Technical Report TR-1563 October 1985

ARROWSMITH-P - A Prototype Expert System for Software Engineering Management *

Victor R. Basili and Connie Loggia Ramsey

Department of Computer Science
University of Maryland at College Park

Abstract

Although the field of software engineering is relatively new, it can benefit from the use of expert systems. Two prototype expert systems have been developed to aid in software engineering management. Given the values for certain metrics, these systems will provide interpretations which explain any abnormal patterns of these values during the development of a software project. The two systems, which solve the same problem, were built using different methods, rule-based deduction and frame-based abduction, so a comparison could be done to see which method better suits the needs of this field. It was found that both systems performed moderately well, but the rule-based deduction system using simple rules provided more complete solutions than did the frame-based abduction system.

* Research supported in part by the National Aeronautics and Space Administration Grant NSG-5123 to the University of Maryland. Computer support provided in part by the Computer Science Center of the University of Maryland.

1. 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 benefit from the use of expert systems. As pointed out above, some of the major advantages to expert systems are derived from the ability to learn from them. This learning experience can take place on at least two different levels. The development of an expert system for software engineering provides a learning experience by pointing out how much we do not know yet, but also by forcing the knowledge engineer to develop and organize relationships between various pieces of knowledge, such as metrics and their interpretations.

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 knowledge as well as a history of information from a particular software development environment, and this can be very helpful to inexperienced managers and developers.

This paper will focus on two prototype expert systems, collectively named ARROWSMITH-P.* This represents a first attempt at organizing some of the knowledge and defining some of the problems associated with the process of creating expert systems for software engineering. ARROWSMITH-P is intended to aid the manager of a software development project in an automated manner. The systems work as follows. First, it is determined whether or not a software project is following normal development 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 low productivity, for any abnormal software development patterns. These systems can 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 2 provides a brief overview of the underlying methodology used to build the expert systems discussed in this paper. Section 3 details the implementations of ARROWSMITH-P, and Section 4 discusses the issues and problems associated with this process. Section 5 furnishes the details for the evaluation of the two expert systems. Section 6 then discusses results and conclusions from the development and testing of the expert systems. Finally, Section 7 discusses current and future research needs.

---

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

2. BACKGROUND

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 [1], 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.

2.1. Rule-Based Deduction

A common method for expert systems, and essentially the "standard" in AI today, 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 non-procedural 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 [2].) 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 [3] and PROSPECTOR [4] are two well-known examples of expert systems which incorporate rule-based deduction.

2.2. 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 [5], 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 [6] for a review). INTERNIST [7], KMS.HT [1], [8], PIP [9], and IDT [10] 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** 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 [8], [11], [12], [13].

3. IMPLEMENTATIONS

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

3.1. Methodology

The following methodology for constructing expert systems for software engineering management was developed. (An earlier version of this reasoning was presented in [14].) 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 that 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 start of coding and start of 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, interpretations, such as unstable specifications or good testing, are determined which 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 interpretations 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

---

** 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.

interpretations or diagnoses of the abnormalities are developed. This approach 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. A top-down approach would first define the various disorders or causes and then associate the correct manifestations or effects with each disorder.

The input to the expert systems consists 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.

3.2. Actual Implementations

ARROWSMITH-P is based on previous research conducted on the NASA/Goddard Space Flight Center Software Engineering Laboratory (SEL) environment [14]. Since the SEL environment is homogeneous, it was possible to use the bottom-up methodology described above to produce historical, environment-specific baselines of normalized metrics from the highly reliable data of nine software projects. (See [15], [16], [17], [18], [19] for fuller descriptions of the SEL environment.) Altogether, nine metrics (shown in Table 1) proved satisfactory, exhibiting baselines with reasonable standard deviations. 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 interpretations for abnormal values of metrics were mostly derived from Frank McGarry of NASA/GSFC and Jerry Page of CSC, experts who have had a great deal of experience in this field and particularly in the SEL environment. The set of interpretations was later modified and made time-line specific for use in the development of ARROWSMITH-P. (The complete list of interpretations used in the expert systems is displayed in Table 2.) In addition, measures to indicate how certain one is that the deviation of a particular metric has resulted from a particular interpretation were included.

As stated previously, two different methods were used to build the two 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 2. In the rule-based system, the rules are of the form

TABLE 1 - 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 2 - INTERPRETATIONS USED IN EXPERT SYSTEM

* Unstable Specifications
Late Design
New or Late Development
* Low Productivity
* High Productivity
* High Complexity or Tough Problem
* High Complexity 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 Development
* Tight Management Plan or Good Configuration Control
* Computer Problems or Inaccessibility or Environmental Constraints
* Lots of Terminal Jockeys

Note - * indicates that this interpretation was used in the evaluation of the expert systems

"IF manifestations THEN interpretations," while in the frame-based system, there is one frame (containing a list of manifestations) for each interpretation. The two systems 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. However, the certainty factors attached to the rules could not be directly translated to measures of likelihood in the frames so these measures of likelihood were omitted. For example, 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 [20].) Figure 1 shows a sample section of each knowledge base. Example sessions with the expert systems are provided in Appendix 1.

```
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>.
                        .
                        .
                        .
```

Figure 1a - Small Section of Rule-Based Deduction Expert System.

```
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 ],
                    .
                    .
                    .
```

Figure 1b - Small Section of Frame-Based Abduction Expert System

4. 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 [20]. 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 is often unclear. 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 system (see Figure 1a) 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 system which provides all answers of minimum cardinality. The inference mechanism works very 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.

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.

5. EVALUATION OF EXPERT SYSTEMS

A preliminary evaluation of ARROWSMITH-P has been done. The method used to do the evaluation was simply to compare the interpretations provided by the expert systems against what actually happened during the development of the projects, thereby obtaining a measure of agreement. The actual results were gathered from information in 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. Altogether, 20 out of the 33 possible interpretations could be checked against measures from these forms. (These are starred in Table 2.)

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. In addition, some of the interpretations derived from analyzing the data in the database were very evident, while others were somewhat uncertain. Therefore, these two classes were partitioned in the analysis of agreement between the expert systems and the information in the database.

The interpretations for the acceptance test phase were evaluated for all nine projects, and this analysis was performed for both expert systems. The results are displayed in Table 3. 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 first number depicts those interpretations which were explicit in the database, while the second number represents those which were marginally indicated. The entries in the disagreement column are those interpretations indicated by the database, but not listed by the expert system. Again, the first number represents those which are certain and the second number is those which are marginal. Finally, the column labeled "Extra" specifies the number of extra interpretations (out of the 20 possible from the information in the database) listed by the expert system. This number is not that meaningful in determining the performance of the rule-based system at this time because, as discussed previously, the rule-based system was 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.

The expert systems performed moderately well given that (1) so much of the knowledge and relationships are unclear in this field, (2) the expert systems used only five variables and only nine metrics derived from these variables to achieve the list of interpretations, (3) many of the interpretations in the database are subjective in nature and therefore may not always be correct, and (4) there may be discrepancies between the interpretations of the particular phase and the overall interpretations for the project.

Table 3 - Agreement between Expert System and Information in Database

Project	Rule-Based Deduction System			Frame-Based Abduction System		
	Agreement	Disagreement	Extra	Agreement	Disagreement	Extra
1	1 - 0	3 - 3	2	1 - 0	3 - 3	2
2	0 - 3	3 - 1	9	0 - 2	3 - 2	3
3	0 - 0	0 - 2	5	0 - 0	0 - 2	5
4	2 - 0	4 - 3	2	2 - 0	4 - 3	2
5	3 - 0	3 - 0	5	3 - 0	3 - 0	5
6	1 - 1	0 - 2	2	1 - 1	0 - 2	2
7	5 - 2	0 - 0	5	1 - 0	4 - 2	0
8	1 - 0	4 - 3	1	1 - 0	4 - 3	1
9	2 - 2	1 - 1	5	2 - 2	1 - 1	5
Total	15 - 8	18 - 15	36	11 - 5	22 - 18	25

The rule-based system performed better, agreeing with 45% (15/33) of the very evident interpretations from the database and 35% (8/23) of the more uncertain interpretations. The frame-based system agreed with 33% (11/33) of the clearcut database conclusions and 22% (5/23) of the more uncertain interpretations. Of course, the agreement with the more evident database interpretations is much more important than agreement with the uncertain conclusions. It is interesting to observe that both expert systems provided the exact same interpretations (with respect to the 20 interpretations discussed here) in seven out of nine projects. The only differences occurred in projects 2 and 7, where the frame-based system resulted in very few interpretations (adding the number of interpretations in agreement with the database to the number of extra interpretations, there were 5 for project 2 and 1 for project 7) which covered all of the manifestations. The rule-based system performed much better on these two projects, agreeing with 43% (3/7) of the combined clearcut and marginal database interpretations for project 2 and 100% (7/7) of the interpretations for project 7. The frame-based system agreed with only 29% (2/7) of the database conclusions for project 2 and 14% (1/7) of the database conclusions for project 7. Also, these differences resulted in 31% (11/36) fewer extra interpretations for the frame-based system, but again, it is better to have extra interpretations than to miss correct interpretations. It should be noted that evaluation and testing of these expert systems will continue, and any information learned about incorrect relationships, etc. will be incorporated into the systems to make them stronger.

6. DISCUSSION

The goal of this study was to build useful expert systems for software engineering; a major subgoal was to determine what type of expert system might be best suited for this field with respect to ease of implementation and accuracy of results. Two methods, rule-based deduction and frame-based abduction, were chosen as methods for implementation. Another common method for building expert systems is statistical pattern classification, but this method was not used because the needed statistics are not available yet in this relatively new field. It should be noted that estimates are not acceptable

because system performance is greatly reduced when estimates are used [21], [22], [23].

The initial knowledge was derived from empirical software engineering research and organized in a table format, so the first set 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. The 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 manifestations 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 [20].

The two expert systems performed moderately well, especially when one considers that a relatively small number of metrics were used to suggest many interpretations, and that many of the relationships between the metrics and the interpretations are unclear. In seven out of nine projects, the two systems provided the same interpretations. However, when analyzing the results from all nine projects, the rule-based system provided more interpretations and exhibited a higher rate of agreement with the database than did the frame-based system. This is directly attributable to two facts: (1) simple rules were used in the rule-based system, allowing completeness of the list of interpretations, and (2) the frame-based system only provides those explanations of minimum cardinality. Therefore, we conclude that the rule-based system with simple rules is probably more applicable to the field of software engineering at this point in time. However, this may very well not be true in the future, as more is learned in this field.

This study has provided many additional new insights into the development of expert systems for software engineering by stressing the need to define 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.

7. FUTURE RESEARCH DIRECTIONS

The development of ARROWSMITH-P is a preliminary attempt at constructing expert systems for software engineering management. The information contained in the knowledge base can be refined, and new knowledge, such as information about error metrics [24], [25], can be incorporated into these systems as more is learned. As these systems are evaluated further and incorrect relationships are brought to the surface, they 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 will be performed on the two systems. The data from the new projects will then be incorporated into the environment-specific baselines of metrics so the systems continue to be updated as the environment changes. Another extension of this project will be to redesign the systems using a top-down approach, looking first at the possible interpretations and then deciding what metrics might provide information about those interpretations. This

should provide new insights and a more complete picture.

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.

8. ACKNOWLEDGEMENT

The authors are grateful to Frank McGarry, Dr. Jerry Page, Dr. James Reggia, James Ramsey, Bill Decker, and Dave 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] J. Reggia and B. Perricone, KMS Reference Manual, Tech. Report TR-1136, Computer Science Department, University of Maryland, 1982.

[2] F. Hayes-Roth, D. Waterman, and D. Lenat, Principles of Pattern-Directed Inference Systems, pp. 577-601 in *Pattern-Directed Inference Systems,* ed. Waterman and Hayes-Roth, Academic Press, 1978.

[3] E. Shortliffe, *Computer-Based Medical Consultations: MYCIN,* Elsevier, 1976.

[4] A. N. Campbell, V. F. Hollister, R. O. Duda, and P. E. Hart, Recognition of a Hidden Mineral Deposit by an Artificial Intelligence Program, *Science* **217**, pp. 927-928, 3 September 1982.

[5] M. Minsky, A Framework for Representing Knowledge, pp. 211-277 in *The Psychology of Computer Vision,* ed. P. Winston, McGraw-Hill, Inc., 1975.

[6] J. Reggia, Computer-Assisted Medical Decision Making, pp. 198-213 in *Applications of Computers in Medicine,* ed. M. Schwartz, IEEE Press, 1982.

[7] R. Miller, H. Pople, and J. Myers, Internist-1: An Experimental Computer-Based Diagnostic Consultant for General Internal Medicine, *New England Journal of Medicine* **307**, pp. 468-476, 1982.

[8] J. Reggia, D. Nau, and P. Wang, Diagnostic Expert Systems Based on a Set Covering Model, *International Journal of Man-Machine Studies,* pp. 437-460, Nov. 1983.

[9] S. et al Pauker, Towards the Simulation of Clinical Cognition, *American Journal of Medicine* **60**, pp. 981-996, 1976.

[10] H. Shubin and J. Ulrich, IDT: An Intelligent Diagnostic Tool, pp. 290-295 in *Proceedings of the National Conference on Artificial Intelligence,* AAAI, 1982.

[11] J. Reggia, D. Nau, and P. Wang, A Theory of Abductive Inference in Diagnostic Expert Systems, Tech. Report TR-1338, Computer Sci. Dept., Univ. of Maryland, College Park, MD, December 1983.

[12] D. S. Nau and J. A. Reggia, Relationships Between Deductive and Abductive Inference in Knowledge-Based Diagnostic Expert Systems, pp. 500-509 in *Proceedings of the First International Workshop on Expert Database Systems,* 1984.

[13] Y. Peng, A General Theoretical Model for Abductive Diagnostic Expert Systems, Tech. Report TR-1402, Computer Science Department, University of Maryland, May 1984.

[14] C. Doerflinger and V. Basili, Monitoring Software Development Through Dynamic Variables, pp. 434-445 in *Proceedings of the IEEE Computer Society's International Computer Software and Applications Conference,* Nov. 1983. (also to appear in IEEE Transactions on Software Engineering).

[15] V. R. Basili, M. V. Zelkowitz, F. E. McGarry, R. W. Reiter, Jr., W. F. Truszkowski, and D. M. Weiss, The Software Engineering Laboratory, SEL-77-001, Software Engineering Laboratory, NASA/Goddard Space Flight Center, Greenbelt,Maryland, May 1977.

[16] V. R. Basili and D. M. Weiss, A Methodology for Collecting Valid Software Engineering Data, *IEEE Transactions on Software Engineering* **SE-10**, 6, pp. 728-738, Nov. 1984.

[17] V. R. Basili and M. V. Zelkowitz, Analyzing Medium-Scale Software Developments, pp. 116-123 in *Proceedings of the Third International Conference on Software Engineering,* Atlanta, Georgia, May 1978.

[18] D. N. Card, F. E. McGarry, J. Page, S. Eslinger, and V. R. Basili, The Software Engineering Laboratory, SEL-81-104, Software Engineering Laboratory, NASA/Goddard Space Flight Center, Greenbelt,Maryland, Feb. 1982.

[19] Annotated Bibliography of Software Engineering Laboratory (SEL) Literature, SEL-82-006, Software Engineering Laboratory, NASA/Goddard Space Flight Center, Greenbelt,Maryland, Nov. 1982.

[20] C. Ramsey, J. Reggia, D. Nau, and A. Ferrentino, A Comparative Analysis of Methods for Expert Systems, *International Journal of Man-Machine Studies,* 1985. Submitted.

[21] A. Shapiro, The Evaluation of Clinical Predictions, *New England Journal of Medicine* **296**, pp. 1509-1514, 1977.

[22] A. Tversky, Assessing Uncertainty, *36 (B),* pp. 148-159, 1974.

[23] D. et al Leaper, Computer-Assisted Diagnosis of Abdominal Pain Using Estimates Provided by Clinicians, *British Medical Journal* **4**, pp. 350-354, 1972.

[24] D. M. Weiss and V. R. Basili, Evaluating Software Development by Analysis of Changes: Some Data From the Software Engineering Laboratory, *IEEE Transactions on Software Engineering* **SE-11**, 2, pp. 157-168, Feb. 1985.

[25] V. R. Basili and B. T. Perricone, Software Errors and Complexity: An Empirical Investigation, *Communications of the ACM* **27**, 1, pp. 42-52, Jan. 1984.

APPENDIX 1a - 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 1b - 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.