N89-15569

Artificial Intelligence Approaches to Software Engineering

James D. Johannes, PhD.
James R. Mac Donald

The University of Alabama In Huntsville
Computer Science Department
Huntsville Alabama, 35899

ABSTRACT

This paper examines the artificial intelligence approaches to software engineering. The software development life cycle is a sequence of not so well-defined phases. Improved techniques for developing systems have been formulated over the past 15 years, but pressure continues to attempt to reduce current costs. Software development technology seems to be standing still. The primary objective of the knowledge-based approach to software development presented in this paper is to avoid problem areas that lead to schedule slippages, cost overruns, or software products that fall short of their desired goals.

Identifying and resolving software problems early, often in the phase in which they first occur, has been shown to contribute significantly to reducing risks in software development. Software development is not a mechanical process but a basic human activity. It requires clear thinking, work, and rework to be successful. The artificial intelligence approaches to software engineering presented support the software development life cycle through the use of software development techniques and methodologies in terms of changing current practices and methods. These should be replaced by better techniques that improve the process of software development and the quality of the resulting products. The software development process can be structured into well-defined steps, of which the interfaces are standardized, supported and checked by automated procedures that provide error detection, production of the documentation and ultimately support the actual design of complex programs.

INTRODUCTION

Artificial Intelligence (AI) approaches to software engineering development assist in establishing a knowledge about techniques and methodologies that improve the process of software development and the quality of the resulting products. Given the task of developing a software system, what knowledge is required? To start building of a system of thousand or maybe a million of delivered lines of source code is a daunting prospect. No one should begin without a clear understanding about how the development is to be undertaken. Establishing a software development methodology when undertaking software development, on no matter what scale, is required. Every software organization already has some methodology for building

software systems. However, while some software is developed using modern software engineering techniques, most of it is still built in an ad hoc way.

The questions that an AI software engineering system can answer are, what software engineering techniques are there, which are appropriate to our problem at this stage of development, and how can we monitor the quality of the products under development? The knowledge based software engineering paradigm can be summarized as "machine-in-the-loop", where all software project activities are machine mediated and supported. The approach is to assist the programmers rather than replace them. The system acts as an active participant in the software system development process. The system must keep track of details and assist with the routine aspects of the software development life cycle thus allowing the software engineer to concentrate on the more difficult parts.

SOFTWARE ENGINEERING EXPERT SYSTEM

The knowledge based Software Engineering Expert System (SEES) environment can be loosely defined as a computer-based collection of tools, programs, algorithms, etc, which aids in the development of software and/or hardware systems during some phase of the development process. It is a collection of tools, each supporting some part of the software development process, along with tools coordinating and managing the software engineering process [6,9,12,14]. All system development life cycle activities must be machine mediated and supported by the knowledge-based environment as directed by the manager of the project. These activities will be recorded to provide the "knowledge base of software design / programming methods" of the system evolution. These will be used by the SEES to determine how the parts interact, what assumptions they make about each other, what the rationale behind each evolutionary step was, how the project satisfies its requirements, and how to explain all these to the system developers and management of the projects involved. Desirable characteristics for a SEES are:

- supports software using multiple programming languages
- support hardware development for a mixed target-machine complexes
- preserve integration with existing programs and data
- assist all project members (software engineers, managers, technical writers, secretaries, etc.)
- integrated and extensible system knowledge base
- supports component reusability
- user friendly
- supports entire project life cycle with special emphasis on prototyping
- Accommodates multiple projects

This knowledge base SEES is dynamically acquired as a by-product of the development and actual management of each project. It includes not only the individual manipulation steps which ultimately lead to an implementation, but also the rationale behind those development steps. This will make it possible to shift more and more tasks from the software engineer to the machine. To make the process possible, it is necessary to formalize all life cycle activities. In order for the knowledge base software engineering environment to begin to participate in the activities described in the life cycle of the development process (and not just merely record them) the activities must be at least partially formalized.

Formalization is the most fundamental basis for automated support. It creates the opportunity for the environment to undertake responsibility for the performance of the activity, analysis of its effects, and eventually deciding which activities are appropriate. Not only will individual activities become increasingly formalized, but so, too, will coordinated sets of them which accomplish larger development steps. In fact, the development process itself will be increasingly formalized as coordinated activities among multiple projects.

Software Engineering Knowledge Base

A formal software engineering model of definitions and rules that permit a human being to reason about the objects in the this domain, and their interrelations are most desirable, and perhaps necessary, precursor to any techniques for mechanical reasoning and problem solving in the software engineering domain. The knowledge base must contain the knowledge and understanding of the software development process subject matter and incorporate the logical aspect of human intelligence. It must be able to generate problem solutions from situations never before encountered and not anticipated by the software engineering system designers. It must be able to infer the true state of the system from incomplete and/or inaccurate measurements. The knowledge concerning each domain must, at least conceptually, be available in the knowledge base that is used by the various tool reasoning about the current state of the SEES environment.

The type of knowledge required can be divided into two parts: software engineering knowledge and application specific knowledge. The first part is conventional objects of computer science such as control constructs, arrays, sorts, structured programming techniques and their associated algorithms and implementations relationships. This knowledge is the type expressed in computer science text books such as Data Structures and Algorithms [1,15,16,18,23]. The second type is knowledge required about the world in which the target software application is to operate. The system is driven by a database of inexact and judgmental knowledge. Data (knowledge) about the problem domain may be of various forms. Some data may be applicable to the knowledge base; these are generally called (inference) rules since their function is to deduce (new) facts about the domain from the existing data. Other data may take the form of heuristics for deciding when rules or project data can be usefully applied.

The knowledge must be represented in a fashion appropriate for external use and must also be represented internally in such a way that it can be accessed, updated, and efficiently maintained. Several external representations are often desired. For example, the form in which software engineering expert presents knowledge to the knowledge base may differ drastically from the form in which the system represent this information to someone who is not a software engineering expert, a manager, or novice. For the nonexpert, the knowledge would be explained in lay terms, some aspect of the knowledge about certain objects or situations.

In conventional data processing the programmer determines all the relationships among the system modules. AI SEES environment techniques allow the environ-

ment itself to determine relationships among the software system symbols that were not made explicit by the programmer. This occurs because the environment has rules for manipulating relationships among symbols whose meanings have be represented within the program by the programmer. This manipulation of relationships among symbols is concerned with preserving not just the data provided but also the knowledge embodied in the relationships among the software elements.

Knowledge Acquisition

Knowledge acquisition is a bottleneck in the construction of SEES environment [13,14]. The SEES knowledge engineer's function is to be a go-between and assist the expert software engineer in building a system that will demonstrate the a level of expertise about the software development process. One of the most difficult aspects of the knowledge acquisition task is helping the software development expert to structure the domain knowledge and to identify and formalize the domain concepts. Potential sources of knowledge include human experts, reports, data bases, and the experience of the software engineers. The knowledge of the software engineering process is subjective, ill-codified, and partly judgmental. The process of extracting knowledge from an software engineer expert during the development process and transferring it to a computer program (expert system) is an important and difficult problem.

Software engineering knowledge acquisition involves problem definition, implementation, and refinement, as well as representing facts and relations acquired from the software development process. The software engineering expert must interact with the SEES environment to build the expertise of the expert system. The main advantages of building an expert system knowledge base are transparency and flexibility. A software engineering expert system knowledge base is developed in two main phases. The first phase is to identify and conceptualize the problem. Identification includes selecting and acquiring a software engineer expert, knowledge source, resources, and clearly defining the software development problem. Conceptualization includes uncovering the key concepts and relations that are needed to characterize the problem. What is the knowledge that software engineers know, and how can it be effectively represented in an SEES? When is the divide and conquer strategy appropriate? For the specific application is the mergesort, the quicksort or selection sort the proper implementation ? The expertise to be elucidated is a collection of specialized facts, procedures, and judgmental rules about a narrow domain area rather than general knowledge about the domain or common sense knowledge about the software development process. The following questions must be answered before proceeding with the next phase:

- What types of data are available?
- What is important in the data interrelations?
- What is given and what is inferred?
- What does a solution look like and what concepts are used in it?
- What aspects of human expertise are essential in solving software development problems?
- What is the nature and extent of "relevant knowledge" that underlies the human solutions?
- Are there identifiable partial hypotheses that are commonly used?
- How do objects in the domain relate?

- Can a diagram of the hierarchy, casual relation, set inclusion, part whole relations, etc., be built?
- What processes are involved in the problem solution?
- What are the constraints on these processes?
- What is the information flow during the software development process?
- Can the knowledge needed to solve a problem be identified and separated from the knowledge used to justify a solution?
- Are the data sparse and insufficient, or plentiful and redundant?
- Is there uncertainty attached to the information?
- Does the logical interpretation of data depend on their order of occurrence over time?
- What is the cost of data acquisition?
- How are data acquired or elicited? What classes of questions need to be asked to obtain data?
- Are the data reliable, accurate, precise; or are they unreliable, inaccurate, or imprecise?

The purpose of knowledge acquisition is to identify and obtain the knowledge needed from a particular software application to be embodied in the SEES environment which is to solve some problem in that application domain. As such, knowledge acquisition is related to the requirements definition phase of a software project. In a traditional software project it is possible to define the proposed system's requirements fully before beginning the design of the software architecture. However in an SEES project this knowledge is not readily available in the same sense as it is in a traditional software project. AI techniques are being employed largely because these techniques lend themselves well to extensive iterative acquisition and refinement of the knowledge from the software engineering and the application domain [3,4,13,14].

The problem that remains, then, is how to minimize the time needed for software system development, and how to make the software development process as effective as possible? Making the process effective involves ways to maximize the amount of knowledge acquired, to maximize the accuracy (in terms of applicability to the project) of the knowledge, and to minimize the effort needed to set up and maintain the software development process. The minimization of software development knowledge acquisition time can be encouraged by providing an environment in which changes are easy, code re-use is easy, turn-around time for changes is quick, and the system is guided down few (if any) dead-end paths during its development. To help focus on the needed knowledge and to maximize accuracy of the knowledge obtained, an early emphasis should be placed on both the overall system's eventual performance and on the knowledge needed to evaluate that performance.

Software Life Cycle Support

A SEES environment must support the software development life cycle. Which system development life cycle should the expert system support? It is very popular to view the software development procedure using the term life cycle [5,8,19,25,26]. The phase seems to be almost a fad or buzz word. There are many representations of the life cycle. Each industry has its own (or several) representations, and each of these tends to be modified somewhat for specific projects. In many cases, the concept of 'life cycle' was used in the sense of 'a suggested ordering of activities in software development, assuming ideal conditions'. This could be the art of programming by trial and error ('hacking'); a method that is unacceptable in professional

software engineering. It also covers MIL-STD-490, MIL-STD-1521A, and IEEE Standard for Software Quality Assurance Plans (Std. 730-1981) [17, 18]. Life cycle models are used to emphasize different aspects, e.g. process of development, roles of people involved, etc.. In each case, a life cycle model describes a sequence of steps which may be activities (for instance design, coding, testing, etc.), or is used to clarify the roles for management in the software production process.

A problem exists in representing the coexistence of important aspects of software development. For example, the technical production phase, the management production control, and a step into the application area such as prototype experiments can not be represented. Another problem which some models try to solve is that of directed backtracking. With each phase a set of decisions is associated, of which only one is taken at a given point in time. The decisions, taken in the several phases, are not independent of each other. For both the ongoing development and the maintenance phase it is important to know which decisions belong to a specific stage and where they were taken. As yet, no one has developed a knowledge base project environment that supports a multitude of software life cycles. Most of the systems reviewed in the literature [11,14,24] have their own unique software development life cycle that would have to be integrated into the developers, particular development methodology.

Throughout the reports written on knowledge based software engineering [11,12,14,20,22], there tends to be a severe trivialization of the problem associated with the actual task of translating the specification into code. A significant and disturbing issue brought out in the Kestrel report is that it will take 3 to 6 months of practice before a competent software engineering professional can work with the Knowledge Based Software Assistant (KBSA) system [12]. It appears, based on this statement, that Kestrel Set Theoretic approach leaves much to be desired. Kestral's approach in the KBSA development over the next 3-5 years is minimal and show little thought about the tasks involved in the creation of a reliable knowledge-based system used to develop reliable software systems of the future. The Kestral KBSA system should be able to take advantage of a ten year development cycle. The progress of the hardware revolution will continue (cheaper memory, faster / smaller / interconnected machine's) allowing the KBSA to take advantage of these advances.

Attacking the problem from its high-level esoteric aspects will generate a well thought out specification. The automatic transformation of a high-level prototype into something that can be used as a real system is difficult to imagine. Unfortunately, rapid prototyping rarely discovers those hidden data structures and relationships that are necessary to make a real system operate. Even in sophisticated implementations designed to deal with each aspect of the system's operation, the end result will still rely on the software engineer to design an algorithm that does the job effectively.

People are the highest cost driver attributes in the software development life cycle [7,18]. Shortage of software engineers personnel is between 50,000 and 100,000 people. The suppliers (primarily university computer science departments) do not have sufficient resources to meet the future demand. The demand for a knowledge based SEES environment to aid people in performing their task and coor-

dinating their activities with other members of the team through the knowledge in the system is apparent.

The current life cycle paradigm arose in an era where computers were more expensive than people. There is a need to create a software life cycle paradigm based on automation of the steps within the life cycle. In life cycle models described in the literature, the production phases design, implementation, and test are considered. Phases for requirements analysis and maintenance are often missing. The management of software development is given little or no consideration is almost all life cycle models, with the exception of the first phase of the project planning. The life cycle models look like mappings of three aspects namely management, technical production, and system application or preparation of application. These may be visualize as three simultaneous line of activities, onto one sequence of activities from project conception to system use and maintenance.

Modern programming methodology addresses the difficulties of implementing the chosen application system, not of determining the right system to implement. This has resulted in a batch-oriented development cycle concept predicated on fixing the requirements prior to beginning implementation. This approach assumes that the problem can be correctly determined in detail before a solution is ever seen by the customer. While this approach has been very successful in working around the problems associated with large system implementation efforts which confounded programming teams in the sixties, it avoids the reality dealing with legitimate situations wherein the character of a good solution is itself ill-defined in advance.

The prevailing standard development practices in industry use redundant descriptions to ensure that description mismatches are detected and to guarantee that the implementation corresponds exactly to the original specifications. This serves to freeze the implementation and make it hard to change by accident. It also serves to make the implementation hard to change on purpose if the original specifications are found to be in error.

Phased SEES Development

A possible direction to take is the incremental improvement in each portion of the existing software life cycle for software development. This approach would be a conservative, evolutionary approach described in the "Software Technology in 1990's: Using a New Paradigm" [2]. Because this approach is based on existing software life cycles, the evolutionary approach is limited by any weakness of that life cycle. Existing life cycles are not considered to be good candidate for an SEES environment because of two fundamental flaws that aggravate the maintenance problem. There appears to be no technology formalism associated with managing the knowledge intensive activities that constitute the software development process. The life cycle models reviewed in literature are informal, labor intensive, and largely lack formal documentation. Information about what specific process occurred during each phase of the development and the rational behind each decision is crucial for the maintenance process. During the software development process programming skill is applied to optimize the source code. This optimization over time makes the maintenance problem harder by making the software harder to un-

derstand [21]. The increasing dependencies among the components and scattering related design decision information about the development process over time requires machine mediation.

In the SEES approach of supporting the current life cycle models, rather than making a major revision to the activities and products of the life cycle, the existing life cycle elements and their interaction are examined for possible use of knowledge based tools. Carnegie Group and Boeing Computer Services are building a knowledge based software development environment based on this approach [20]. The environment supports the software engineer and project management using artificial intelligence. The system will provide a framework in which conventional software tools can be integrated with tools based on AI. The objectives of these efforts is to increase software engineering productivity. Currently the knowledge base software engineering environments are in their infant stages of development.

Software project management has the responsibility for planning, controlling and coordinating software life cycle activities. Currently, project managers are hampered by the informal and undocumented nature of the activities, and the fragmentary, obsolete, and inconsistent data available. More effective project management requires not only improved management techniques, but also a better software development environment that captures the total project life cycle activities and the rational behind the development process for a project. The knowledge base SEES is an intelligent environment (or collection of environments) which aids personnel in performing their tasks, and coordinates their activities with other members of the team.

TRW's Distributed Computing Design System (DCDS) provides an integrated set of environments for development of real-time distributed software systems [10]. The primary focus of DCDS is to improve system reliability, software productivity, and to minimize schedule and cost risks. Unlike the work done at Kestrel, the DCDS is strongly focused on those aspects of distributed processing involving component interaction, function architecture pairing, data distribution, deadlock avoidance and system recovery. The approach to DCDS is to define the different phases of the software development life cycle in terms of different languages, with each language specifically designed to support that aspect of each life cycle process. Information is passed between these languages through a common database and interface specification. The DCDS design is currently based on five languages and methodologies, specifically designed to attack: System requirements, Software requirements, Distributed design, Module development, and Test support. DCDS has two key aspects that it shares with Knowledge-based systems: the central database that collects all documentation form requirements to code and test cases, and the use of specialized languages designed for specific problems. Knowledge based systems support both a central knowledge base and a very high level but wide spectrum language. If the DCDS languages are taken together, they form the basis of a wide spectrum language.

CONCLUSION

This paper has examined an artificial intelligence approach to software engineering. The software development life cycle has been presented as a sequence of not so well-defined phases and as such presents a major hurdle in SEES development. Improved techniques for developing systems have been formulated over the past 15 years, but shortcuts continue to be exercised in attempts to reduce current year costs. In this sense, software development technology seems to be standing still. The SEES approach will reduce the software development problem areas that lead to schedule slippages, cost overruns, or software products that fall short of their desired goals.

A knowledge based SEES approach to the software development process will someday become a reality. However, many industry practitioners are crating new problems in trying to solve old ones. The selection of a new specific life cycle model for software development has the danger of making the problem just as unsolvable after its introduction as it was before. Will new paradigms for software development give the necessary productivity gain? Will the cost of their implementation cause the total development cost to exceed that of the development via the traditional models? These knowledge base software engineering systems will not be trivial to learn. Training, on the order of weeks months will be required to achieve acceptable efficiency in the production of software systems. The results will be a higher system reliability and maintainability as well as present less risk to the system developer.

A primary difference between artificial intelligence and more traditional ADP approaches is summarized by the slogan "In the Knowledge Lies the Power." The operative word is knowledge, rather than data or processor speed. Knowledge intensive systems attempt to model the imperfectly-understood decision processes of the domain practitioner and, like the human practitioner, make decisions with less than certainty.

BIBLIOGRAPHY

1. Aho, A. V., J. E. Hopcroft, and J. D. Ullman, "Data Structures and Algorithms", Addison-Wesley, 1983.

2. Balzer, R., et al., "Software Technology in the 1990's: Using a New Paradigm", IEEE Computer, November 1983.

3. Barr, Avron and E.A. Feigenbaum, "The Handbook of Artificial Intelligence, Volume 1, William Kaufmann, Inc., Los Altos, Ca., 1981.

4. Barr, Avron and E.A. Feigenbaum, "The Handbook of Artificial Intelligence, Volume 2, William Kaufmann, Inc., Los Altos, Ca., 1982.

5. Boehm, B. W. "Software Life Cycle Factors," TRW Software Series, Jan 1981.

6. Boehm, B. W., et al., "A Software Development Environment for Improving Productivity", IEEE Computer, June 1984.

BIBLIOGRAPHY (Continued)

7. Bruce, P. and S. M. Pederson. " The Software Development Project: Planning and Management", NY: John Wiley and Sons, 1982.

8. Daly, E., "Management of Software Development," IEEE Transactions on Software Engineering, May 1977.

9. Davis, C. G. and C. R. Vick, "The Software Development System," IEEE Transactions on Software Engineers, Jan 1977, vol 3, num 1.

10. --,DCDS A Unified Environment for System Software Development", Summary Description, Volume 1, TRW, Huntsville, AL., January 1987.

11. Goldberg, A. "Knowledge-based Programming: A Survey of Programming Design and Construction Techniques", Kestrel Institute, Palo Alto ,Ca., July 1986.

12. Green, C. et al., "Report on a Knowledge-based Software Assistant", Kestrel Institute, Palo Alto, Ca., June 1983.

13. Hayes-Roth, F., et al., "Building Expert Systems", Addison-Wesley Publishing Company, Inc., 1983.

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

15. Horowitz, E., Sahni S., "Fundamentals of Data Structures", Computer Software Press, Inc., 1982.

16. Horowitz, E., Sahni S., "Fundamentals of Computer Algorithms", Computer Software Press, Inc., 1984.

17. IEEE Computer Society. IEEE Standard for Software Quality Assurance Plans, NY: IEEE, Inc, 1982.

18. Jensen, R. and C Tonies, "Software Engineering", Englewood Cliffs,NJ: Prentice-Hall, 1979.

19. Kerola, P. and P. Freeman, "A Comparison of Lifecycle Modles," IEEE Computer Society, Fifth International Conference on Software Enginerring , San Diego, 1981, pp 90-99. Siler Spring, MD: IEEE, Inc,1981.

20. --,"Knowlege-based Software Development Envrionment, Carnegie Group Inc., Augest 1985.

21. McClure, C., " Managing Software Development and Maintenance", NY, Van Nostran Reinhold Ltd, 1981.

22. Smith, D.R., et al.,"Research on Knowledge-Based Software Environments at Kestral Institute", IEEE Transactions on Software Engineering, November 1985.

23. Sommerville, I. "Software Engineering", London: Addison- Wesley Publishers Limited, 1982.

24. Swanson, E. B. "The Dimensions of Maintenance," Tutorial; Automated Tools For Software Engineering, NY: IEEE Inc, pp 240-245.

25. Teichroew, D., "Improvements in the System Life Cycle," Tutorial on Software Design Techniques, San Framcisco: IEEE, Inc, 1976 pp 64-70.

26. Zvegintzov, N., "What life? What cycle?" AFIPS Conference Proceedings, 1982 National Computer Con-ference,Houston, 1982, pp 561-568. Arlington, Va: AFIPS Press 1982.