N88-17232

APPROACHES TO THE VERIFICATION OF RULE-BASED EXPERT SYSTEMS

Chris Culbert, Gary Riley, Robert T. Savely
Artificial Intelligence Section - FM72
NASA/Johnson Space Center
Houston, TX 77058

ABSTRACT

Expert systems are a highly useful spinoff of the artificial intelligence research efforts. One major stumbling block to extended use of expert systems is the lack of well-defined verification and validation (V&V) methodologies. Since expert systems are computer programs, the definitions of "verification" and "validation" from conventional software are applicable. The primary difficulty with expert systems is the use of development methodologies which don't support effective V&V. If proper techniques are used to document requirements, V&V of rule-based expert systems is possible, and may be easier than with conventional code. For NASA applications, the flight technique panels used in previous programs should provide an excellent way of verifying the rules used in expert systems. There are, however, some inherent differences in expert systems that will affect V&V considerations.

INTRODUCTION

Expert systems are one of the most important spin-offs from the artificial intelligence research efforts. Expert systems have been around for a number of years and some applications have proven highly successful. However, despite their apparent utility and the growing number of applications being developed, not all expert systems reach the point of operational use. One reason for this is the lack of well understood techniques for V&V of expert systems.

Developers of computer software for use in mission or safety critical applications have always relied upon extensive V&V to ensure that safety and/or mission goals were not compromised by software problems. Also, software developers have learned that aggressive V&V used early in the software life cycle can dramatically lower life cycle costs and improve software quality. Expert systems are computer programs, and without V&V they will not be accepted as either safe or cost-effective solutions to problems.

Despite the clear need for V&V, considerable confusion exists over how to accomplish V&V of an expert system. There are even those who question whether or not it can be done. As some authors have suggested (Green and Keyes[1]) this has led to a vicious circle: V&V of expert systems is not done because nobody requires it. Nobody requires V&V of expert systems because nobody knows how it can be accomplished. Nobody knows how to do V&V of expert systems because nobody has done it.

This cycle must be broken for expert system applications to succeed. However, we must first understand what we are talking about when we discuss validation and verification.

DEFINING THE TERMINOLOGY

One basic problem with V&V of expert systems has been the lack of consistent definitions for both validation and verification. Partly because expert systems have their own terminology, there seems to be a tendency to consider expert systems as something more than "just computer programs". Since the development of an expert system uses new concepts such as knowledge engineers, inference engines, and knowledge representation, it would seem plausible that the meanings of verification and validation may also have changed. However, this is not true.

At the user level, an expert system *is* 'just a computer program' and this is the level that effective V&V must address. Therefore, it is appropriate to use the definitions for verification and validation that apply to conventional software. The following definitions come from the IEEE Standard Glossary of Software Engineering Terminology[2]:

Verification.	The process of determining whether or not the products of a given phase of software development meet all the requirements established during the previous phase.
Validation.	The process of evaluating software at the end of the development process to ensure compliance with software requirements.

Boehm[3] suggests more informal definitions might be:

Verification.	"Am I building the product right?"
Validation.	"Am I building the right product?"

When put in this framework, it is clear that expert systems should be both verifiable and 'validatable' in the conventional sense. If one accepts that V&V of expert systems can be done, the next question is how it should be done. As with conventional software, the key to V&V lies in the development methodology.

THE COMMON APPROACH TO DEVELOPING EXPERT SYSTEMS

Most existing expert systems are based upon relatively new software techniques which were developed to describe human heuristics and to provide a better model of complex systems. In expert system terminology, these techniques are called knowledge representation. Although numerous knowledge representation techniques are currently in use (rules, objects, frames, etc) they all share some common characteristics. One shared characteristic is the ability to provide a very high level of abstraction. Another is the explicit separation of the knowledge which describes how to solve problems from the data which describes the current state of the world.

Each of the available representations have strengths and weaknesses. With the current state-of-the-art, it is not always obvious which representation is best to use in solving a problem. Therefore, most expert system development is done by rapid prototyping. The primary purpose of initial prototype is to demonstrate the feasibility of a particular knowledge representation. It is not unusual for entire prototypes to be discarded if the representation doesn't provide the proper reasoning flexibility.

Another common characteristic of expert system development is that relatively few requirements are initially specified. Typically, a rather vague, very general requirement is suggested, e.g., "We want a program to do just what Charlie does". Development of the expert system starts with an interview during which the knowledge engineer tries to discover both what it is that Charlie does and how he does it. Often there are no requirements written down except the initial goal of "doing what Charlie does". All the remaining system requirements are formulated by the knowledge engineer during development. Sometimes, the eventual users of the system are neither consulted nor even specified until late in the development phase. As with conventional code, failure to consult the intended users early in the development phase results in significant additional costs later in the program.

So where does all this lead? The knowledge engineer is developing one or more prototypes which attempt to demonstrate the knowledge engineer's understanding of Charlie's expertise. However, solid requirements written down in a clear, understandable, *easy to test* manner generally don't exist. This is why most expert systems are difficult to verify and validate; not because they are implicitly different from other computer applications, but because they are commonly developed in a manner which makes them impossible to test!

NEW APPROACHES TO DEVELOPMENT METHODOLOGIES

From the preceding section, it should be clear that the problem is the use of development methodologies which generally do not generate requirements which can be tested. Therefore, the obvious solution is to use a methodology which will produce written requirements which can be referred to throughout development to verify correctness of approach and which can be tested at the end of development to validate the final program.

Unfortunately, it's not that simple. Some expert systems can probably be developed by using conventional software engineering techniques to create software requirements and design specifications at the beginning of the design phase (Bochsler and Goodwin[4]). However, the type of knowledge used in other expert systems doesn't lend itself to this approach. It is best obtained through iterative refinement of a prototype which allows the expert to spot errors in the expert system reasoning before he can clearly specify the correct rules.

Since it would appear that rapid prototyping and iterative development are a necessary part of expert system development, an appropriate model for expert system development might be the spiral model suggested by Boehm[5] and modified by Stachowitz and Combs[6] (Fig. 1). This model allows continued iterative development while still providing documented requirements.

Another approach would be to write most of the requirements and specification documentation after completion of the prototyping phase. In essence, the prototype would form the basis for the requirements and would act as a "living spec". This allows the knowledge engineer to find the most appropriate knowledge representation method and gain a reasonable understanding of the problem. It also requires that coding stop at the end of the design phase so the requirements can be written. This approach is outlined in figure 2 and was developed at a NASA workshop on Verification of Knowledge Based Systems at Ames Research Center in April, 1987.

Figure 1 - Boehm's Spiral Model

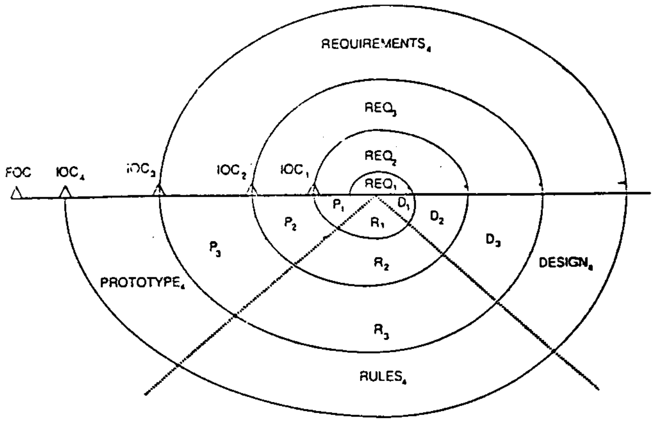
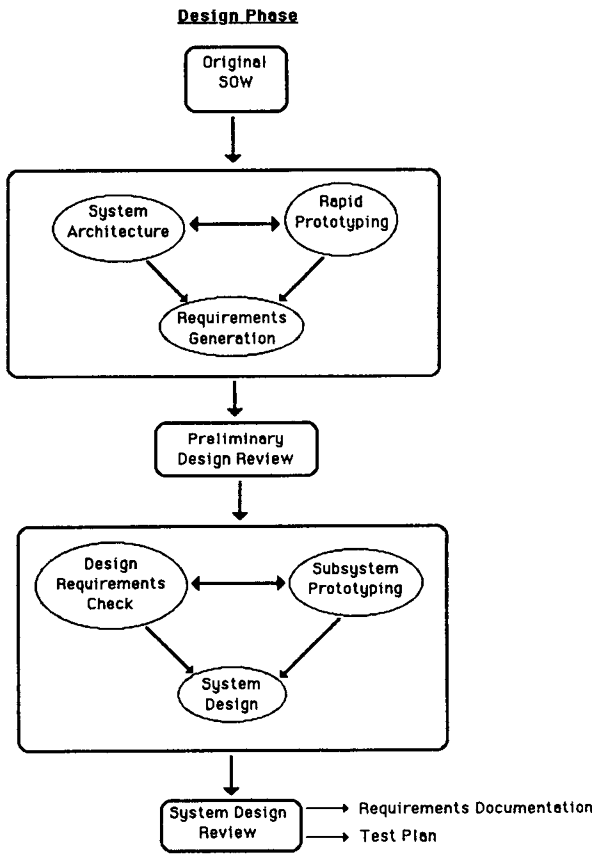


Figure 2 - Expert System Development Methodology



The approach chosen for use will probably depend on the size of the system, the complexity of the knowledge representation, and the eventual application environment. For large systems with many modules or functions it may be difficult to write all the requirements at the completion of a single prototyping phase. The spiral model would be most appropriate in this case. For smaller systems which require few iterations, the second model may be more appropriate.

MAKING THE REQUIREMENTS WORK

Once we accept that requirements and specifications must be written and a methodology for how and when to write them has been adopted, the actual work of verifying and validating the program must be done. There are some general issues which apply to any expert system and some issues that may apply to NASA expert systems in particular.

General Issues

Along with a requirements document a test plan should be written. Most of the criteria used to evaluate conventional software applies. The test plan should describe how the requirements and/or prototype will be checked for completeness, consistency, feasibility, maintainability and testability.

Some of this work can be done automatically. Testing a rule language for completeness and consistency may actually be easier than testing conventional programs. The explicit separation of knowledge elements from control and data elements may allow relatively straightforward analysis of the prototype by automated tools (Stachowitz and Combs[6]). If automated methods are not used, other standard methods such as code reviews and manual examination of the rules may also be comparatively easy, again due to the independent nature of the knowledge elements.

Feasibility of knowledge representation is usually fully tested in the early prototypes, but the feasibility of other elements of the expert system, such as performance, user interfaces, data interfaces, etc. must be verified and validated as well.

Finally, the requirements must be examined to ensure that they are able to be tested. They should be specific, unambiguous and quantitative where possible. Objective requirements will aid in the development of rigorous test cases for final validation.

Issues Specific to NASA Expert Systems

Expert systems applications for NASA programs such as the Space Station will be able to use verification techniques not necessarily applicable outside the NASA environment. These verification techniques are a direct derivative of the methods used to develop procedures, flight rules, and flight software for the Apollo and Shuttle programs. They consist of Flight Technique Panels which regularly review both the procedures for resolving a problem and the analysis techniques used to develop those procedures.

If expertise is not readily available, the analysis efforts typically use high fidelity simulations based on system models to derive and evaluate control parameters. If expertise is available through previous experience, the existing experts are reviewed by the panel and their knowledge placed in the appropriate context. The panels consist of system users, independent domain experts, system developers, and managers to ensure adequate coverage of all areas of concern. In previous programs, the typical output of such a panel was a set of flight rules describing the operational requirements for a system.

Sometimes these flight rules were translated into computer programs (typically as decision trees) and embedded in the onboard or ground computers. An additional verification step was needed to guarantee that the flight rules approved by the panel were properly coded. More often, computer limitations in the Space Shuttle caused the flight rules to remain in document form used directly by flight controllers and mission crews.

For future programs, many of the flight rules which come from the Flight Technique Panels can be coded directly into expert systems. Expert systems developed in this manner will have undergone extensive verification through the panel review. They should also prove easier to verify in code form because the rule language will allow the program to closely resemble the original flight rule.

Expert system applications outside of NASA could use this same "panel" approach. The disadvantage of using the approach discussed here is the relatively high cost of development and the need for extensive simulation capability to define unknown system characteristics. Programs of the complexity and size with which NASA regularly deals make this approach mandatory. Smaller programs may not be able to afford the resources or effort involved in verifying a system to this extent, but the size of the panel and the length of the review process can be scaled down to something appropriate for the complexity and size of the application. For some applications, the panel approach could look very similar to independent code review techniques.

Exhaustive testing through simulation remains the most effective method available for final validation. However, for any system of reasonable complexity, exhaustive testing is both prohibitively expensive and time consuming. Space Shuttle applications typically used extensive testing with data sets representative of the anticipated problems or failure modes. This method is not guaranteed to eliminate all software bugs, but it can prevent the *anticipated* problems. If used properly, representative testing can eliminate enough problems to make the software acceptable for mission and safety critical applications.

OTHER ISSUES FOR EXPERT SYSTEM V&V

So far, this paper has essentially ignored the differences between conventional software and expert systems. There are differences between these two types of software, and those differences will affect V&V efforts. Some of the differences are discussed in the following.

Verifying the Correctness of Reasoning

Verifying that an expert system solves a problem for the right reasons is sometimes as important as getting the right answer. This is particularly important for rule-based expert systems since each rule is essentially an independent module. In sequential programs, order of calculation is very easy to control and the possible paths through the program to a given solution can often be identified.

By comparison, a rule-based expert system fires rules opportunisticly and the number of potential rule combinations which lead to a solution can be combinatorially high. In such a language, identifying all possible paths to a solution is very difficult. Therefore, it is important to ensure that the expert system has gotten the right answer for the right reasons. This can be accomplished through explanations provided by the expert system or through tracing of the rule logic during execution.

Verifying the Inference Engine

The inference engine in a rule-based expert systems is a completely separate piece of code from the knowledge base. This portion of the program has rigid requirements that can be outlined and tested independently from the rest of the expert system. Often, it can be verified once and then used for multiple expert systems.

Verifying the Expert

An issue that is often raised with expert systems is how to certify the expert whose knowledge is used as the base of an expert system. For expert systems developed using the flight techniques panel method, the standard review process of the panel will ensure correctness of the experts approach in the final rules.

For expert systems developed in other manners, the question is automatically resolved as long as the expert system is validated. The entire purpose of validation is to ensure that the expert system meets all original requirements, including correctness of solution. If the expert system fails to meet these requirements, then one of three things has happened; the knowledge engineer has incorrectly coded the expert's knowledge, the expert has incorrectly described how he arrives at a solution (or does not understand it himself), or the expert's method of determining the solution is incorrect (in which case he probably isn't really an expert!). Any of these problems will be detected by the validation process and hopefully corrected.

Real-Time Performance

Expert systems which must provide guaranteed performance in real-time environments are another area that has been questioned. Most conventional programs provide performance "guarantees" through extensive simulation of the expected performance environment. Expert systems can provide the same kind of performance "guarantees". It might be more appropriate to regard these "guarantees" as upper limits which will not be exceeded for any permitted inputs.

Less often, some kinds of conventional programs are analyzed at the machine instruction level to specifically determine the amount of time required to process a given data set. Achieving the same kind of capability in a rule-based expert system is more difficult. Examining a rule-language at the machine instruction level would be both laborious and time consuming. However, as with conventional code, it can be done for a given data set entered in a specific sequence.

Complex Problems with Multiple Experts

Although the majority of the expert systems currently being developed use expertise from a single, restricted domain, it is likely that expert systems will be developed which combine the expertise of multiple experts from multiple domains. This could lead to systems which produce answers beyond the capability of any one person to evaluate.

The panel review method already discussed for NASA applications is clearly the appropriate method for resolving a problem of this type. The review process used by the panel will allow inputs from any number of domain experts and will also establish the methods of validating system responses.

Traceability of Requirements

A key part of verification is the process of tracing each module or functional element of a program back to the requirements. This process helps guarantee that the program will solve the basic problem and have the desired characteristics. It also prevents unnecessary code or features.

However, tracing requirements after they have been coded in rules may be more difficult than for conventional code. Some requirements may require multiple rule firings or the interaction of many elements in the program to achieve the desired result. Some rules may be general in nature and therefore support multiple requirements. Specifically identifying which rules support which requirements may be difficult.

This problem can become even more difficult when hybrid representation techniques are used, i.e. when both rules and objects are used to satisfy the program's requirements. Tracing requirements through a combination of representation schemes could conceivably be very difficult. Clearly, this is an area that needs some work. The complexity of this issue may even preclude the use of hybrid tools in critical applications.

Verifying the Boundaries of the Expert System Domain

A problem common to most expert systems is the brittleness of the system near the boundaries of the problem domain. It is not difficult to design an expert system which recognizes when a problem is completely outside the bounds of it's domain. It is more difficult to develop expert systems which are able to handle problems which are right at the boundaries of it's domain. That is, problems which the expert system partially recognizes, but does not have all the information needed to solve. For safety or mission critical applications, the expert system must fail gracefully (e.g., fail-safe).

Verifying that the expert system handles such situations properly could be difficult. The boundaries of a problem domain are often somewhat fuzzy. Problems which fall on the boundaries may be best recognized during testing rather than identified early in development. V&V of an expert system must be carefully aimed at identifying these boundaries if the experts can not readily do so. V&V must also ensure that the expert system fails gracefully in these circumstances.

CONCLUSIONS

Verification and validation of expert systems is very important for the future success of this technology. Software will never be used in non-trivial applications unless the program developers can assure the users/managers that the software is reliable and generally free from error. Therefore V&V of expert systems must be done. Although there are issues inherent to expert systems which introduce new complexities to the process, verification and validation can be done. The primary hindrance to effective V&V is the use of methodologies which do not produce traceable, testable requirements. Without requirements, V&V are meaningless concepts. For NASA applications, an extension of the flight technique panels used in previous programs should provide very high levels of verification for expert systems.

REFERENCES

1. Green, C. and Keyes, M., "Verification and Validation of Expert Systems", Workshop on Knowledge Based System Verification, NASA/Ames Research Center, Mountain View, CA., April 15-17, 1987.

2. IEEE Standard Glossary for Software Engineering Terminology, IEEE Std. 729-1983, Los Alamitos, CA., 1983.

3. Boehm, B.W., "Verifying and Validating Software Requirements and Design Specifications", IEEE SOFTWARE JOURNAL, January 1984.

4. Bochsler, D.C. and Goodwin, M.A., "Software Engineering Techniques Used to Develop an Expert System for Automated Space Vehicle Rendezvous", Proceeding of the Second Annual Workshop on Robotics and Expert Systems, Instrument Society of America, Research Triangle Park, NC., June 1986,

5. Boehm, B.W., " A Spiral Model of Software Development and Enhancement". ACM Software Engineering Notes, March 1986.

6. Stachowitz, R.A. and Combs, J.B., "Validation of Expert Systems", Proceedings Hawaii International Conference on Systems Sciences, Kona, Hawaii, January 6-9, 1987.