An Evaluation of Expert Systems for Software Engineering Management

CONNIE LOGGIA RAMSEY, MEMBER, IEEE, AND VICTOR R. BASILI, SENIOR MEMBER, IEEE

Abstract—**Although the field of software engineering is relatively new, it can benefit from the use of expert systems because of the ability to learn from them. We believe that a major limitation to building expert systems for software engineering is the fact that much of the knowledge in this field is not well understood yet. Therefore, the development of expert systems in this field must be considered exploratory. This project focused on the development of four separate, prototype expert systems to aid in software engineering management. Given the values for certain metrics, these systems provide interpretations which explain any abnormal patterns of these values during the development of a software project. The four expert systems, which solve the same problem, were built using two different approaches to knowledge acquisition, a bottom-up approach and a top-down approach, and two different expert system methods, rule-based deduction and frame-based abduction. In a comparison to see which methods might better suit the needs of this field, it was found that the bottom-up approach led to better results than did the top-down approach, and the rule-based deduction systems using simple rules provided more complete and correct solutions than did the frame-based abduction systems.**

Index Terms—**Expert systems, software development, software engineering management.**

I. Introduction

THE importance of expert systems is growing in industrial, medical, scientific, and other fields. Several major reasons for this are: 1) the necessity of handling an overwhelming amount of knowledge in these areas, 2) the potential of expert systems to train new experts, 3) the potential to learn more about a field while organizing knowledge for the development of expert systems, 4) cost reductions sometimes provided by expert systems, and 5) the desire to capture corporate knowledge so it is not lost as personnel changes.

Although the field of software engineering is still relatively new, it can certainly benefit from the use of expert systems because of the ability to learn from them. The development of any expert system requires organized knowledge; therefore, the knowledge engineer can learn more about the field of software engineering as he is forced to develop, understand, and organize relationships between various pieces of knowledge.

On another level, the expert systems in this field can be used to train and help people, including software managers. They can contain general software engineering principles as well as a history of information from a particular software development environment which can be particularly helpful to inexperienced managers and developers.

Since software engineering is still such a new field with much of its knowledge unclear, expert systems developed in this field must be considered exploratory prototypes. This project focused on software engineering management. A first attempt was made at creating and systematically analyzing and comparing expert systems which intelligently relate software engineering project measurements and explanations of project behavior. This was an exploratory learning experience which has provided an initial baseline for future work [4], [29].

The high level goal of this project was to examine different approaches to expert system development for software engineering management and determine strengths and limits of the various approaches as they relate to the field. Some of the questions this study tried to answer were: 1) Are expert systems for software engineering management feasible at this time? 2) What methodology should be used for knowledge acquisition? 3) What type of expert system methodology best suits software engineering management? 4) Do the experts themselves agree on the information to be used? 5) Are certain software environments more suited for expert systems than others? 6) Are we ready to develop systems with environment-independent, general truths? 7) What information should be included in the system?

This paper will discuss the comparison of several prototype expert systems, collectively named ARROWSMITH-P.[1] Earlier versions of these expert systems are described in [3]. ARROWSMITH-P is intended to aid the manager of a software development project in an automated manner. The goal of these systems is to help detect and assess the problems which might occur during the coding and testing of a project as early as possible. The systems work as follows. First, it is determined whether or not a software project is following normal development

Manuscript received November 10, 1986; revised January 31, 1989. This work was supported in part by the National Aeronautics and Space Administration under Grant NSG-5123 to the University of Maryland. Computer support was provided in part by the Computer Science Center of the University of Maryland.

C. L. Ramsey is with the Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory, Washington, DC 20375.

V. R. Basili is with the Institute for Advanced Computer Studies and the Department of Computer Science, University of Maryland, College Park, MD 20742.

IEEE Log Number 8927386.

[1] Martin Arrowsmith, created by Sinclair Lewis in the novel *Arrowsmith*, was in constant search of truth in scientific fields. The "P" stands for Prototype.

patterns by comparing measures such as programmer hours per line of source code against historical, environment-specific baselines of such measures. Then, the "manifestations" detected by this comparison, such as an abnormally high rate of programmer hours per line of source code, serve as input to each expert system, and each system attempts to determine the reasons, such as *high complexity* or *low productivity*, for any abnormal software development patterns. Early detection of potential problems can provide invaluable assistance to the manager of a software development project. These expert systems should be updated as the environment changes and as more is learned in the field of software engineering.

The rest of this paper is organized as follows. Section II provides a brief overview of the underlying methodologies used to build the expert systems discussed in this paper. The knowledge representation and inference techniques of the methodologies are presented here. Section III describes aspects of the software engineering development environment used for this study. Section IV details the implementations of ARROWSMITH-P, i.e., how the different approaches were utilized to build the expert systems. In Section V, some of the technical issues and problems associated with this process are discussed. Section VI furnishes the details for the evaluation of the expert systems. Section VII then discusses results and conclusions from the development and testing of the expert systems. Finally, Section VIII discusses current and future research needs.

II. Background on Expert Systems

In general, an expert system consists of two basic components, a domain-specific knowledge base and a domain-independent inference mechanism. The knowledge base consists of data structures which represent general problem-solving information for some application area. The inference mechanism uses the information in the knowledge base along with problem-specific input data to generate useful information about a specific case.

The set of expert systems in ARROWSMITH-P was constructed using KMS [25], an experimental domain-independent expert system generator which can be used to build rule-based, frame-based and Bayesian systems. The ARROWSMITH-P systems were built using two different methods: rule-based deduction and frame-based abduction. These two methods are briefly described below.

A. Rule-Based Deduction

A common method for expert systems is rule-based deduction. In this approach, domain-specific problem-solving knowledge is represented in rules which are basically of the form:

"IF <antecedents> THEN <consequents>",

although the exact syntax used may be quite different (e.g., PROLOG). If the antecedents of such a rule are determined to be true, then it logically follows that the consequents are also true. Note that these rules are not branching points in a program, but are nonprocedural statements of fact.

The inference mechanism consists of a rule interpreter which, when given a specific set of problem features, determines applicable rules and applies them in some specified order to reach conclusions about the case at hand. Rule-based deduction can be performed in a variety of ways, and rules can be chained together to make multiple-step deductions. (For a fuller description, see [13].) In addition, in many systems one can attach "certainty factors" to rules to capture probabilistic information, and a variety of mechanisms can be used to propagate certainty measures during problem solving. MYCIN [26] and PROSPECTOR [8] are two well-known examples of expert systems which incorporate rule-based deduction.

B. Frame-Based Abduction

Another important method for implementing expert systems is frame-based abduction. Here, the domain-specific problem-solving knowledge is represented in descriptive "frames" of information [15], and inference is typically based on hypothesize-and-test cycles which model human reasoning as follows. Given one or more initial problem features, the expert system generates a set of potential hypotheses or "causes" which can explain the problem features. These hypotheses are then tested by 1) the use of various procedures which measure their ability to account for the known features, and 2) the generation of new questions which will help to discriminate among the most likely hypotheses. This cycle is then repeated with the additional information acquired. This type of reasoning is used in diagnostic problem solving (see [22] for a review). INTERNIST [14], KMS.HT [25], [23], PIP [17], and IDT [27] are typical systems using frame-based abduction.

In order to simulate hypothesize-and-test reasoning, KMS employs a generalized set covering model in which there is a universe of all possible manifestations (symptoms) and a universe which contains all possible causes (disorders). For each possible cause, there is a set of manifestations which that cause can explain. Likewise, for each possible manifestation, there is a set of causes which could explain the manifestation. Given a diagnostic problem with a specific set of manifestations which are present, the inference mechanism finds all sets of causes with minimum cardinality[2] which could explain (cover) all of the manifestations. For a more detailed explanation of the theory underlying this approach and the problem-solving algorithms, see [23], [24], [16], and [18].

III. Background on Software Environment

The software which provided the data for this study was developed at the NASA Goddard Space Flight Center. This software development environment is homogeneous,

[2]Ockham's razor, which states that the simplest explanation is usually the correct one, together with the assumption of independence among causes motivate the requirement of minimum cardinality.

i.e., many similar projects are developed for the same application area. There has been a standard process model developed over the years; the methodology for development is similar across projects, and there is a great deal of reuse of code from prior projects.

The NASA Software Engineering Laboratory has been collecting reliable software project data such as programmer hours and lines of code for approximately fifteen years. The data used for the knowledge bases of the expert systems was chosen from this database of information because it was standard data for the environment and covered a great deal of the software life cycle phases being studied.

The experts who aided in knowledge acquisition were two managers who had successfully supervised software development in this environment for many years. They were also involved in the collection and analysis of data for prior projects and therefore understood the implications of the information in the database.

IV. Implementations

In this section, we will first present the methodology developed for building expert systems for software engineering management. Then we will discuss the actual implementations of ARROWSMITH-P.

A. Methodology

The following two methodologies of knowledge acquisition for constructing expert systems for software engineering management were developed. They can best be described as a bottom-up methodology and a top-down methodology. (An earlier version of the bottom-up reasoning was developed by Doerflinger and Basili [12].)

1) Bottom-Up Methodology: Given a homogeneous environment, it is possible to produce historical, environment-specific baselines of normalized metrics from the data of past software projects. Normalized metrics are derived by comparing variables such as programmer hours and lines of code against each other. This is done so influences such as the size of the individual project are factored out. The baseline for each metric is defined as the average value of that metric for the past projects at various discrete time intervals (such as early coding or acceptance testing). Only those metrics which exhibit baselines with reasonable standard deviations should be used; too little variety in the values of the measures proves uninteresting, while too much variety is not very meaningful. In addition, one ideally wants a relatively small number of meaningful metrics whose values are easily obtainable.

Next, experts can determine interpretations, such as *unstable specifications* or *good testing*, which would explain any significant deviation (more than one standard deviation less than or greater than the average) of a particular metric from the historical baseline. The deviation of some metric can be thought of as a manifestation or symptom which can be "diagnosed" as certain interpretations or causes. Furthermore, these relationships between interpretations and manifestations should be made time-line specific because, for example, an interpretation during early coding might not be valid during acceptance testing. In addition, measures to indicate how certain one is that the deviation of a particular metric has resulted from a particular interpretation can be included.

The approach, described above, can be classified as a bottom-up approach because it seems to go in the opposite direction of cause-and-effect. First the symptoms (deviant metric values) that something is abnormal are explored, and then the underlying interpretations or diagnoses of the abormalities are developed. This approach to knowledge acquisition is reasonable in a homogeneous environment because the metrics are homogeneous, and deviations are indicative that something is wrong. However, this approach contrasts with the development of expert systems in other fields, such as medicine, which typically use a top-down approach.

2) Top-Down Methodology: A top-down approach to knowledge acquisition can be similar to the bottom-up approach in that the same manifestations and causes can be used. However, it would first define the various interpretations or diagnoses and then indicate the metrics which would be likely to have abnormal values for each interpretation.

Using the top-down approach, the experts view the knowledge from a different perspective when defining the relationships that exist between the interpretations and manifestations. This approach can be seen as a more general approach than the bottom-up approach is to knowledge acquisition in the field of software engineering management. In the bottom-up methodology, the metrics are analyzed first and these are, by their nature, environment-specific. The focus is automatically limited to the specific environment. Conversely, in the top-down methodology, the experts think first of the causes or interpretations and then indicate the effects or likely metrics which would show deviant values if a certain interpretation existed. This generalizes the problem across environments somewhat because the emphasis seems to be switched to the interpretations which can be universal.

3) Using the Expert Systems: Once the expert systems have been developed, the input to each expert system would then consist of those metrics from a current project which deviate from a historical baseline of the same metrics at the same time of development for similar projects. The knowledge bases consist of information about various potential causes, such as *poor testing* or *unstable specifications*, for any abnormally high or low measures, and the expert system provides explanations for any abnormal software development patterns.

B. Actual Implementations

ARROWSMITH-P consists of four independent expert systems, one using a bottom-up approach to knowledge acquisition and rule-based deduction, a second using the

bottom-up approach and frame-based abduction, a third using a top-down approach to knowledge acquisition and rule-based deduction, and a fourth using the top-down approach and frame-based abduction.

The bottom-up methodology described above was based on previous research conducted on the NASA Goddard Space Flight Center Software Engineering Laboratory (SEL) environment [12]. Since the SEL environment is homogeneous, it was possible to produce historical, environment-specific baselines of normalized metrics from the highly reliable data of nine software projects. (See [7], [5], [6], [9], and [1] for fuller descriptions of the SEL environment.)

The bottom-up development was performed first, and nine metrics, derived from five variables, were chosen because they were standard data measurements for the environment and covered a great deal of the software life cycle phases being studied. They also proved satisfactory because they exhibited baselines with reasonable standard deviations. The metrics are displayed in Table I. These same metrics were later used during the top-down development to ensure consistency and to allow a comparative study to be performed. The time-line for the baselines was divided (after a slight modification) into the following five discrete intervals: early code, middle code, late code, systems test, and acceptance test.

The initial sets of interpretations and the relationships between the interpretations and the abnormal values of metrics were mainly derived from two experts who have had a great deal of experience in this field and particularly in the SEL environment. The experts were asked what they thought high and low values of metrics might mean, and the interpretations they suggested were used in the experiment [12]. During the bottom-up development of ARROWSMITH-P, mainly one of these experts modified the existing sets and made them time-line specific. In addition, measures to indicate how certain one is that the interpretation and the abnormal metric value are connected were included. During the top-down development, the same two experts were again asked to provide the relationships for all five time phases, and the intersection of their responses was used for the expert systems. Some of their other indicated relationships were used as well; when the experts did not agree on a relationship, we discussed the situation to understand the reasoning behind the relationship and to see how certain an expert felt about the relationship. The list of interpretations used and tested in the bottom-up and top-down expert systems is displayed in Table II. (Other interpretations were used as well, but these could not be tested. See [3] for the complete list.)

As stated previously, two different expert system methods were used to build the expert systems for this application in order to determine which method better suits the needs of this field. The two methods used were rule-based deduction and frame-based abduction which were described in Section II. In the rule-based systems, the rules are of the form "IF manifestations THEN interpretations," while in the frame-based systems, there is one frame (containing a list of manifestations) for each interpretation. Please note that these formats are independent of whether the relationships between manifestations and interpretations were defined using a bottom-up or a top-down approach to knowledge acquisition. The rule-based and frame-based systems which used the bottom-up approach were intentionally built to be as consistent with one another as possible. The causes and manifestations used were identical in both cases, as were the relationships between them. The same was true for the two expert systems which employed the top-down approach. However, the certainty factors attached to the rules and the measures of likelihood in the frames could not be directly translated to each other so some of these measures were omitted. For example, within the bottom-up approach we were relatively certain that an abnormally high value of computer time per software change is caused by *good, reliable code* so this was given a certainty factor of 0.75. However, if that particular metric appears abnormally high very infrequently and that particular interpretation is common, then we would not be able to state that *good, reliable code* generally results in an abnormally high value of computer time per software change. (For a discussion of similar problems see [21].) Fig. 1 shows a sample section of a rule-based and a frame-based knowledge base. Example sessions with the expert systems are provided in the Appendix.

TABLE I
METRICS USED IN EXPERT SYSTEM

- Computer Runs per Line of Source Code
- Computer Time per Line of Source Code
- Software Changes per Line of Source Code
- Programmer Hours per Line of Source Code
- Computer Time per Computer Run
- Software Changes per Computer Run
- Programmer Hours per Computer Run
- Computer Time per Software Change
- Programmer Hours per Software Change

TABLE II
INTERPRETATIONS USED IN EXPERT SYSTEM

Unstable Specifications
Low Productivity
High Productivity
High Complexity or Tough Problem
High Complexity or Compute Bound Algorithms Run or Tested
Low Complexity
Simple System
Error Prone Code
Good Solid and Reliable Code
Large Portion of Reused Code
Lots of Testing
Little Testing
Good Testing or Good Test Plan
Lack of Thorough Testing
Poor Testing Program
Changes Hard to Make
Loose Configuration Management or Unstructured Development
Tight Configuration Management or Control
Computer Problems or Inaccessibility or Environmental Constraints
Lots of Terminal Jockeys

V. Research Issues and Problems

The field of expert systems is relatively new, and therefore, the development process of expert systems still faces many problems. The selection of which method to use for building them is not generally clear, although an attempt has been made to provide guidelines for the selection of an appropriate method in [21]. Furthermore, most expert systems are shallow in nature and cannot handle temporal or spatial information well.

In addition to general problems, negative effects are compounded when the knowledge to be included in such systems is incomplete. The science of software engineering is not well-defined yet, and therefore many details about the relationships between various components are often unclear. The experts themselves may not even agree on the information used in the expert systems. As a result, the knowledge base of any expert system developed in this field is particularly exploratory and prototypical in nature. This is in contrast to expert systems developed in established fields such as medicine where the information contained in the knowledge base is based on many years of experience.

Due to the uncertainty of the data in the knowledge base for a field such as software engineering, one must deal with the issues of completeness versus correctness and completeness versus minimality. When dealing with a diagnostic problem, the more certain one is of relationships between causes and manifestations, the more exact the answer can be, ultimately leading to the one correct answer. However, when dealing with very uncertain relationships, it is preferable to list many outcomes so as to avoid missing the correct explanation, and to let the experienced person using the expert system decide what the correct explanation really is. Therefore, rules with simple antecedents were used in the rule-based deduction systems [see Fig. 1(a)] because the more involved patterns needed for complex antecedents are not yet known. If one tried to "guess" what these patterns are without actually being certain, this would lead to incomplete solutions which miss some of the correct interpretations. For example, a high value for computer runs per line of code, a high value for computer time per line of code, and a high value for programmer hours per line of code are all indications of *low productivity*. So, we might construct the following rule for this pattern:

> IF computer runs per line of code is above normal, and computer time per line of code is above normal, and programmer hours per line of code is above normal THEN the interpretation is *Low Productivity*.

However, what if it turns out that computer time per line of code is almost never above normal? Then this rule will almost never succeed, and we will miss the interpretation of *low productivity* even if it happens to be true.

This issue also leads to concern in the frame-based abduction systems which provide all answers of minimum cardinality. This inference mechanism works well for most diagnostic problem solving, but one must be cautiously aware of the fact that not all possible explanations are provided by this expert system. For example, if an abnormally high value of computer runs per line of code and an abnormally low value of programmer hours per software change can be explained by the combination of two interpretations, *low productivity*, and *good testing*, and also by a single interpretation, *error prone code* alone, then only the single interpretation will be provided by this system. This is because the single interpretation has a lower cardinality than the two interpretations together. As was the case in this study, some researchers now feel that the idea of providing only answers of minimum cardinality (minimal set covers) is inadequate sometimes. Research is currently being performed on a newer and better method called *irredundant covers* which provides all irredundant sets of causes which cover all of the manifes-

```
ATTRIBUTES:

/* INPUT ATTRIBUTES */
COMPUTER RUNS PER LINE OF SOURCE CODE (SGL):
    ABOVE NORMAL,
    NORMAL,
    BELOW NORMAL.
                        .
                        .
                        .
/* INFERRED ATTRIBUTE */
INTERPRETATION (MLT):
    UNSTABLE SPECIFICATIONS
    LOW PRODUCTIVITY
    HIGH PRODUCTIVITY
    GOOD TESTING OR GOOD TEST PLAN
                        .
                        .
                        .
RULES:
CRLC1 IF COMPUTER RUNS PER LINE OF CODE = ABOVE NORMAL,
        & TIME = EARLY CODING
        THEN INTERPRETATION = LOW PRODUCTIVITY <0.25>,
        & INTERPRETATION = ERROR PRONE CODE <0.75>.
                        .
                        .
                        .
SCLC3 IF SOFTWARE CHANGES PER LINE OF CODE = ABOVE NORMAL,
        & TIME = LATE CODING
        THEN INTERPRETATION = GOOD TESTING OR GOOD TEST PLAN <0.25>,
        & INTERPRETATION = ERROR PRONE CODE <0.75>.
                        .
                        .
                        .
```

(a)

```
ATTRIBUTES:

/* INPUT ATTRIBUTES */
COMPUTER RUNS PER LINE OF SOURCE CODE (SGL):
    ABOVE NORMAL,
    * NORMAL,
    BELOW NORMAL.
                    .
                    .
                    .
/* INFERRED ATTRIBUTE - FRAMES */
INTERPRETATION (MLT):
    LOW PRODUCTIVITY
     [DESCRIPTION:
          COMPUTER RUNS PER LINE OF CODE = ABOVE NORMAL;
          COMPUTER TIME PER LINE OF CODE = ABOVE NORMAL;
          PROGRAMMER HOURS PER LINE OF CODE = ABOVE NORMAL ],
    GOOD TESTING OR GOOD TEST PLAN
     [DESCRIPTION:
          SOFTWARE CHANGES PER LINE OF CODE = ABOVE NORMAL;
          SOFTWARE CHANGES PER COMPUTER RUN = ABOVE NORMAL;
          COMPUTER TIME PER SOFTWARE CHANGE = BELOW NORMAL;
          PROGRAMMER HOURS PER SOFTWARE CHANGE = BELOW NORMAL ],
                    .
                    .
                    .
```

(b)

Fig. 1. (a) Small section of (a) rule-based deduction expert system, (b) frame-based abduction expert system.

tations [19], [11]. (A set of interpretations which covers all of the manifestations is *irredundant* if none of its proper subsets also cover all of the manifestations.)

One final, but very important, fact should be noted here. ARROWSMITH-P was built using the data from one particular homogeneous environment. Therefore, the information in the knowledge base reflects this one environment and would not be transportable to other environments. However, the ideas and methods used to build ARROWSMITH-P are transportable, and that is what is important.

VI. Evaluation of Expert Systems

A. Methods of Evaluation

ARROWSMITH-P has been evaluated in several ways. The correctness of each system was measured by comparing the interpretations provided by the expert system against what actually happened during the development of the projects, thereby obtaining a measure of agreement. This analysis was performed for ten projects (the original nine plus a newer project which was completed after the development of the expert systems) in all five time phases for each of the four expert systems. Each of the original nine projects was compared against historical baselines of the remaining eight projects to determine abnormal metric values, and the tenth project, which was tested later, was compared against the original nine. A total set of 50 cases was tested on each of the four expert systems.

The actual results of what took place during development were gathered from information in another section of the database, mostly from subjective evaluation forms and project statistics forms. The subjective evaluation form contains mostly subjective information (such as a rating of the programming team's performance) and some objective numbers (such as total number of errors) concerning the project's overall development. Since the vast majority of the ratings in the subjective evaluation form is not divided by phase of the project, there probably exist some discrepancies between the results indicated in the forms and the actual interpretations for a particular phase. However, these are the closest data that are available, so we must assume that most of the interpretations for each phase are similar to the interpretations for the entire project.

The results from the expert systems were also analyzed statistically by using a Kappa statistic test [28], [10] on each interpretation. The Kappa statistic determines whether the results are better or worse than chance agreement. It takes into account the number of correct answers and the number of incorrect answers with respect to each interpretation, and it determines the amount of agreement which can be attributable to chance alone. The formula for the Kappa statistic is:

$$K = \frac{P_o - P_c}{1 - P_c}$$

TABLE III

(a) Comparison of Responses Provided by Experts in Each of the Five Time Phases for the Top-Down Expert Systems. (b) Comparison of Final Bottom-Up and Final Top-Down Expert Systems

Time Phase	Number of Relationships Indicated by Experts		
	Expert 1	Expert 2	Intersection
Early Coding	66	60	23
Middle Coding	78	65	28
Late Coding	81	68	38
Systems Test	79	48	30
Acceptance Test	68	42	23

(a)

Time Phase	Number of Relationships Used in Each Approach		
	Bottom-Up	Top-Down	Intersection
Early Coding	61	35	15
Middle Coding	65	43	19
Late Coding	63	50	23
Systems Test	65	40	17
Acceptance Test	62	37	17

(b)

where P_o is the observed proportion of agreement, and P_c is the proportion of agreement expected by chance. A value of 1 for K indicates perfect agreement, a value of 0 indicates that the results can be due to chance alone, and a value less than 0 indicates worse than chance agreement. The Kappa statistic was used for each interpretation in each of the four expert systems. This was done to determine whether certain interpretations are better understood than others.

In addition to testing the performance of the expert systems, an anlaysis was performed to compare the information provided by the two experts for the systems. This was performed by comparing the relationships indicated by each of the experts against each other and also by comparing the relationships indicated in the bottom-up systems against those indicated using the top-down approach.

B. Results

The first results we would like to discuss are those comparing information provided by the experts. This is essential because the expert systems can only perform as well as the knowledge contained in the systems permits. The experts were asked to fill in grids (one for each time phase for the bottom-up approach and one for each time phase for the top-down approach) indicating the relationships between the interpretations and the manifestations as described in Section IV. The comparison between the sets of grids for the top-down approach is provided in Table III(a). (The data for one of the experts using the bottom-up approach is incomplete, so a comparison between the two experts was not made there.) The experts only agreed in about 1/3–1/2 of their indicated relationships. Furthermore, the final set of relationships for the top-down approach is very different from the final set for the bottom-up approach. [See Table III(b).] When deciding on the relationships during the top-down development, the experts even decided to combine some of the interpretations used in the bottom-up approach, feeling there was

TABLE IV
AGREEMENT BETWEEN EXPERT SYSTEM AND INFORMATION IN DATABASE BOTTOM-UP SYSTEMS. (a) EARLY CODING PHASE. (b) MIDDLE CODING PHASE, (c) LATE CODING PHASE, (d) SYSTEMS TEST PHASE, (e) ACCEPTANCE TEST PHASE

Project	Rule-Based Deduction System Agreement	Disagreement	Extra	Frame-Based Abduction System Agreement	Disagreement	Extra
1	1	0	2	1	0	2
2	3	4	7	0	7	1
3	1	0	9	1	0	5
4	0	4	0	0	4	0
5	2	2	5	1	3	3
6	1	3	3	1	3	3
7	1	5	1	1	5	1
8	0	4	0	0	4	0
9	4	2	8	0	6	1
10	7	2	4	1	8	0
Total	20	26	39	6	40	16
Percent Agreement	43% (20/46)			13% (6/46)		

(a)

Project	Rule-Based Deduction System Agreement	Disagreement	Extra	Frame-Based Abduction System Agreement	Disagreement	Extra
1	3	0	7	0	3	1
2	0	7	0	0	7	0
3	1	1	11	0	2	1
4	2	2	1	2	2	1
5	0	5	0	0	5	0
6	4	0	7	0	4	1
7	2	4	4	0	6	2
8	5	3	2	1	7	0
9	2	4	2	2	4	2
10	5	4	5	1	8	3
Total	24	30	39	6	48	11
Percent Agreement	44% (24/54)			11% (6/54)		

(b)

Project	Rule-Based Deduction System Agreement	Disagreement	Extra	Frame-Based Abduction System Agreement	Disagreement	Extra
1	2	1	6	0	3	2
2	4	3	8	2	5	4
3	0	2	0	0	2	0
4	1	3	3	1	3	0
5	0	6	5	0	6	5
6	3	1	5	1	3	1
7	1	5	0	1	5	0
8	3	5	3	1	7	2
9	1	5	2	1	5	1
10	1	8	3	1	8	3
Total	16	39	35	8	47	18
Percent Agreement	29% (16/55)			15% (8/55)		

(c)

Project	Rule-Based Deduction System Agreement	Disagreement	Extra	Frame-Based Abduction System Agreement	Disagreement	Extra
1	1	2	8	1	2	6
2	3	4	7	0	7	1
3	0	1	0	0	1	0
4	2	2	4	1	3	0
5	1	5	0	1	5	0
6	3	1	5	0	4	1
7	2	4	3	2	4	3
8	3	5	1	3	5	1
9	2	4	6	2	4	6
10	1	8	3	1	8	3
Total	18	36	37	11	43	21
Percent Agreement	33% (18/54)			20% (11/54)		

(d)

Project	Rule-Based Deduction System Agreement	Disagreement	Extra	Frame-Based Abduction System Agreement	Disagreement	Extra
1	1	6	2	1	6	2
2	3	4	9	2	5	3
3	0	2	4	0	2	4
4	0	4	0	0	4	0
5	3	3	5	3	3	5
6	2	2	2	2	2	2
7	5	1	3	1	5	1
8	1	7	1	1	7	1
9	1	5	2	1	5	1
10	1	8	3	1	8	3
Total	17	42	31	12	47	22
Percent Agreement	29% (17/59)			20% (12/59)		

(e)

too little difference in meaning between them to be significant, and they also dismissed several interpretations during certain time phases (and *tight management* during all time phases) because they felt that the meaning of those interpretations could not be captured by the available metrics in those particular time periods. We believe that the differences between the two approaches are mainly due to two facts: 1) the experts were seeing the data from a very different point of view, and 2) the metrics are not ideal in that some of the interpretations could not be adequately described in terms of the available metrics, so the experts were not completely certain of all of the relationships that they stated and they changed their opinions over time. However, there were certain relationships which proved more consistent than others. For example, the two experts had strong agreement over the relationships involving programmer hours per line of code, software changes per line of code, and computer time per computer run. These metrics seem to be better understood than the others probably because they are often used for evaluation and comparisons in this field. They also had fairly good agreement with the interpretations of *error prone code*, *lots of reused code*, and *loose management*. The top-down and bottom-up expert systems had good agreement over programmer hours per line of code and software changes per line of code and over the interpretations of *error prone code* and *good solid code*.

The results of evaluating the four expert systems are displayed in Tables IV and V. (An expanded version of this data is presented in the technical report version of this

TABLE V
AGREEMENT BETWEEN EXPERT SYSTEM AND INFORMATION IN DATABASE TOP-DOWN SYSTEMS, (a) EARLY CODING PHASE, (b) MIDDLE CODING PHASE, (c) LATE CODING PHASE, (d) SYSTEMS TEST PHASE, (e) ACCEPTANCE TEST PHASE

Project	Rule-Based Deduction System Agreement	Disagreement	Extra	Frame-Based Abduction System Agreement	Disagreement	Extra
1	0	3	2	0	3	2
2	1	3	5	1	3	2
3	0	1	5	0	1	1
4	0	4	0	0	4	0
5	1	1	3	0	2	2
6	1	2	2	1	2	2
7	1	2	2	1	2	2
8	0	3	0	0	3	0
9	2	2	6	0	4	2
10	3	2	4	2	3	1
Total	9	23	29	5	27	14
Percent Agreement	28% (9/32)			16% (5/32)		

(a)

Project	Rule-Based Deduction System Agreement	Disagreement	Extra	Frame-Based Abduction System Agreement	Disagreement	Extra
1	2	2	6	0	4	4
2	0	5	0	0	5	0
3	1	1	7	1	1	3
4	0	2	1	0	2	1
5	0	4	0	0	4	0
6	3	1	6	0	4	1
7	0	4	4	0	4	2
8	3	4	0	2	5	0
9	1	4	3	1	4	3
10	5	3	3	3	5	2
Total	15	30	30	7	38	16
Percent Agreement	33% (15/45)			16% (7/45)		

(b)

Project	Rule-Based Deduction System Agreement	Disagreement	Extra	Frame-Based Abduction System Agreement	Disagreement	Extra
1	2	3	6	0	5	4
2	4	2	8	2	4	1
3	0	2	0	0	2	0
4	1	3	1	1	3	1
5	0	5	3	0	5	3
6	3	1	5	1	3	2
7	0	6	2	0	6	2
8	3	5	1	1	7	0
9	1	2	3	0	3	1
10	1	8	3	0	9	1
Total	15	37	32	5	47	15
Percent Agreement	29% (15/52)			10% (5/52)		

(c)

Project	Rule-Based Deduction System Agreement	Disagreement	Extra	Frame-Based Abduction System Agreement	Disagreement	Extra
1	2	3	5	1	4	4
2	4	1	8	0	5	2
3	0	0	0	0	0	0
4	1	3	3	0	4	1
5	0	5	3	0	5	3
6	1	2	5	0	3	3
7	1	5	4	0	6	3
8	1	7	0	1	7	0
9	1	3	5	1	3	3
10	1	7	3	0	8	2
Total	12	36	36	3	45	21
Percent Agreement	25% (12/48)			6% (3/48)		

(d)

Project	Rule-Based Deduction System Agreement	Disagreement	Extra	Frame-Based Abduction System Agreement	Disagreement	Extra
1	0	5	5	0	5	5
2	3	1	8	1	3	0
3	0	0	1	0	0	1
4	0	3	0	0	3	0
5	1	4	6	1	4	3
6	0	2	1	0	2	1
7	2	4	4	2	4	4
8	2	4	0	2	4	0
9	0	3	2	0	3	1
10	0	7	3	0	7	1
Total	8	33	30	6	35	16
Percent Agreement	20% (8/41)			15% (6/41)		

(e)

paper [20].) The entries in the agreement column are the number of interpretations which were indicated by both the expert system and the information in the database. The entries in the disagreement column are those interpretations indicated by the database, but not listed by the expert system. Finally, the column labeled "Extra" specifies the number of extra interpretations listed by the expert system. This number is not that meaningful in determining the performance of the rule-based systems at this time because, as discussed previously, the rule-based systems were built to provide as complete a list of interpretations as possible. The manager would then have to decide which interpretations are meaningful and disregard the others. However, in general, it is better to have as few extra interpretations as possible. It should be noted that the total number of interpretations varies from table to table. This is because certain metrics were not available for some projects in some of the time phases. It would be unfair to say the expert systems did not detect certain interpretations if they were not given the manifestations necessary to do so, so these interpretations were not included in the results of the evaluation for those particular cases.

The expert systems performed moderately well given the following limitations: 1) so much of the knowledge and relationships are unclear in this field, 2) the experts themselves do not agree on much of the knowledge, 3) the expert systems used only five variables and only nine metrics derived from these variables to achieve the list of interpretations, 4) the metrics used are not ideal, 5) many of the interpretations in the database are subjective in na-

ture and therefore may not always be correct, and 6) there may be discrepancies between the interpretations of the particular time phase and the overall interpretations for the project.

The systems which were developed with the bottom-up aproach performed better than those developed with the top-down approach, and the rule-based deduction systems performed better than the frame-based abduction systems. Both the bottom-up and top-down rule-based systems performed better than either of the frame-based systems. The bottom-up rule-based system performed best, agreeing with an average of 36 percent (ranging from 29 to 44 percent depending on time phase) of the actual interpretations indicated in the subjective evaluation forms and project statistics forms in the database, and the top-down rule-based system agreed with an average of 27 percent (ranging from 20 to 33 percent) of the database conclusions. The bottom-up frame-based system agreed with an average of 16 percent (ranging from 11 to 20 percent) of the database interpretations, and the top-down frame-based system agreed with an average of 13 percent (ranging from 6 to 16 percent) of the database conclusions. It should be pointed out that each expert system produced relatively consistent results throughout its five time phases.

The bottom-up systems contained more relationships between manifestations and interpretations than did the top-down systems. One might assume that the only reason the bottom-up systems agreed with a higher percentage of the database conclusions was that the bottom-up systems would list more interpretations for the same input manifestations (test case). If it listed more interpretations, it would get more right by chance. However, there was not that big a difference between the number of manifestations per interpretation for the bottom-up systems which was 3.16 and the number for the top-down systems which was 2.77. As mentioned before, during the top-down development, the experts combined certain interpretations and dismissed others altogether during certain time phases so there were fewer interpretations for each phase. Although the intent was to throw out inappropriate interpretations and make the top-down systems that much better, the bottom-up systems still captured a higher percentage of correct relationships than did the top-down systems. The total number of interpretations listed by the bottom-up rule-based system was 276 in the 50 test cases. Of these, 95 were in agreement with the database conclusions. The total number of interpretations listed by the top-down rule-based system was 216, and of these, 59 agreed with the database conclusions. Therefore, the bottom-up rule-based system had an average of 34 percent (95/276) correct interpretations out of all those listed, while the top-down rule-based system averaged only 27 percent (59/216) correct interpretations.

It is interesting to observe that within both the bottom-up and top-down sets of systems the frame-based system always provided a subset of the interpretations listed by the rule-based system (although in 48 percent of the combined bottom-up and top-down cases, the rule-based and frame-based systems listed the exact same interpretations). As stated previously, the relationships between the manifestations and interpretations were identical in the frame-based and rule-based systems within each knowledge acquisition approach used. Then, by the nature of the expert system methodologies, the rule-based system always listed every interpretation associated with every input manifestation, while the frame-based system only provided answers of minimum cardinality which explained all of the manifestations. Since the relationships in the two systems were identical, the frame-based systems could only list the exact same interpretations or a proper subset of those listed by the rule-based systems. As a result, the frame-based systems could not perform better than the rule-based systems with respect to agreement with the database conclusions. The frame-based systems listed an average of 50 percent fewer extra interpretations (ranging from 29 percent to 72 percent depending on time phase) for the bottom-up approach and an average of 48 percent fewer extra interpretations (ranging from 42 to 53 percent) for the top-down approach. However, it is better to have extra interpretations than to miss correct interpretations.

TABLE VI
KAPPA STATISTIC VALUES OF EACH INTERPRETATION IN EACH OF THE FOUR EXPERT SYSTEMS

Interpretation	Bottom-Up Systems		Top-Down Systems	
	RBD	FBA	RBD	FBA
Unstable Specifications	0.120	0.000	-0.065	-0.158
Low Productivity	0.270	-0.065	0.369	0.023
High Productivity	0.000	0.000	0.000	0.000
High Complexity (Tough Problem)	-0.261	-0.236	-0.346	-0.160
Compute Bound Algorithm	-0.139	-0.154	-0.253	-0.168
Low Complexity	0.122	-0.066	0.016	0.155
Simple System	0.121	0.124	***	***
Error Prone Code	0.178	0.118	0.046	0.130
Good Solid Code	-0.134	-0.174	-0.372	-0.082
Lots of Reused Code	-0.121	-0.109	-0.075	-0.163
Lots of Testing	-0.040	0.000	-0.273	-0.205
Little Testing	0.051	-0.144	-0.308	-0.238
Good Testing	0.231	0.296	-0.326	-0.198
Poor Testing	0.186	0.188	-0.241	-0.267
Lack of Thorough Testing	-0.190	-0.061	***	***
Changes Hard to Make	0.000	-0.092	0.211	0.149
Loose Management	0.124	0.123	0.427	0.194
Tight Management	-0.062	-0.114	***	***
Computer Problems	0.235	0.091	0.104	-0.092
Lots of Terminal Jockeys	0.049	-0.087	0.052	0.107

Note - K > 0 indicates better than chance agreement; K = 0 indicates chance agreement; K < 0 indicates worse than chance agreement.
RBD - Rule-Based Deduction; FBA - Frame-Based Abduction
*** - these interpretations were not used in the top-down systems

The results of using the Kappa statistic to evaluate the expert systems is shown in Table VI. According to these results, the bottom-up rule-based system performed best again, indicating better than chance agreement for more of the interpretations than the other systems did. A few of the interpretations performed relatively well in all or most of the expert systems. These were *low productivity*, *loose management*, *error prone code*, and *computer problems*. The experts had fairly good agreement with each other and also over time (between the bottom-up and the top-down approaches) on the manifestations for *loose*

management and *error prone code*. They agreed less on *low productivity* and mostly disagreed on *computer problems*. The interpretations of *low complexity*, *simple system*, and *changes hard to make* also did a little better than chance agreement. The experts had fair agreement with each other and over time concerning *changes hard to make*, but mostly disagreed over *low complexity* and *simple system*. It is interesting to note that the interpretations involving testing performed better in both bottom-up systems than in the top-down systems in general. Perhaps testing is better understood using a very environment-specific approach. Several of the interpretations did not perform well in any of the expert systems, doing worse than chance agreement in all or most cases. These were *high complexity (tough problem), compute bound algorithm, good solid code, lots of reused code, lots of testing, little testing, lack of thorough testing, and tight management.*

VII. Discussion

The goal of this study was to determine whether it is possible to build useful expert systems for software engineering management. Some of the questions which we tried to resolve involved determining how to do the knowledge acquisition and what type of expert system methodology might be best suited for this field. We used two approaches to knowledge acquisition and two expert system methodologies. The reader should be careful in drawing too strong a set of conclusions, however, because this was an exploratory experiment using a limited number of techniques for expert systems. It is very possible that other representations of the knowledge using the same or other inference mechanisms would lead to different results. Additionally, it is clear that a better and more extensive set of metrics would provide a more successful management system. This work is being continued on the TAME project [4] where various methods for structuring knowledge are being analyzed. Based upon this study, good results have also been obtained at NASA using a similar system [29].

We believe that a major limitation to developing expert systems for software engineering in general is the fact that much of the knowledge in this field is not well understood yet. Knowledge was gathered from two experts who have had a great deal of experience in this field, and it was found that they did not agree with each other about many of the relationships we were trying to determine. Furthermore, they did not always agree with themselves when looking at the data from a different point of view at a later date.

The expert systems performed moderately well, especially when one considers that many of the relationships between the metrics and the interpretations are unclear. The experts did not agree on many of the relationships, and the expert systems cannot perform better than the information included in them. Indeed, the bottom-up rule-based system performed about as well as the experts agreed with each other. In addition, a relatively small number of metrics were used to suggest many interpretations, and the metrics used were not ideal. The experts felt that some of the interpretations could not be adequately described in terms of the available metrics. For example, it was felt that the complexity interpretations could not be adequately captured without error metric data. The experts even threw out one of the interpretations altogether when they were determining relationships using the top-down approach. However, the five variables used in the metrics were easily obtainable, and this is an important consideration when creating expert systems.

Another fact we would like to stress is that the expert systems for the earlier time phases also performed well. This is especially important because a manager should learn of potential problems as early in the development process as possible. Expert systems can be very helpful because they may detect problems which a manager may not recognize early on.

Two approaches to knowledge acquisition were used and compared. The bottom-up approach produced better results than did the top-down approach. This may well be because the bottom-up approach is more environment-specific. Since the field of software engineering is still new, it is probably better to develop expert systems for one homogeneous environment rather than trying to determine general truths across different environments. In general, it may be advantageous to work with small domains when building expert systems for fields with uncertain knowledge.

The two expert system methodologies, rule-based deduction and frame-based abduction, were also compared with respect to ease of implementation and accuracy of results. The initial knowledge was derived from empirical software engineering research and organized in a table format, so the very first sets of simple rules and frames which were not time-line specific were straightforward to develop. The situation became more complex when the interpretations were made time-line specific. A time phase was added to the antecedent of each rule, so there were five times as many rules as before, specializing for each of the five time phases. Each frame-based system was divided into five systems based on time period because the second dimension of time could not be incorporated into the frames in a reasonable manner. Furthermore, an attempt was made to rewrite the rules to contain more meaningful and complex relationships among the manifestation in the antecedents. However, it was decided to retain the format of simple rules in order to be as complete as possible. It should be noted that for this type of diagnostic problem in a well-defined domain, it is generally much easier and more natural to write frames than to encode the same information in complex rules [21].

In 48 percent of the cases, the rule-based and frame-based systems provided the same interpretations. However, when analyzing the results from all projects, the rule-based systems provided more interpretations and exhibited a higher rate of agreement with the database than did the frame-based systems. This is directly attributable to the fact that simple rules containing one manifestation

in the antecedents were used in the rule-based systems, leading to solutions which contained the complete list of all possible interpretations associated with the manifestations, while the frame-based systems provided only those explanations of minimum cardinality and often missed correct interpretations because the relationships between interpretations and manifestations were not always correct. It is better to have extra interpretations than to miss correct interpretations, so we conclude that a rule-based system with simple rules is probably more applicable to newer fields with unclear knowledge, such as software engineering. However, as a field becomes more established, a frame-based system may provide better solutions. Also, newer methods of implementing frame-based abduction with irredundant covers should provide better results than those currently provided by frame-based abduction using minimal set covers.

This study has provided many additional new insights into the development of expert systems for software engineering management. It is feasible to develop prototype expert systems at this point in time, but one must realize that in any new field with uncertain knowledge, the expert systems cannot perform better than the state of knowledge in the field permits. One of the best reasons to develop these systems may be to learn from their development. The knowledge engineer can learn a great deal about a field as he organizes the information. Then, analyzing the performance of the working systems can give further insight about what is and what is not understood. In order to develop better expert systems for software engineering management, one needs to define fully the relationships that exist between the components. In particular one must define what development characteristics would result in what types of abnormal measures, how this changes through various project development phases, and how certain one is that an abnormal measure results from a certain characteristic. As more is learned about software engineering management, more can be incorporated into useful expert systems.

VIII. Future Research Directions

The development of ARROWSMITH-P was a preliminary attempt at constructing expert systems for software engineering management. Replications of this experiment using varying approaches to building the expert systems will lead to stronger confidence in the results and a better understanding of the effects.

There is certainly a need for futher research in the field of software engineering. As more is learned, the information contained in the knowledge bases can be refined, and new knowledge, such as information about error metrics [30], [2] or information about other phases of development such as requirements or design, can be incorporated into the expert systems to make them stronger. As incorrect relationships are brought to the surface, the systems can be changed to incorporate the knowledge gained from testing. Eventually, the rules should become more complex as relationships between manifestations and causes become better defined. In addition, the testing of current, ongoing projects can be performed on the expert systems. The data from the new projects can then be incorporated into the environment-specific baselines of metrics so the systems continue to be updated as the environment changes.

In a more general sense, a theoretical framework for developing expert systems for software engineering is needed. For example, a categorization scheme, which would address such issues as when a top-down system is better than a bottom-up system and vice versa, should be built. Also, perhaps a new and different type of inference mechanism or method for building expert systems would better suit the needs of some aspects in this field. All of these issues require a great deal of further research and analysis.

Appendix A
A Sample Interactive Session with the Rule-Based Deduction Expert System

```
THIS EXPERT SYSTEM WILL HELP A MANAGER OF A SOFTWARE PROJECT
DETERMINE IF THE PROJECT IS ON SCHEDULE OR IN TROUBLE.
PLEASE ANSWER THE FOLLOWING QUESTIONS.

 COMPUTER RUNS PER LINE OF SOURCE CODE:
 (1)ABOVE NORMAL
 (2)NORMAL
 (3)BELOW NORMAL
 = ?
2.

 COMPUTER TIME PER LINE OF SOURCE CODE:
 (1)ABOVE NORMAL
 (2)NORMAL
 (3)BELOW NORMAL
 = ?
2.

 SOFTWARE CHANGES PER LINE OF SOURCE CODE:
 (1)ABOVE NORMAL
 (2)NORMAL
 (3)BELOW NORMAL
 = ?
2.

 PROGRAMMER HOURS PER LINE OF SOURCE CODE:
 (1)ABOVE NORMAL
 (2)NORMAL
 (3)BELOW NORMAL
 = ?
2.

 COMPUTER TIME PER COMPUTER RUN:
 (1)ABOVE NORMAL
 (2)NORMAL
 (3)BELOW NORMAL
 = ?
2.

 SOFTWARE CHANGES PER COMPUTER RUN:
 (1)ABOVE NORMAL
 (2)NORMAL
 (3)BELOW NORMAL
 = ?
2.

 PROGRAMMER HOURS PER COMPUTER RUN:
 (1)ABOVE NORMAL
 (2)NORMAL
 (3)BELOW NORMAL
 = ?
3.

 PROJECT TIME PHASE:
 (1)EARLY CODE PHASE
 (2)MIDDLE CODE PHASE
 (3)LATE CODE PHASE
 (4)SYSTEMS TEST PHASE
 (5)ACCEPTANCE TEST PHASE
 = ?
2.

 COMPUTER TIME PER SOFTWARE CHANGE:
 (1)ABOVE NORMAL
```

```
(2)NORMAL
(3)BELOW NORMAL
= ?
2.

PROGRAMMER HOURS PER SOFTWARE CHANGE:
(1)ABOVE NORMAL
(2)NORMAL
(3)BELOW NORMAL
= ?
3.

POSSIBLE INTERPRETATIONS ARE:
 ERROR PRONE CODE <0.94>
 EASY ERRORS OR CHANGES BEING FOUND OR FIXED <0.81>
 LOTS OF TESTING <0.75>
 LOTS OF TERMINAL JOCKEYS <0.75>
 UNSTABLE SPECIFICATIONS <0.50>
 NEAR BUILD OR MILESTONE DATE <0.50>
 GOOD TESTING OR GOOD TEST PLAN <0.25>
 MODIFICATIONS BEING MADE TO RECENTLY TRANSPORTED CODE <0.25>
```

Note - User answers are in boldface.

Appendix B
A Sample Interactive Session with the Frame-Based Abduction Expert System

```
THIS EXPERT SYSTEM WILL HELP A MANAGER OF A SOFTWARE PROJECT
DETERMINE IF THE PROJECT IS ON SCHEDULE OR IN TROUBLE.
THIS PARTICULAR SYSTEM SHOULD BE USED FOR THE MIDDLE CODING PHASE.
PLEASE ANSWER THE FOLLOWING QUESTIONS.

FOCUS OF SUBPROBLEM:
 ***THIS SUBPROBLEM IS CURRENTLY ACTIVE***
 GENERATOR:
  COMPETING POSSIBILITIES:
   UNSTABLE SPECIFICATIONS
   LATE DESIGN
   NEW OR LATE DEVELOPMENT
   LOW PRODUCTIVITY
   HIGH PRODUCTIVITY
   HIGH COMPLEXITY OR TOUGH PROBLEM
   HIGH COMP OR COMPUTE BOUND ALGORITHMS RUN OR TESTED
   LOW COMPLEXITY
   SIMPLE SYSTEM
   REMOVAL OF CODE BY TESTING OR TRANSPORTING
   INFLUX OF TRANSPORTED CODE
   LITTLE EXECUTABLE CODE BEING DEVELOPED
   ERROR PRONE CODE
   GOOD SOLID AND RELIABLE CODE
   NEAR BUILD OR MILESTONE DATE
   LARGE PORTION OF REUSED CODE OR EARLY AND LARGER TESTS
   LOTS OF TESTING
   LITTLE OR NOT ENOUGH ONLINE TESTING BEING DONE
   GOOD TESTING OR GOOD TEST PLAN
   UNIT TESTING BEING DONE
   LACK OF THOROUGH TESTING
   POOR TESTING PROGRAM
   SYSTEM AND INTEGRATION TESTING STARTED EARLY
   CHANGE BACKLOG OR HOLDING CHANGES
   CHANGE BACKLOG OR HOLDING CODE
   CHANGES HARD TO ISOLATE
   CHANGES HARD TO MAKE
   EASY ERRORS OR CHANGES BEING FOUND OR FIXED
   MODIFICATIONS BEING MADE TO RECENTLY TRANSPORTED CODE
   LOOSE CONFIGURATION MANAGEMENT OR UNSTRUCTURED DEV
   TIGHT MANAGEMENT PLAN OR GOOD CONFIGURATION CONTROL
   COMPUTER PROBLEMS OR INACCESSIBILITY OR ENV CONSTRAINTS
   LOTS OF TERMINAL JOCKEYS

COMPUTER RUNS PER LINE OF SOURCE CODE:
(1) ABOVE NORMAL
(2) NORMAL
(3) BELOW NORMAL
= ?
2.

COMPUTER TIME PER LINE OF SOURCE CODE:
(1) ABOVE NORMAL
(2) NORMAL
(3) BELOW NORMAL
= ?
2.

SOFTWARE CHANGES PER LINE OF SOURCE CODE:
(1) ABOVE NORMAL
(2) NORMAL
(3) BELOW NORMAL
= ?
2.

PROGRAMMER HOURS PER LINE OF SOURCE CODE:
(1) ABOVE NORMAL
(2) NORMAL
(3) BELOW NORMAL
= ?
2.

SOFTWARE CHANGES PER COMPUTER RUN:
(1) ABOVE NORMAL
(2) NORMAL
(3) BELOW NORMAL
= ?
2.

COMPUTER TIME PER COMPUTER RUN:
(1) ABOVE NORMAL
(2) NORMAL
(3) BELOW NORMAL
= ?
2.

PROGRAMMER HOURS PER COMPUTER RUN:
(1) ABOVE NORMAL
(2) NORMAL
(3) BELOW NORMAL
= ?
3.

FOCUS OF SUBPROBLEM:
 GENERATOR:
  COMPETING POSSIBILITIES:
   LOTS OF TERMINAL JOCKEYS
   EASY ERRORS OR CHANGES BEING FOUND OR FIXED
   LOTS OF TESTING
   ERROR PRONE CODE
   UNSTABLE SPECIFICATIONS

PROGRAMMER HOURS PER SOFTWARE CHANGE:
(1) ABOVE NORMAL
(2) NORMAL
(3) BELOW NORMAL
= ?
3.

FOCUS OF SUBPROBLEM:
 GENERATOR:
  COMPETING POSSIBILITIES:
   EASY ERRORS OR CHANGES BEING FOUND OR FIXED
   ERROR PRONE CODE

COMPUTER TIME PER SOFTWARE CHANGE:
(1) ABOVE NORMAL
(2) NORMAL
(3) BELOW NORMAL
= ?
2.

POSSIBLE INTERPRETATIONS ARE:
 EASY ERRORS OR CHANGES BEING FOUND OR FIXED <H>
 ERROR PRONE CODE <L>
```

Note - User answers are in boldface.

- Both interpretations listed as solutions can explain all of the manifestations, but the first is given a high measure of likelihood (shown by the ⟨H⟩) of being correct, while Error Prone Code is rated low.

Acknowledgment

The authors are grateful to F. McGarry, Dr. J. Page, Dr. J. Reggia, J. Ramsey, B. Decker, and D. Card for their invaluable assistance in this project. The authors would also like to thank the members of their research group for enlightening comments and ideas.

References

[1] "Annotated bibliography of Software Engineering Laboratory (SEL) literature, SEL-82-006," Software Eng. Lab., NASA Goddard Space Flight Center, Greenbelt, MD, Nov. 1982.

[2] V. R. Basili and B. T. Perricone, "Software errors and complexity: An empirical investigation," *Commun. ACM.*, vol. 27, no. 1, pp. 42–52, Jan. 1984.

[3] V. R. Basili and C. L. Ramsey, "ARROWSMITH-P—A prototype expert system for software engineering management," in *Proc. Expert Systems in Government Symposium*, IEEE, McLean, VA, Oct. 1985, pp. 252–264.

[4] V. R. Basili and H. D. Rombach, "The TAME project: Towards

improvement-oriented software environments," *IEEE Trans. Software Eng.*, vol. SE-14, no. 6, pp. 758-773, June 1988.
[5] V. R. Basili and D. M. Weiss, "A methodology for collecting valid software engineering data," *IEEE Trans. Software Eng.*, vol. SE-10, no. 6, pp. 728-738, Nov. 1984.
[6] V. R. Basili and M. V. Zelkowitz, "Analyzing medium scale software developments," in *Proc. Third Int. Conf. Software Engineering*, Atlanta, GA, May 1978, pp. 116-123.
[7] V. R. Basili, M. V. Zelkowitz, F. E. McGarry, R. W. Reiter, Jr., W. F. Truszkowksi, and D. M. Weiss, "The Software Engineering Laboratory, SEL-77-001," Software Eng. Lab., NASA Goddard Space Flight Center, Greenbelt, MD, May 1977.
[8] A. N. Campbell, V. F. Hollister, R. O. Duda, and P. E. Hart, "Recognition of a hidden mineral deposit by an artificial program," *Science*, vol. 217, pp. 927-928, Sept. 1982.
[9] D. N. Card, F. E. McGarry, J. Page, S. Eslinger, and V. R. Basili, "The Software Engineering Laboratory, SEL-81-104," Software Eng. Lab., NASA Goddard Space Flight Center, Greenbelt, MD, Feb. 1982.
[10] J. Cohen, "Weighted Kappa: Nominal scale agreement with provision for scaled disagreement or partial credit," *Psychol. Bull.*, vol. 70, pp. 213-220, 1968.
[11] J. deKleer and B. Williams, "Reasoning about multiple faults," in *Proc. Fifth Nat. Conf. Artificial Intelligence*, Philadelphia, PA, Aug. 11-15, 1986, pp. 132-139.
[12] C. Doerflinger and V. R. Basili, "Monitoring software development through dynamic variables," *IEEE Trans. Software Eng.*, vol. 11, no. 9, pp. 978-985, Sept. 1985.
[13] F. Hayes-Roth, D. Waterman, and D. Lenat, "Principles of Pattern-directed inference systems," in *Pattern-Directed Inference Systems*, Waterman and Hayes-Roth, Eds. New York: Academic, 1978, pp. 577-601.
[14] R. Miller, H. Pople, and J. Myers, "Internist-1: An experimental computer-based diagnostic consultant for general internal medicine," *New England J. Med.*, vol. 307, pp. 468-476, 1982.
[15] M. Minsky, "A framework for representing knowledge," in *The Psychology of Computer Vision*, P. Winston, Ed. New York: McGraw-Hill, 1975. pp. 211-277.
[16] D. S. Nau and J. A. Reggia, "Relationships between deductive and abductive inference in knowledge-based diagnostic expert systems," in *Proc. First Int. Workshop Expert Database Systems*, 1984, pp. 500-509.
[17] S. G. Pauker, G. A. Gorry, J. P. Kassirer, and W. B. Schwartz, "Towards the simulation of clinical cognition," *Amer. J. Med.*, vol. 60, no. 7, pp. 981-996, June 1976.
[18] Y. Peng and J. A. Reggia, "A probabilistic causal model for diagnostic problem-solving," *IEEE Trans. Syst., Man, Cybern.*, vol. 17, pp. 146-162, 395-406, 1987.
[19] Y. Peng and J. A. Reggia, "Plausibility of diagnostic hypotheses: The nature of simplicity," in *Proc. Fifth Nat. Conf. Artificial Intelligence*, Philadelphia, PA, Aug. 11-15, 1986, pp. 140-145.
[20] C. L. Ramsey and V. R. Basili, "An evaluation of expert systems for software engineering management," Dep. Comput. Sci., Univ. Maryland, College Park, Tech. Rep. TR-1708, Sept. 1986.
[21] C. L. Ramsey, J. A. Reggia, D. S. Nau, and A. Ferrentino, "A comparative analysis of methods for expert systems," *Int. J. Man-Machine Studies*, vol. 24, no. 5, pp. 475-499, May 1986.
[22] J. Reggia, "Computer-assisted medical decision making," in *Application of Computers in Medicine*, M. Schwartz, Ed. New York: IEEE Press, 1982, pp. 198-213.
[23] J. A. Reggia, D. S. Nau, and P. Wang, "Diagnostic expert systems based on a set covering model," *Int. J. Man-Machine Studies*, vol. 19, no. 5, pp. 437-460, Nov. 1983.
[24] J. A. Reggia, D. S. Nau, P. Wang, and Y. Peng, "A formal model of diagnostic inference," *Inform. Sci.*, vol. 37, pp. 227-285, 1985.
[25] J. A. Reggia and B. Perricone, "KMS reference manual," Dep. Comput. Sci., Univ. Maryland, College Park, Tech. Rep. TR-1136, 1982.
[26] E. Shortliffe, *Computer-Based Medical Consultations: MYCIN*. New York: Elsevier, 1976.
[27] H. Shubin and J. Ulrich, "IDT: An intelligent diagnostic tool," in *Proc. Nat. Conf. Artificial Intelligence*, AAAI, 1982, pp. 290-295.
[28] R. Spitzer, J. Cohen, J. Fleiss, and J. Endicott, "Quantification of agreement in psychiatric diagnosis," *Archives General Psychiatry*, vol. 17, pp. 83-87, 1967.
[29] J. D. Valett, W. Decker, and J. Buell, "Software management environment," in *Proc. SEL Workshop 1988*, NASA Goddard Space Flight Center, Greenbelt, MD, Dec. 1988.
[30] D. M. Weiss and V. R. Basili, "Evaluating software development by analysis of changes: Some data from the software engineering laboratory," *IEEE Trans. Software Eng.*, vol. SE-11, no. 2, pp. 157-168, Feb. 1985.

Connie Loggia Ramsey (M'88) received the B.A. degree in biology from the State University of New York at Binghamton and the M.S. degree in computer science from the University of Maryland at College Park.

She is currently a Research Scientist at the Navy Center for Applied Research in Artificial Intelligence at the Naval Research Laboratory, Washington, DC. Her current research interests include expert systems, classification problem solving, reasoning with uncertainty, machine learning, and parallel processing.

Ms. Ramsey is a member of the IEEE Computer Society and the American Association for Artificial Intelligence.

Victor R. Basili (M'83-SM'84) is Professor and Chairman of the Department of Computer Science at the University of Maryland, College Park. He was involved in the design and development of several software projects, including the SIMPL family of programming languages. He is currently measuring and evaluating software development in industrial and government settings and has consulted with many agencies and organizations, including IBM, GE, CSC, GTE, MCC, AT&T, Motorola, HP, NRL, NSWC, and NASA. He is one of the founders and principals in the Software Engineering Laboratory, a joint venture between NASA Goddard Space Flight Center, the University of Maryland and Computer Sciences Corporation, established in 1976. He has been working on the development of quantitative approaches for software management, engineering, and quality assurance by developing models and metrics for the software development process and product. He has authored over 90 papers. In 1982, he received the Outstanding Paper Award from the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING for his paper on the evaluation of methodologies.

Dr. Basili is currently the Editor-in-Chief of the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING and was Program Chairman for several conferences including the 6th International Conference on Software Engineering. He has served on the Editorial Board of the *Journal of Systems and Software*. He is a member of the Board of Governors of the IEEE Computer Society.