, Cambridge, Massachusetts, 1979.
- Rich, C. and R. C. Waters, The Programmer's Apprentice Project, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1987.
- Rich, C. and R. C. Waters, Toward a Requirements Apprentice: On the Boundary Between Informal and Formal Specifications, A.I. Memo No. 907, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1986a.
- Rich, C. and R. C. Waters, The Programmer's Apprentice: A Program Synthesis Scenario, A.I. Memo No. 933, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1986b.
- Schank, R. C. and P. G. Childers, The Cognitive Computer, Addison-Wesley, Reading, Massachusetts, 1984.

- Scott, J. H., "The Management Science Opportunity: A Systems Development Management Viewpoint," MIS Quarterly, 2(4), 59-61 (1978).
- Sommerville, I., Software Engineering, Second Edition, Addison-Wesley Publishing Company, Reading, Massachusetts, 1985.
- Spies, P. B., "Designing Systems for Users," Library Hi Tech, 1(1), 75-84 (1983).
- Storm, I. L. and S. Preiser, "An Index of Complexity for Structured Programs," pp. 130-133 in Workshop on Quantitative Software Models, October 9-11, 1979, Kiamesha Lake, New York, IEEE Computer Society Press, Washington, D. C., 1979.
- Tavendale, R. D., "A Technique for Prototyping Directly from a Specification," pp. 224-229 in Proceedings of the Eighth International Conference on Software Engineering, August 28-30, 1985, London, England, IEEE, New York, New York, 1985.
- Vosburgh, J. R. and M. A. Tanous, "Software Productivity Looms as Aerospace/Defense Issue," pp. 153-157 in Proceedings of Technology Strategies '88 Conference on the U.S. Army Information Systems Engineering Command Advanced Technology Office, February 9-12, 1988, Alexandria, Virginia, The American Defense Preparedness Association, 1988.
- Waters, R. C., "KBEmacs: Where's the AI?," AI Magazine, 7(1), 47-56 (1986).
- Waters, R. C., "The Programmer's Apprentice: A Session with KBEmacs," IEEE Transactions on Software Engineering, SE-11(11), 1296-1320 (1985).
- Wess, B. P. Jr., "Artificial Intelligence Techniques Speed Software Development," Mini-Micro Systems, 17(11), 127-136 (1984).
- Zelkowitz, M. V., A. C. Shaw, and J. D. Gannon, Principles of Software Engineering and Design, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1979.

13. ANNOTATED BIBLIOGRAPHY

13.1 GENERAL

Ary, D. and S. Saib, "TIMM/TUNER - The Intelligent Vax Computer Tuner," VAX/RSTS Professional, 7(2), 32-40 (1985).

This article was written by two employees of the General Research Corporation to report on their usage of GRC's expert system shell, TIMM (The Intelligent Machine Model). They have used the knowledge of experts in adjusting a VMS-based VAX system's parameters to suggest performance tuning tasks. A "walkthrough" of a sample consultation is given followed by suggestions for system enhancements.

[Aid VAX Tuning]

Basili, V. R. and C. L. Ramsey, "ARROWSMITH-P - A Prototype Expert System for Software Engineering Management," pp. 252-264 in K. N. Karna (ed.) Proceedings of Expert Systems in Government Symposium, McLean, Virginia, October 24-25, 1985, IEEE Computer Society Press, Washington, D. C., 1985.

The authors built two versions of a software management expert system: one rule-based; one frame-based. The systems were developed in KMS, an expert system shell used at the University of Maryland. The knowledge bases contain possible causes for aberrations in such measurements as programming hours, computer time, and number of changes. When an abnormal software development pattern is detected, an explanation of possible causes is provided. When the prototypes were compared, it was determined that the rule-based version provided more complete solutions than the frame-based prototype. Results of the comparison are provided as are plans for system(s) revisions.

[ARROWSMITH-P - Management]

Blanchard, D. C. and R. M. Myers, "The Knowledge Representation Tool," pp. 137-147 in Proceedings of ROBEXS '85: The First Annual Workshop on Robotics and Expert Systems, NASA/Johnson Space Center, June 27-28, 1985, Instrument Society of America, Research Triangle Park, North Carolina, 1985.

In a sketchy, and, at times, difficult to follow article, the authors present a discussion of KRT (Knowledge Representation Tool). KRT is a LISP-based system that aids a software engineer in the "System Model" approach to structured analysis. (The System Model uses data flow diagrams, mini-specifications, and a data dictionary.) KRT represents knowledge in an object-oriented programming style; process objects are subdivided until they are refined to the level of mini-specifications. The authors suggest that KRT could aid software engineers in several ways; the primary example given is in the area of software maintenance.

[Aid Structured Analysis
and Software Maintenance]

Blum, B. I. and V. G. Sigillito, "An Expert System for Designing Information Systems," Johns Hopkins APL Technical Digest, 7(1), 23-30 (1986).

The authors point out that knowledge needed to develop software (i.e., software engineering knowledge) can be viewed in two major divisions: product dependent and algorithmic, or application specific and heuristic. The two are suitable for representation in an expert system knowledge base. Using this division as a guiding factor, the authors propose the development of an integrated environment for system building (ESB). ESB will consist of three modules: a definition model for capturing application domain knowledge; a transformation module where an expert system changes the specifications developed in the definitions module into executable specifications; and a generation module for generating the program.

[Aid Analysis, Specification, Design, Code Generation]

Cronk, R. N. and D. V. Zelinski, "ES/AG: System Generation Environment for Intelligent Application Software," pp. 96-100 in Proceedings SOFTFAIR II: A Second Conference on Software Development Tools, Techniques, and Alternatives, San Francisco, California, December 2-5, 1985, IEEE Computer Society Press, Washington, D. C., 1985.

An integrated set of software development tools and languages that is expert system-based is described - Expert System/Application Generator (ES/AG). The knowledge base contains three types of knowledge: factual, procedural, and non-procedural. Knowledge is represented by rules (for non-procedural knowledge) and frame-like symbols (for factual and procedural knowledge). Non-procedural control strategies, explanation facilities, a user interface to the problem-solving model, a LISP interpreter, and debugging facilities are included. ES/AG runs in a Unix environment and has been used at Bell Laboratories for several applications (e.g., equipment configuration and software cost estimating).

[ES/AG]

Dunning, B. B., "Expert System Support for Rapid Prototyping of Conventional Software," pp. 2-6 in Proceedings of Autotestcan '85: IEEE International Automatic Testing Conference, Uniondale, New York, October 22-24, 1985, IEEE, New York, New York, 1985.

The author reviews conventional software development methods and rapid prototyping procedures. He then says that the flexibility of a LISP processor and an expert system shell (e.g., KEE or EXPLORER) can make rapid prototyping easier. However, he does state that the savings in time and costs are highest during the design phase. Implementation usually means rewriting the entire prototype in a conventional language; this process takes seven times the original effort. Major advantages and a few disadvantages of this approach to software development are cited. Overall, a rather terse look at a topic that deserves more indepth analysis.

[Aid Rapid Prototyping]

Fickas, S. F., "Automating the Transformational Development of Software," IEEE Transactions on Software Engineering, SE-11(11), 1268-1277 (1985).

The author uses AI techniques to alleviate the major weakness (undermechanization) of Balzer's transformational implementation (TI) model. Fickas found that the formalization of goals, strategies, selection rationale, and human TI methods were areas that needed to be addressed before the model could be automated.

GLITTER was developed (written in HEARSAY III) and used for creating a package router and a small text editor. GIST was the specification language used in GLITTER. It was suggested that information generated by GLITTER on problem-solving steps could help in maintenance; research is currently underway to classify possible changes to specifications and to identify the associated salvageable code.

Overall, a somewhat confusing article. the small degree of automation offered may not justify the use of a new system and specification language.

[GLITTER/GIST - Program Transformation]

Frenkel, K. A., "Toward Automating the Software-Development Cycle," Communications of the ACM, 28, 578-589 (1985).

Two automatic programming research efforts are discussed: Intermetrics' (Cambridge, Massachusetts) compiler code generator, and the University of Waterloo's (Ontario, Canada) real-time debugging system, Message Trace Analyzer. While the author agrees with some researchers who feel that expert systems may be "oversold" and that a proliferation of AI languages could produce much of the same problems that now exist with the conventional software development languages, she also feels that the need for increased productivity is so great that any avenue of relief will be pursued. Furthermore, the work done now in expert systems will just add to the next generation of software productivity tools.

[Aids Code Generation/Debugging]

Haradhvala, S., B. Knobe, and N. Rubin, "Expert Systems for High Quality Code Generation," pp. 310-313 in Proceedings of the First Conference on Artificial Intelligence Applications, Denver, Colorado, December 5-7, 1984, IEEE Computer Society Press, Washington, D.C., 1985.

Intermetrics employees describe the evolutionary process of developing an expert system to aid in compiler code generation. The final system was based upon a modified version of Cattell's (CMU) Production Quality Compiler Compiler (PQCC) project for the Bliss-II. Written in Pascal, it runs on IBM 370's. The knowledge base contains some 500 production rules; a depth-first search strategy is followed. Development took three to six person months.

[Aid Code Generation]

Harandi, M. T., "Applying Knowledge-Based Techniques to Software Development," Perpsectives in Computing, 6(1), 14-21 (1986).

A discussion of the features of KBPA, a knowledge-based programming assistant developed at the University of Illinois. The author details the aspects and problems in such systems and highlights the prototype implementation of a design, coding, and debugging unit. The paper is good reading for understanding the issues in knowledge-based techniques.

[Design, Debugging, Code Generation]

Harandi, M. T. and M. D. Lubars, "A Knowledge Based Design Aid for Software Systems," pp. 67-74 in Proceedings SOFTFAIR II: A Second Conference on Software Development Tools, Techniques, and Alternatives, San Francisco, California, December 2-5, 1985, IEEE Computer Society Press, Washington, D. C., 1985.

The paper discusses the capabilities of a prototype developed by researchers at the University of Illinois at Urbana-Champaign. The system addresses the specification and design phases of software engineering. Using the dataflow modeling method of program design, the development process is viewed as a series of refinements (i.e., leveling). The system has three components: a knowledge base with design schemas, a data dictionary, and application domain knowledge; a design refinement unit which is an agenda-driven inference engine; and natural language and graphics interfaces. The prototype runs on a Sun workstation, and, according to the authors, has been used to develop small example systems.

[Aid Specification/Design]

Hill, C., "A Software Revolution Looms on the Horizon," InformationWEEK, 94, 40 (1986).

This article stresses the quantitative and qualitative improvements to be gained in MIS software development by utilizing automated productivity tools. It states that many Fortune 500 companies are employing such tools to eliminate applications backlogs and improve quality, citing significant reductions in development schedules. It concludes by emphasizing the benefits of development tools to both programming staffs and MIS managers.

[Future of MIS]

Janusz, P. E. and P. T. Eckert, "Software Quality Assessment Measure," pp. 282-284 in Proceedings of the Annual Reliability and Maintainability Symposium, IEEE, New York, New York, 1986.

The Army assessed the feasibility of using expert systems to aid quality assurance, Software Quality Assessment Measure (SQAM). The authors used manuals and reports to analyze one aspect of a QA officer's job (i.e., review of system storage allocations for adequacy). That segmented task was modeled using Teknowledge's expert system shell, M.1a (for expert consultation) and dBase III (for the operations checklist and menu generation).

[SQAM System]

Kornell, J., "A VAX Tuning Expert Built Using Automated Knowledge Acquisition," pp. 38-41 in Proceedings of the First Conference on Artificial Intelligence Applications, Denver, Colorado, December 5-7, 1984, IEEE Computer Society Press, Washington, D.C., 1985.

The paper covers the use of General Research Corporation's (GRC) expert system shell, TIMM. The author is a GRC employee who used the tool to develop a prototype. TIMM/Tuner tunes VAX computers to gain maximum performance. The system is modular in nature and contains seventeen knowledge bases. The author points out that when changes in the system configuration or load occur (e.g., the number of terminals or users increase), up to one hundred parameters must be checked and adjusted accordingly. Overall, a very brief look at the prototype is given, with most of the discussion centering upon the capabilities and advantages of the expert system shell.

[TIMM/Tuner]

McCrone, J., "Alvey Shows a Defence Bias," Computing, September 5, 14 (1985).

Even though Britian's Alvey was intended to move research from the laboratory into commercial use, the program has now focussed on MoD (Ministry of Defence) needs. This news article covers some of the reasons why Alvey's software engineering program has become defense oriented. Major reasons suggested were the collapse of the Ada community's projects and NATO's push for the use of Ada. Four primary ipse (integrated project support environment) projects are discussed. The Eclipse project addresses an Ada ipse which will run on the VAX minicomputer. Aspect will port Ada to Unix software development tools. MDSE (Mascot Design Support Environment) is being extended to include an expert system that will aid software design and prototyping. Forest addresses the specifications stage of system development.

[Alvey Projects]

Meyer, B., "The Software Knowledge Base," pp. 158-165 in Eighth International Conference on Software Engineering, Imperial College, London, United Kingdom, August 28-30, 1985, IEEE Computer Society Press, Washington, D. C., 1985.

A knowledge-based approach versus a database approach to storing information about a software project is discussed. The project, referred to as Software Knowledge Base (SKB), aids with the storage of software components and their relationships. Design criteria and software relations and constraints are included. The author feels such a knowledge base could be used for all phases of the development life cycle (e.g., specifications, design, testing, project management). Also discusssed are follow-on research efforts that are currently underway at the University of California, Santa Barbara (e.g., concurrent development of an SKB in PROLOG and a relational database using INGRES).

[Project Management]

Pidgeon, C. W. and P. A. Freeman, "Development Concerns for a Software Design Quality Expert System," in Proceedings of the 22nd ACM/IEEE Design Automation Conference, Las Vegas, Nevada, June 23-26, 1985, IEEE Computer Society Press, Washington, D. C., 1985.

This paper reviews a number of articles regarding software design systems. It is a comprehensive study of the issues dealing with systems based on Module Interconnection and Program Design Languages (MIL/PDL). The author gives an example of interactions that take place between human designers and an expert system dealing with quality design.

[Design]

Ramamoorthy, C. V., V. Garg, and R. Aggarwal, "Environment Modeling and Activity Management in Genesis," pp. 2-9 in Proceedings SOFTFAIR II: A Second Conference on Software Development Tools, Techniques, and Alternatives, San Francisco, California, December 2-5, 1985, IEEE Computer Society Press, Washington, D. C., 1985.

Genesis is a Unix tool being developed by the University of California, Berkeley researchers to support software development. It is primarily a knowledge based resource and activities management system. Its knowledge base contains an entity-relation-attribute model that is extended by rules about software resources and the development process. The tool consists of the following components: a specification language; a resource extractor (to provide traceability of resources between requirements and the rest of the system); a resource manager (to manage entities); and an activity manager (to check for inconsistency and to coordinate the work of multiple programmers). Research on this project has been in process since 1983. Several planned and possible enhancements are listed.

[GENESIS - Management]

Rawlings, T. L., "A Technological Approach to Automating Software Maintenance," pp. 147-149 in Proceedings of the First Software Maintenance Workshop, 1983, IEEE, New York, New York, 1984.

The author says much software maintenance could be eliminated with automatic program generators where changes in specifications just mean system regeneration. From user specifications, a software manufacturing system (DARTS) generates source code for all computers in a distributed system. The author feels much of the problem with software maintenance lies in capturing and communicating knowledge about the software system. Using a knowledge based system to capture such information is suggested. Since DARTS captures the programmer's expertise, users can change specifications and regenerate the program without assistance from the original developer.

[DARTS - Maintenance]

Rich, C., "Artificial Intelligence and Software Engineering: The Programmer's Apprentice Project," pp. 29 in Proceedings of the 1984 Annual Conference of the Association for Computing Machinery: The Fifth Generation Challenge, San Francisco, California, October 8-10, 1984, Association for Computing Machinery, Inc., New York, New York, 1984.

Rich presents a briefing on the status of the Programmer's Apprentice Project at MIT. The goals are presented: to develop an intelligent assistant for programmers. Information about the application domain is provided: how programmers analyze, modify, verify, document, etc. the programming process. Plans for a new program editor and its capabilities are discussed; the new editor will allow many logical changes in a program to be achieved by one command.

[PROGRAMMER'S APPRENTICE - Analysis/Design]

Rokey, M., "The Dataflow Architecture: A Suitable Base for the Implementation of Expert Systems," Computer Architecture News, 13(4), 8-14 (1985).

The author proposes the dataflow model of architecture over the conventional Von Neumann style for building expert systems. One major advantage listed is the inherent parallelism in the dataflow model leading to more efficient rule-searching. Furthermore, the problems of incremental change can be removed in this model. The paper suggests that it may be worthwhile to build systems with such a model.

[Design]

Ruth, G. R., "PROTOSYSTEM I - An Automatic Programming System," pp. 215-221 in C. Rich and R. C. Waters (eds.), Readings in Artificial Intelligence and Software Engineering, Morgan Kaufman Publishers, Inc., Los Altos, California, 1986.

The paper discusses a research project underway at MIT. The PROTOSYSTEM I project has the goal of taking user specifications, automatically designing the program, and generating the code. To date, only the PL/1 and JCL code generating modules have been developed.

[Code Generation]

Schindler, Jr., P. E., "An Intelligent Way to Develop Software," InformationWEEK, 71, 17 (1986).

This article contends that knowledge-based systems can be employed in traditional software development environments. Such systems can retain critical design information in the early stages of a project for use in later stages. The article also indicates IBM's recent interest in AI applications. This is evidenced by IBM's introduction of a knowledge-based COBOL structuring tool and expert system shells for VM and MVS systems.

[AI for Software Development]

Stephens, M. and K. Whitehead, "The Analyst - A Workstation for Analysis and Design," pp. 364-369 in Eighth International Conference on Software Engineering, Imperial College, London, United Kingdom, August 28-30, 1985, IEEE Computer Society Press, Washington, D. C., 1985.

Requirements analysis (CORE) and design (MASCOT) methods are supported via expert systems and knowledge base techniques in a personal workstation, ANALYST. The graphics and windowing capabilities are written in PASCAL; the rule-based methods, and storage and retrieval of application information are written in PROLOG. The authors feel that by using a knowledge-based approach, new rules can be added as needed and new methods can be added to support other phases of the software development life cycle. Shortcomings of the system's performance as judged against human experts are cited, as are potential system enhancements.

[ANALYST - Analysis/Specs/Design]

Studer, R., "Knowledge-Based Software Engineering Environments,"
Computer Physics Communication, 38(2), 277-287 (1985).

A method is proposed for controlling and managing communication between team members working on a project that is organized in a decentralized manner. A distributed knowledge-based software engineering environment (DSEE), is used to collect and distribute information to team members. The knowledge base contains four types of information: objects and their relationships in a data base; structure and responsibility of the project team; description of each tool provided by the system; and team member skills and experience levels, as well as team function. The Temporal Hierarchical Data Model with Petri Net concepts (THM-Net) has been chosen as the architectural model for the system.

[DSEE - Project Management]

Sussman, G. J., "Intelligent Support for the Engineering of Software," pp. 397-399 in Eighth International Conference on Software Engineering, Imperial College, London, United Kingdom, August 28-30, 1985, IEEE Computer Society Press, Washington, D. C., 1985.

A short article that suggests other branches of engineering (e.g., electrical engineering) may be able to contribute to the process of software engineering. The problems of debugging are cited. AI research in areas of formulating theoretical constructs as computational algorithms for software development is suggested. The author notes value in the LISP-family of tools in terms of recursion and manipulation. He feels such "flexibility" could support rapid prototyping.

[Aid Debugging]

Wolfe, A., "Software Productivity Moves Upstream," Electronics, 58(12), 80-86 (1986).

The article focusses on the work being done in the area of automating the software development life cycle, especially those phases "upstream" from code generation. Research endeavors are discussed: MCC's LEONARDO (design phase); the Software Engineering Institute's SOFTWARE FACTORY (all phases); IBM/Japan's PROMPTER (code generation); Lockheed's work in low-level design; and TRW's work on design tools and cost models. The author says that testing and maintenance are major issues that must be addressed. With NASA and SDI efforts increasing, the importance of quality assurance and testing to detect and correct all possible errors is paramount. With automated design and documentation, maintenance will improve.

[Current Research]

13.2 PROTOTYPING

"AI Environment Speeds Software Development," Systems and Software, 3(8), 111-118 (1984).

This paper discusses the advantages in using the Symbolics 3600 environment for software development and rapid prototyping.

Beregi, W. E., "Architecture Prototyping in the Software Engineering Environment," IBM Systems Journal, 23(1), 4-17 (1984).

The author examines various defects in present day software methodology. A disciplined approach, utilizing formal specification techniques, rapid prototyping, and static and dynamic behavior analysis techniques to verify system expectations is presented.

Bottom, J. S., A. D. Bernard, and K. W. Anderson, "Application Prototyping with Microcomputer Database Managers," pp. 60-73 in Proceedings of the Office Automation Society International Conference and Workshop for Office Professionals, September 3-6, 1985, San Francisco, California, Office Automation International Society, 1985.

The paper details the various tools that are essential for rapid prototyping. Various commercially available microcomputer database packages are compared in terms of ease of use and power.

Carey, T. T. and R. E. A. Mason, "Information System Prototyping: Techniques, Tools, and Methodologies," INFOR, 21(3), 176-190 (1983).

The paper reviews prototyping techniques in use. A number of prototyping tools, their techniques and methodologies are discussed.

Gremillion, L. L. and P. Pyburn, "Breaking the Systems Development Bottleneck," Harvard Business Review, 61(2), 130-137 (1983).

Three alternative approaches to the traditional life-cycle approach are presented. The purpose of the proposed methods is to get the user involved in the process of software development. Criteria for selecting the appropriate development strategy are discussed.

Johnson, J. R., "A Prototypical Success Story," Datamation, 29(11), 251-256 (1983).

This article discusses various levels of prototyping. The author presents development situations where prototyping and fourth generation languages yield best results.

Loomis M. E. S. and T. P. Loomis, "Prototyping and Artificial Intelligence," pp 65-73 in M. E. Lipp (ed.) Prototyping: State of the Art Report, Pergamon Infotech, Maidenhead, Berkshire, England, 1986.

In this paper, the advantages and disadvantages of applying artificial intelligence techniques for rapid prototyping are discussed.

Mason, R. E. A., T. T. Carey, and A. Benjamin, "A Tool for Information Systems Prototyping," ACM Sigsoft Software Engineering Notes, 7(5), 120-125 (1982).

This paper discusses an "architecture - based" methodology, where a prototype is developed using interactive scenarios. ACT/1 is a development tool specifically designed for this purpose. Its usage as a product specification and production tool are examined.

Naumann, J. D., "Prototyping: The New Paradigm for Systems Development," MIS Quarterly, 6(3), 29-44 (1982).

This article discusses various principles underlying prototyping. Of interest is a graphic cost comparison between the life-cycle and prototyping approach. Examples of projects developed by prototyping are presented.

Spies, P. B., "Designing Systems for Users," Library Hi Tech, 1(1), 75-84 (1983).

Design errors account for more than fifty percent of the overall development cost. According to the author, this is the motivation for rapid prototyping. Examples of success with the prototyping approach are cited.

Wess, B. P. Jr., "Artificial Intelligence Techniques Speed Software Development," Mini-Micro Systems, 17(11), 127-136 (1984).

This paper outlines the ways in which Artificial Intelligence techniques (e.g., Prolog) were used in the development of a commercial product.

APPENDIX A: SOFTMAN SCREENS

The run-time version of the SOFTMAN system is resident on a 5 1/4" floppy diskette. A "SOFTMAN" command will load the program. After entering the current date, the user interacts with the system via a "Main Menu." By positioning the arrow on the same line as the desired selection and pressing "Enter," a menu selection is made.

Upon choosing an option, the user must respond to a series of questions. Responses are primarily in the form of numeric data. The number and nature of the questions vary with the option chosen.

Once the series of questions has been completed, or the operation (e.g., file deletion) accomplished, the "Main Menu" is again displayed. When no further options are desired, the user can return to the "Main Menu," place the arrow on the same line as the "Exit" option, and SOFTMAN will return the user to the operating system.

Please enter today's
date as dd-mm-yy : 4-2-88

Please use DIGITS only for the date.

Example : 5-2-87
for 5th of February, 1987

MAIN MENU

PLEASE CHOOSE A COMMAND

↓ down
↑ up

■ New Project
Consult Old Project
Complete a Project
Directory of Projects
Delete a Project
Set/Change Date
Change Project Path Specifier
Retrieve an Old Consultation
Retrieve an Update
Update metrics data
Exit (SOFTMAN System)

<RETURN> makes selection

NEW PROJECT

Please enter a name for the project : TEST2

The name should NOT begin with a digit
It should have maximum 8 characters and
no spaces in between letters.

Enter a 0 to exit now.
Press <Enter> to see directory of projects.

COMMENT WINDOW

NEW PROJECT

MARY EMRICH

Please enter last name of the manager :

The name can be of 20 characters maximum.

COMMENT WINDOW

NEW PROJECT

Please choose a language type you will use :

1. Pascal, Fortran, Cobol etc.
2. Ada etc.
3. 4GL like ORACLE etc.

COMMENT WINDOW

NEW PROJECT

Please enter the estimated code size of the project

Choose one of the following by its number :

1. Small (less than 2500 lines)
2. Medium (2500 - 20000 lines)
3. Large (more than 20000 lines)

COMMENT WINDOW

NEW PROJECT

Please enter the number of people doing active coding : 12

COMMENT WINDOW

NEW PROJECT

Please enter coding start date as (dd-mm-yy):

1-1-87

COMMENT WINDOW

NEW PROJECT

Please enter the estimated time required to finish the project. (in months) 24

COMMENT WINDOW

MAIN MENU

PLEASE CHOOSE A COMMAND

↑ down
↑ up

↑

New Project
Consult Old Project
Complete a Project
Directory of Projects
Delete a Project
Set/Change Date
Change Project Path Specifier
Retrieve an Old Consultation
Retrieve an Update
Update metrics data
Exit (SOFTMAN System)

<RETURN> makes selection

CHOOSING A PROJECT

Please enter the name of the project TEST2

If you do not remember the exact name then press Return (Enter) to see the directory of projects.

Enter a 0 to exit now.

COMMENT WINDOW

PROJECT ESTIMATES

```

Name of the project      : TEST2
Manager of the project   : MARY EMRIC
Language category        : 1
Estimated size of project : MEDIUM
  
```

If you want to change any estimate then
select it from the menu

```

Number of active coders      : 12
Date of start of coding      : 1-1-87
Estimated months to complete : 24
Escape to earlier menu
Continue with Consultation
  
```

↑ down
↑ up

<RETURN> makes selection
COMMENT WINDOW

MAIN MENU

PLEASE CHOOSE A COMMAND

↓ down
↑ up

New Project
Consult Old Project
Complete a Project
Directory of Projects
Delete a Project
Set/Change Date
Change Project Path Specifier
Retrieve an Old Consultation
Retrieve an Update
Update metrics data
Exit (SOFTMAN System)

<RETURN> makes selection

DIRECTORY OF PROJECTS

TEST3

TEST2

TEST1

COMMENT WINDOW

Press any key to continue

MAIN MENU

PLEASE CHOOSE A COMMAND

New Project
Consult Old Project
Complete a Project
Directory of Projects
Delete a Project
Set/Change Date
Change Project Path Specifier
Retrieve an Old Consultation
Retrieve an Update
Update metrics data
Exit (SOFTMAN System)



↓ down
↑ up

<RETURN> makes selection

DELETION OF PROJECTS

Enter the name of the project to delete (without extension) : TEST3

Press Return (Enter) to see directory
Enter a 0 to exit now.

BE SURE YOU WANT TO DELETE THIS PROJECT
BECAUSE ALL CONSULTATIONS AND PARAMETERS
OF THE PROJECT WILL BE DELETED.

COMMENT WINDOW

Are you sure (y/n) :

Yes

No



MAIN MENU

PLEASE CHOOSE A COMMAND

↓ down
↑ up

New Project
Consult Old Project
Complete a Project
Directory of Projects
Delete a Project
Set/Change Date
Change Project Path Specifier
Retrieve an Old Consultation
Retrieve an Update
Update metrics data
Exit (SOFTMAN System)



<RETURN> makes selection

Please enter today's
date as dd-mm-yy : 4-2-88

Please use DIGITS only for the date.

Example : 5-2-87

For 5th of February, 1987

MAIN MENU

PLEASE CHOOSE A COMMAND

↓ down
↑ up

New Project
Consult Old Project
Complete a Project
Directory of Projects
Delete a Project
Set/Change Date
Change Project Path Specifier
Retrieve an Old Consultation
Retrieve an Update
Update metrics data
Exit (SOFTMAN System)

<RETURN> makes selection

CHANGING PATH SPECIFIER

The current path specifier is : A:\

Please enter new path specifier :
Enter a 0 if you want to retain
existing path specifier.

COMMENT WINDOW

MAIN MENU

PLEASE CHOOSE A COMMAND

New Project
Consult Old Project
Complete a Project
Directory of Projects
Delete a Project
Set/Change Date
Change Project Path Specifier
Retrieval an Old Consultation
Retrieval an Update
Update metrics data
Exit (SOFTMAN System)



↓ down
↑ up

<RETURN> makes selection

CHOOSING A PROJECT

TEST1

Please enter the name of the project

If you do not remember the exact name
then press Return (Enter) to see the
directory of projects.

Enter a 0 to exit now.

COMMENT WINDOW

LIST OF CONSULTATIONS

Choose one consultation to retrieve

Stage	Number	Date	
1	1	2-2-88	↓ down
1	1	2-2-88	↑ up
3	1	3-2-88	
None selected			

<RETURN> makes selection

COMMENT WINDOW

DETAILS OF CONSULTATION

Project name	:	TEST1
Manager name	:	AGARWAL
Language category	:	1
Estimated code size	:	MEDIUM
Number of coders	:	10
Date of start of coding	:	1-1-86
Estimated months to complete	:	15
Revisions made to estimates	:	Yes
Stage :	1	Date of consultation : 2-2-88
Programmer hours	:	10
No. of statements	:	10
No. of runs	:	10
No. of changes	:	10
CPU time	:	10
Time schedule	:	Overdue
Code size	:	O.K.
Productivity	:	Low
Errors	:	Too Many
CPU time	:	Too Much
Computer runs	:	Too Many

COMMENT WINDOW

Press any key to continue

MAIN MENU

PLEASE CHOOSE A COMMAND

New Project
Consult Old Project
Complete a Project
Directory of Projects
Delete a Project
Set/Change Date
Change Project Path Specifier
Retrieve an Old Consultation
Retrieve an Update
Update metrics data
Exit (SOFTMAN System)



↓ down
↑ up

<RETURN> makes selection

CHOOSING A PROJECT

TEST1

Please enter the name of the project

If you do not remember the exact name
then press Return (Enter) to see the
directory of projects.

Enter a 0 to exit now.

COMMENT WINDOW

LIST OF UPDATES

Choose one update to retrieve

Number	Date	
1	2-2-88	↓ down
2	2-2-88	↑ up
3	2-2-88	
None selected		

<RETURN> makes selection

COMMENT WINDOW

DETAILS OF UPDATE

Date of update : 2-2-88

Before update

After update

Project name	:	TEST1
Manager name	:	AGARWAL
Language category	:	1
Estimated code size	:	MEDIUM
Number of coders	:	10
Date of start of coding	:	1-1-87
Estimated months to complete	:	13

TEST1
AGARWAL
1
MEDIUM
10
1-1-87
5

COMMENT WINDOW
Press any key to continue

MAIN MENU

PLEASE CHOOSE A COMMAND

New Project
Consult Old Project
Complete a Project
Directory of Projects
Delete a Project
Set/Change Date
Change Project Path Specifier
Retrieve an Old Consultation
Retrieve an Update
Update metrics data
Exit (SOFTMAN System)



↓ down
↑ up

<RETURN> makes selection

Please enter access password here

You will have a maximum of 3 attempts
to enter the correct password. If you
fail then the system will be locked and
an alarm will be sounded

If you want to exit now then press Escape

Attempt number : 1 Password :

COMMENT WINDOW

Metrics Updated

Press any key to continue

MAIN MENU

PLEASE CHOOSE A COMMAND

↓ down
↑ up

New Project
Consult Old Project
Complete a Project
Directory of Projects
Delete a Project
Set/Change Date
Change Project Path Specifier
Retrieve an Old Consultation
Retrieve an Update
Update metrics data
Exit (SOFTMAN System)

<RETURN> makes selection

APPENDIX B: SAMPLE CONSULTATION

A sample consultation is run in SOFTMAN by choosing the "Consult" option from the main menu and entering the project name. Then, the user selects the coding stage and provides numeric values for the five input development parameters. SOFTMAN calculates metric values and responds with a bar menu listing detected anomalies.

In some cases, the user can analyze each anomaly. By making a selection from the menu, SOFTMAN will suggest solutions based on the type of anomaly. In some cases, the user is asked to revise initial project parameters. When the consultation is finished, it can be saved for later viewing by choosing the "Save" option.

MAIN MENU

PLEASE CHOOSE A COMMAND

↓ down
↑ up

↑
New Project
Consult Old Project
Complete a Project
Directory of Projects
Delete a Project
Set/Change Date
Change Project Path Specifier
Retrieve an Old Consultation
Retrieve an Update
Update metrics data
Exit (SOFTMAN System)

<RETURN> makes selection

.. . . .

CHOOSING A PROJECT

Please enter the name of the project test1

If you do not remember the exact name
then press Return (Enter) to see the
directory of projects.

Enter a 0 to exit now.

COMMENT WINDOW

PROJECT ESTIMATES

```

Name of the project      : TEST1
Manager of the project   : AGARWAL
Language category       : 1
Estimated size of project : MEDIUM
  
```

The estimates of this project have been revised before

If you want to change any estimate then
select it from the menu

```

Number of active coders      : 10
Date of start of coding      : 1-1-86
Estimated months to complete : 15
Escape to earlier menu
Continue with Consultation
  
```

↓ down
↑ up

<RETURN> makes selection
COMMENT WINDOW

CODING STAGE

Please select a choice
for the coding stage

1. Early coding.

2. Middle coding.

3. Late coding.
- ↓

↑

↑
- down

up

<RETURN> makes selection

COMMENT WINDOW

Now you will be asked some questions about the progress of the project dealing with code lines, computer runs etc. If you do not have this data then leave the program at this stage and collect the data first

For any question you can press <Enter> on a blank entry to get more help than is displayed on screen

Hit Esc to exit or any other key to continue

CONSULTATION INPUTS

How many programmer hours have
been spent so far in the coding ?

This is the total number of hours that
the programmers have spent in the coding
process.

COMMENT WINDOW

Press <Enter> for more help

This is a more detailed help for the number of programmer hours spent so far in the development.

Programmer hours are defined as the time spent in actually coding, debugging or testing the programs. Time spent for writing documentation is NOT included here.

On the average it is noted that in a 8-hour working day a good programmer spends at least 6 hours doing worthwhile programming.

COMMENT WINDOW

Press any key to return

CONSULTATION INPUTS

How many programmer hours have
been spent so far in the coding ? 1000

This is the total number of hours that
the programmers have spent in the coding
process.

-----COMMENT WINDOW-----

Press <Enter> for more help

CONSULTATION INPUTS

How many statements of executable
code have been generated so far ? 100

This is the total number of lines of
executable code generated so far. This
DOES NOT include documentation of code
or comment lines.

-----COMMENT WINDOW-----

Press <Enter> for more help

CONSULTATION INPUTS

How many computer runs have been
made in the coding ? 100

These are the total number of computer
runs (attempted executions and not any
compile attempts) made so far.

===== COMMENT WINDOW =====

Press <Enter> for more help

CONSULTATION INPUTS

How many software changes to code 1000
have been made so far ?

This is the total number of changes made
in the software while coding was being
done (from the first day of coding). A
change is defined as any error change,
removal (syntax and semantics), logical
structure change or parameters change.

COMMENT WINDOW

Press <Enter> for more help

CONSULTATION INPUTS

How much computer (CPU) time (secs.) 1000
has been spent so far in the coding? ?

This is the total amount of CPU time
in all the computer runs performed by
the programmers.

=====COMMENT WINDOW=====

Press <Enter> for more help

LIST OF PROBLEMS AT THIS STAGE

Please choose one to investigate

Time Schedule Overdue
Code Length O.K.
Productivity Low
Errors Too Many
CPU Time Too Much
Computer Runs Too Many
Save and Quit
Quit without Saving

↓ down
↑ up

<RETURN> makes selection

Actual:

Value: 763 (169%)

Optimal:

Value: 450 (100%)

Today's date indicates that you have
crossed the estimated time you had set
for the completion of the project.

COMMENT WINDOW

Do you want help in trying to solve this problem :

Yes

No



HELP FOR OVERDUE PROJECT SCHEDULE

The only way to solve this problem is to revise the estimate for the time required for project completion given by the user when the project parameters were entered

The current estimate is :

15

Do you want to revise this estimate :

Yes

No



COMMENT WINDOW

HELP FOR OVERDUE PROJECT SCHEDULE

The only way to solve this problem is to revise the estimate for the time required for project completion given by the user when the project parameters were entered

The current estimate is :	15	
Do you want to revise this estimate :	Yes	No
Please enter new estimate :	24	

COMMENT WINDOW

LIST OF PROBLEMS AT THIS STAGE

Please choose one to investigate

Time Schedule Overdue
Code Length O.K.
Productivity Low
Errors Too Many
CPU Time Too Much
Computer Runs Too Many
Save and Quit
Quit without Saving

↓ down
↑ up

<RETURN> makes selection

There is no error in this parameter as
its value is o.k.

Please choose another number from the
error list.

COMMENT WINDOW
Press any key to continue

LIST OF PROBLEMS AT THIS STAGE

Please choose one to investigate

Time Schedule Overdue
Code Length O.K.
▲ Productivity Low
Errors Too Many
CPU Time Too Much
Computer Runs Too Many
Save and Quit
Quit without Saving

↓ down
↑ up

<RETURN> makes selection

Actual:

Value: 100 (0%)

Optimal:

Value: 250000 (100%)

Assuming standards that an average programmer produces 1000 lines of code in a month, the metrics indicate that the project team is not within this productivity range.

COMMENT WINDOW
Do you want help in trying to solve this problem :

Yes

No



LIST OF PROBLEMS AT THIS STAGE

Please choose one to investigate

Time Schedule Overdue
Code Length O.K.
Productivity Low
▲ Errors Too Many
CPU Time Too Much
Computer Runs Too Many
Save and Quit
Quit without Saving

↓ down
↑ up

<RETURN> makes selection

Actual:

Value: 10 (5000%)

Optimal:

Value: 0.2 (100%)

The code that is being generated for the project is error-prone since the number of changes indicated are higher than the standards.

COMMENT WINDOW

Do you want help in trying to solve this problem : Yes No

LIST OF PROBLEMS AT THIS STAGE

Please choose one to investigate

Time Schedule Overdue	↓	down
Code Length O.K.	↑	up
Productivity Low		
Errors Too Many		
▲ CPU Time Too Much		
Computer Runs Too Many		
Save and Quit		
Quit without Saving		

<RETURN> makes selection

Actual: Value: 100 (333%)
Optimal: Value: 30 (100%)

The CPU time indicates that there are flaws in the system which is making it take a long time to execute. A lot of costly computer time is being wasted.

COMMENT WINDOW		
Do you want help in trying to solve this problem :	Yes	No

LIST OF PROBLEMS AT THIS STAGE

Please choose one to investigate

Time Schedule Overdue	↓	down
Code Length O.K.	↑	up
Productivity Low		
Errors Too Many		
CPU Time Too Much		
Computer Runs Too Many		
Save and Quit		
Quit without Saving		

<RETURN> makes selection

Actual:

Value: 1 (1000%)

Optimal:

Value: 0.1 (100%)

The project team is making too many computer runs. It is not good practice to have so many runs.

COMMENT WINDOW

Do you want help in trying to solve this problem : Yes No

LIST OF PROBLEMS AT THIS STAGE

Please choose one to investigate

↓ down
↑ up

Time Schedule Overdue
Code Length O.K.
Productivity Low
Errors Too Many
CPU Time Too Much
Computer Runs Too Many
Save and Quit
Quit without Saving .

<RETURN> makes selection

IF YOU EXIT WITHOUT SAVING THEN ALL
THE DATA INPUT SO FAR IN THIS ROUND
OF CONSULTATION WILL BE LOST.

Are you sure you want to do this : COMMENT WINDOW Yes No 

Saving consultation

APPENDIX C: GLOSSARY OF RELEVANT TERMS

A

Abstract Data Types

Data structure or type whose implementation is transparent to the user; defined by a set of operations that can be performed on it.

AIRMICS

The Army Institute for Research in Management Information, Communications and Computer Science, Atlanta, Georgia.

Arrowsmith-P

A software management project done as a Masters thesis, the University of Maryland, under Dr. V. Basili.

C

Coding Phase

The SDLC phase in which the design is converted to computer programs.

Coding Stage (SOFTMAN)

The coding phase is generally divided into early, middle, and late stages.

D

Data Flow Diagrams

Software design representation schemes based on data flow between the different modules.

Design Phase

The SDLC phase in which the high-level design of the system is developed.

Dynamic Metrics

Metrics that are continually updated.

E

Environment-Dependent Data

Data determined by the software and hardware configuration; used to drive the Softman system.

Expert System Development Tool

Software which helps in building expert systems by providing intelligent editors, knowledge representation facilities, database links, etc.

F

Flowcharts

Software design representation mechanisms based on the program's logic flow.

Fourth Generation Languages

Non-procedural languages which have procedural components (e.g., data structures and procedures) incorporated, thus relieving the user of such details.

Frame

A collection of slots which represent events, objects, and their attributes.

Frame-Based Approach

A knowledge-based system wherein information is stored as frames.

Functional Modules

Distinct, and independent routines; each performing a separate function (e.g., report generator and input output routine).

G

GLITTER

Research project at the University of Oregon, Eugene; based on TI.

H

HEURISTICS

Rules-of-thumb that are used for decision making by intelligent systems.

HIPO Charts

Hierarchical Input/Output Diagrams; data flow diagrams.

How and Why

Expert system facilities commonly used to explain the system's line of reasoning.

I

Implementation and Verification Phase

The SDLC phase in which the system is put on-line and extensively tested.

Inference Engine

Component of an expert system architecture which performs the match, select, and execute cycle.

Input Parameters

Quantitative measurements about software characteristics (e.g., number of lines of code and number of errors).

Intelligence/Compiler

An expert system development tool that is marketed by IntelligenceWare, Los Angeles, California.

K

KBPA

Knowledge-Based Programming Assistant, University of Illinois, at Urbana-Champaign.

KBSA

Knowledge-Based Software Assistant, Rome Air Development Center, Rome, New York.

M

Maintenance Phase

The SDLC phase in which the software is continuously upgraded and modified to meet user needs and to correct bugs which surface during usage.

Method (GLITTER)

A formal mechanism for representing decision-making strategies in GLITTER.

Metric

A quantitative software measure used to compare Actual and Optimal values of code characteristics (e.g., errors and lines of code).

Q

Output Parameters

SOFTMAN's qualitative decisions regarding the project (e.g., time schedule and productivity).

P

PA

Programmer's Apprentice, MIT.

Plan Calculus

A method used by PA to represent programs and programming knowledge.

Project Parameters

Estimates made by a manager about a particular project when it is first run in SOFTMAN.

Q

Query Language

Language used to obtain information from a database.

R

RADC	Rome Air Development Center, Rome, New York.
Prototyping	Software development process wherein the program is iteratively refined until it meets complete specifications.
Requirements Phase	The first phase of the SDLC in which system requirements are determined at a very high level.
Rule-Based Approach	A knowledge-based system wherein information is stored as If-Then structures.

S

SDLC	Software Development Life Cycle.
Selection-Rule (GLITTER)	Mechanism used in GLITTER to resolve conflicts between two goals which are simultaneously ready for firing.
Softman	The Software Manager's Apprentice; project currently underway on the Oak Ridge Reservation.
Software Crisis	Phenomenon caused by the failure of conventional software development methods to meet software needs.
Software Development Life Cycle	Process of developing a software system from conception through implementation and maintenance.
Software Management	Process of co-ordinating and controlling a software development team's activities in terms of budget, user needs, time constraints, etc.

Specification Languages

VHLLs which allow program expression in terms of specifications; progressively refining them to lower abstraction levels.

Specifications Phase

The SDLC phase in which requirements are expressed in a formal language.

T

Temporal Changes

Changes over time.

Transformational Implementation (TI)

Software development paradigm which starts with a high-level specification of the program and applies various transformations until the program is generated.

V

VHLL

Very High Level Language.

W

Waterfall Method

Traditional method of developing software; entails sequential stages.