EXPERT SYSTEMS FOR SOFTWARE ENGINEERING MANAGEMENT : A SUMMARIZED EVALUATION

Connie Loggia Ramsey* and Victor R. Basili
Department of Computer Science
University of Maryland
College Park, Maryland 20742

ABSTRACT

Although the field of software engineering is relatively new, it can benefit from the use of expert systems. Four separate, prototype expert systems have been developed 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 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.

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 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.

---

* Currently at the Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory, Washington, D.C. 20375

Published 1988 by Elsevier Science Publishing Co., Inc.
Expert Systems and Advanced Data Processing
M.L. Emrich, A.R. Sadlowe, and L.F. Arrowood, Editors

Since software engineering is still such a new field with much of its knowledge unclear, special attention has to be given to basic research issues concerning the development of expert systems in this field. The high level goals of this project were to determine: (1) Are expert systems for software engineering, or any new field with uncertain knowledge, feasible? (2) What methodology should be used for knowledge acquisition? (3) What type of expert system methodology best suits software engineering? (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 systems?

This paper will summarize the results of the development and comparative analysis of four separate, prototype expert systems, collectively named ARROWSMITH-P. A more in-depth analysis of these systems can be found in (Ramsey, 1986a), and earlier versions of these expert systems are described in (Basili, 1985). This represents an early attempt at defining the process of creating expert systems for software engineering management. 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 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 2 provides a brief overview of the underlying methodology used to build the expert systems discussed in this paper, and Section 3 details the implementations of ARROWSMITH-P. Section 4 furnishes the details for the evaluation of the expert systems. Section 5 then discusses results and conclusions from the development and testing of the expert systems. Finally, Section 6 discusses current and future research needs.

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 (Reggia, 1982a), 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. Rule-based deduction is a common method, and it is briefly described below; frame-based abduction is a newer method which has not been widely used, so it is described in more detail.

2.1. 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. (For a full description of rule-based deduction, see (Hayes-Roth, 1978).)

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 (Minsky, 1975), 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 (Reggia, 1982b) for a review). INTERNIST (Miller, 1982), KMS.HT (Reggia, 1982a), (Reggia, 1983a), PIP (Pauker, 1976), and IDT (Shubin, 1982) 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 (Reggia, 1983a), (Reggia, 1983b), (Nau, 1984), (Peng, 1984).

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 implementations of ARROWSMITH-P.

3.1. 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 presented in (Doerflinger, 1983).)

3.1.1. Bottom-Up Methodology

Given a homogeneous environment (i.e. one in which many similar projects are developed for the same application area), 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 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 abnormalities 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.

3.1.2. 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. 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.

3.1.3. 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 base consists 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 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 (Doerflinger, 1983). 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 (Basili, 1977), (Basili, 1984b), (Basili, 1978), (Card, 1982), (SEL, 1982) for fuller descriptions of the SEL

environment.)

The bottom-up development was performed first, and nine metrics, derived from five variables, proved satisfactory, exhibiting baselines with reasonable standard deviations. The metrics are displayed in Table 1. 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 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. During the bottom-up development, and later during the top-down development, the experts were asked to provide these relationships for all five time phases. The list of interpretations used and tested in the bottom-up and top-down expert systems is displayed in Table 2.

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. 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. See (Ramsey, 1986a) for more details concerning this implementation and research issues related to this implementation.

4. EVALUATION OF EXPERT SYSTEMS

4.1. METHODS OF EVALUATION

The four expert systems have been evaluated and compared 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

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
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

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. These forms are described more fully in (Ramsey, 1986a)

The results from the expert systems were also analyzed statistically by using a Kappa statistic test (Spitzer, 1967), (Cohen, 1968) on each interpretation. The Kappa statistic determines whether the results are better or worse than chance agreement. It was used for each interpretation in each of the four expert systems to determine whether certain interpretations are better understood than others.

In addition to testing the performance of the expert systems, an analysis 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.

4.2. 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 only agreed in about 1/3 - 1/2 of their indicated relationships overall. Furthermore, the final set of relationships for the top-down approach is very different from the final set for the bottom-up approach. 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.

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, and (4) the metrics used are not ideal.

The systems which were developed with the bottom-up approach 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% (ranging from 29% to 44% 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% (ranging from 20% to 33%) of the database conclusions. The bottom-up frame-based system agreed with an average of 16% (ranging from 11% to 20%) of the database interpretations, and the top-down frame-based system agreed with an average of 13% (ranging from 6% to 16%) of the database conclusions. It should be pointed out that each expert system produced relatively consistent results throughout its five time phases.

The results of using the Kappa statistic to evaluate the expert systems also show that the bottom-up rule-based system performed best, 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.* An interesting observation was 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. For a more detailed analysis of the results of evaluating the expert systems, see (Ramsey, 1986a).

5. 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.

The major limitation to developing expert systems for software engineering in general is 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. 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.

The bottom-up approach to developing the expert systems 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 selection of which expert system methodology to use for building expert systems is not usually clear, although an attempt has been made to provide guidelines in (Ramsey, 1986b). In 48% 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 antecedent 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. (A set of interpretations which covers all of the manifestations is *irredundant* if none of its proper subsets also covers all of the manifestations. See (Peng, 1986), (deKleer, 1986) for a full description of these ideas.)

This study has provided many additional new insights into the development of expert systems for software engineering. 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.

6. FUTURE RESEARCH DIRECTIONS

The development of ARROWSMITH-P was a preliminary attempt at constructing expert systems for software engineering management. There is certainly a need for further 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 (Weiss, 1985), (Basili, 1984a) 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.

7. 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.

This research was supported in part by the National Aeronautics and Space Administration Grant NSG-5123 to the University of Maryland. The computer support was provided in part by the Computer Science Center of the University of Maryland.

REFERENCES

Basili, V. R., 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.

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

Basili, V. R. and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation," *Communications of the ACM,* **27** (1), 42-52, Jan. 1984 (1984a).

Basili, V. R. and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transactions on Software Engineering* **SE-10** (6), 728-738, Nov. 1984 (1984b).

Basili, V. R. and C. L. Ramsey, "ARROWSMITH-P - A Prototype Expert System for Software Engineering Management," pp. 252-264 in *Proceedings of the Expert Systems in Government Symposium, October 1985, McLean, Virginia,* IEEE, 1985.

Card, D. N., 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.

Cohen, J., "Weighted Kappa: Nominal Scale Agreement with Provision for Scaled Disagreement or Partial Credit," *Psychological Bulletin* **70**, 213-220, 1968.

deKleer, J. and B. Williams, "Reasoning About Multiple Faults," pp. 132-139 in *Proceedings of the Fifth National Conference on Artificial Intelligence, August 11-15, 1986, Philadelphia, PA,* 1986.

Doerflinger, C. and V. Basili, "Monitoring Software Development Through Dynamic Variables," *IEEE Transactions on Software Engineering,* **SE-11** (9), 978-985, Sept. 1985.

Hayes-Roth, F., 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.

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

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

Nau, D. S. 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.

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

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

Peng, Y. and J. Reggia, "Plausibility of Diagnostic Hypotheses: The Nature of Simplicity," pp. 140-145 in *Proceedings of the Fifth National Conference on Artificial Intelligence, August 11-15, 1986, Philadelphia, PA*, 1986.

Ramsey, C. and V. Basili, "An Evaluation of Expert Systems for Software Engineering Management," submitted to *IEEE Transactions on Software Engineering*, 1986 (1986a). (Also available as University of Maryland Tech. Report TR-1708).

Ramsey, C., J. Reggia, D. Nau, and A. Ferrentino, "A Comparative Analysis of Methods for Expert Systems," *International Journal of Man-Machine Studies* **24** (5), 475-499, 1986 (1986b).

Reggia, J. and B. Perricone, "KMS Reference Manual," Tech. Report TR-1136, Computer Science Department, University of Maryland, College Park, MD, 1982 (1982a).

Reggia, J., "Computer-Assisted Medical Decision Making," pp. 198-213 in *Applications of Computers in Medicine*, ed. M. Schwartz, IEEE Press, 1982 (1982b).

Reggia, J., D. Nau, and P. Wang, "Diagnostic Expert Systems Based on a Set Covering Model," *International Journal of Man-Machine Studies*, **19** (5), 437-460, Nov. 1983 (1983a).

Reggia, J., D. Nau, and P. Wang, "A Theory of Abductive Inference in Diagnostic Expert Systems," Tech. Report TR-1338, Computer Science Department, University of Maryland, College Park, MD, December 1983 (1983b).

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

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

Spitzer, R., J. Cohen, J. Fleiss, and J. Endicott, "Quantification of Agreement in Psychiatric Diagnosis," *Archives of General Psychiatry* **17**, 83-87, 1967.

Weiss, D. M. 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), 157-168, Feb. 1985.