The Safety Implications of Emerging Software Paradigms

G. J. Suski
W. L. Persons
G. L. Johnson

This paper was prepared for submittal to the
International Federation of Automatic Control Safety, Reliability and Applications of
Emerging Intelligent Control Technologies '94
Hong Kong
December 12-14, 1994
October 1994

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.
DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
THE SAFETY IMPLICATIONS OF EMERGING SOFTWARE PARADIGMS*

Gregory J. Suski, Warren L. Persons, and Gary L. Johnson

Computer Safety & Reliability Group
Fission Energy and Systems Safety Program
Lawrence Livermore National Laboratory

Abstract: This paper addresses some of the emerging software paradigms that may be used in developing safety-critical software applications. Paradigms considered in this paper include knowledge-based systems, neural networks, genetic algorithms, and fuzzy systems. It presents one view of the software verification and validation activities that should be associated with each paradigm. The paper begins with a discussion of the historical evolution of software verification and validation. Next, a comparison is made between the verification and validation processes used for conventional and emerging software systems. Several verification and validation issues for the emerging paradigms are discussed and some specific research topics are identified.

Keywords: Artificial Intelligence, Knowledge-Based Systems, Neural Networks, Genetic Algorithms, Fuzzy Systems, Safety-Critical, Software Safety

1.0 INTRODUCTION

The increasing use of new software paradigms (e.g., knowledge-based systems, fuzzy logic controllers, and neural networks) in non-safety-related systems is now leading to the use—and proposed further use—of these new software paradigms in the development of safety-critical systems. This trend concerns software safety and reliability experts since methods for assuring software safety, even for well understood conventional programming approaches, are still considered developmental and problematic.

1.1 What is the Problem?

Many risks are involved in the development of safety-critical software applications. Developers must build safe and reliable software products that satisfy the requirements allocated to the software portion of the system, while others must provide assurance that this software is safe and reliable. Unless carefully managed, the complexities and uncertainties inherent in these procedures can cause unnecessary risk, delay, and expense to both the developers and the users of the software.

A major concern regarding these emerging paradigms is that software development practices are immature, continue to evolve and change, and often do not support effective software verification and validation (V&V) activities. This situation creates fear in developers, users, and the general public. Where does this fear come from? How do we assess these new paradigm-based applications? How do we mitigate potential problems? Many of these concerns are rooted in the historical experiential approaches used and the results associated with these paradigms, as well as their interrelationships with research in the discipline of artificial intelligence (AI). An historical perspective is presented below.

1.2 Historical Perspective

The Evolution of the Software Development Process. Historically, software has been produced without a clear written document describing what the software is to do. A lack of discipline in the software develop-
opment process consumed resources and caused considerable customer dissatisfaction. Some organizations discovered that having a requirements specification prior to writing code led to a better product and reduced costs. However, surprises still occurred, so software testing was made more formal and more meaningful since testing was performed against software requirements specifications. The software requirements specification (SRS) also improved as ambiguous and untestable elements were removed. Still, the results tended to contain undesirable "features" and defects that were discovered only during use.

Discipline in software development increased and the notion of "design and test" was introduced into the software development life cycle. Eventually, software development was recognized as an "engineering" discipline and the term "software engineering" was introduced. Quality assurance issues were addressed as reviews and inspections slowly made their way into the software development process. Configuration management was added as software engineering matured and the notion of a software life cycle became commonplace, with activities such as planning, requirements, design, implementation, test, installation, operation and maintenance, and retirement.

Research in Artificial Intelligence. Research into the topic of making computers more like humans began in the 1950s and evolved into a formal discipline known as artificial intelligence (AI). Several definitions of AI appear in the literature, but one that typifies the discipline is found in (Barr, 1981): "Artificial intelligence (AI) has been defined as that part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit the characteristics we associate with intelligence in human behavior—understanding language, learning, reasoning, solving problems, and so on."

The field of AI is enormous, but the major topics of interest are captured in the following taxonomy: search techniques, natural language processing, problem solving, machine learning, machine vision, pattern matching and recognition, decision support and reasoning, logic (inferenceing), reasoning under uncertainty, and knowledge-based systems.

Until recently, many people believed that AI researchers were from the "dark side" of computer science and the application of AI was for the non-mainstream computer scientist. This group believed most of the resources allocated to AI research were wasted since there were no practical uses for this unfamiliar technology.

However, during the 1980s, AI became one of the hottest areas of computing research since the integrated circuit. Many people credit this change to the increasing success of knowledge-based (expert) systems used in various applications (ranging from financial analysis and forecasting to medical consulting and diagnostic applications), the Japanese commitment to use AI technology, and the very slowly increasing use of AI techniques in conventional programs such as word processors, device controllers, spreadsheets, and optical character readers.

2.0 The Emerging Paradigms

Several emerging software paradigms are gaining popularity in many application domains, including those that are safety-critical. These include the use of knowledge-based systems (KBSs), neural networks (NNs), genetic algorithms (GAs), and fuzzy systems. Each paradigm is introduced below.

A knowledge-based system (KBS) is a class of artificial intelligence programs that replicates human expertise in a narrowly defined problem domain. They are important and valuable in that they can capture "rules of thumb" or heuristics that a human expert uses to solve problems. These heuristics are the result of much experience on the part of the human expert and are often difficult to capture and represent. KBSs have the additional capability to reason, infer, and make decisions from incomplete information.

Artificial neural networks, more commonly called neural networks (NNs), are interconnected, parallel information processing units based on a mathematical model of the human brain—how it functions and how it is organized. NNs "solve" or classify a new problem by essentially matching the new problem's characteristics with knowledge of existing problems. However, NNs differ from KBSs in two important ways: (1) KBSs use rules to generate output while NNs use information gained from previous examples (patterns) to generate outputs, and (2) KBSs require complete information, while NNs can produce results from noisy, incomplete information.

Genetic algorithms (GAs) blend both AI search techniques and classical probabilistic approaches and are gaining attention in the field of optimization. GAs use operations found in the biological discipline of natural genetics to guide them on their search through the problem's solution space.

Fuzzy systems are very similar to knowledge-based systems, but almost always incorporate an important capability to deal with uncertainty. Human reasoning processes involve fuzzy terms and uncertainty at times. Whenever a value judgment is made without a complete set of information, "fuzziness" has been used to help make the decision or reach a conclu-
In the field of AI, several activities occur in the development of each of these systems, including knowledge acquisition (KA), knowledge representation (KR), creating and manipulating knowledge in a knowledge base (KB), using knowledge and logic to perform decision making or inferencing activities, learning and adding to the knowledge base, and explaining how decisions or results were obtained.

Knowledge acquisition is, in essence, a learning problem for the system. KA involves creating an initial set of knowledge and augmenting the knowledge possessed by a software system as it performs its functions. This occurs when the system receives additional information from the human expert or user, or when the system creates additional knowledge through its inferencing methods.

Knowledge representation is the process of encoding real-world knowledge in an efficient format so it can be used by software systems. KR can be thought of as the analysis of the information that a human expert uses to solve problems and the structuring of this information so that it can be codified in relationships stored within the KB of a KBS. This knowledge (information) is represented symbolically in various forms and is used in the reasoning process. Some of the more common ways to represent knowledge include rule-based production systems, frame-based representations, semantic networks, and classes.

The decision making and inferencing activity consists of making deductions using an inferencing strategy and a control strategy that can be applied to traditional (monotonic) logic, non-monotonic logical systems, or fuzzy logic systems. Fuzzy information is essentially unreliable data (inexact data) from which assumptions and inferences are drawn.

An explanation facility is the capability of a software system to explain to its users and developers how it reached its conclusions by describing which knowledge was used and how it was used in the reasoning process.

2.1 Safety-Critical Software and Safety Issues

Reliability and safety are related, but not identical, concepts. Reliability can be thought of as the probability that a system fails in any way, while safety is concerned with the consequences of the failure. Both are important in safety-critical applications. When any system is controlled by a computer, the impact of the computer system on reliability and safety must be considered as part of the system design. One component of a computer system is the software, so software reliability and safety are matters of concern both to organizations developing software for safety-critical applications and to those evaluating it. However, software is only one computer system component. The other portions of the system are the users, the computer hardware, and the instrumentation (sensors and actuators) to which the computer is connected.

In this paper, the term safety-critical software is used as defined in the IEEE Standard for Software Safety (1994) which states “Safety-critical software is software whose inadvertent response to stimuli, failure to respond when required, response out-of-sequence, or response in unplanned combination with others can result in an accident. This includes software whose operation or failure to operate can lead to a hazardous state, software intended to recover from hazardous states, and software intended to mitigate the severity of, or recover from, an accident.”

2.2 What System Functions are Appropriate for Emerging Paradigm Components?

A key element in the design of safety-critical systems is the proper allocation of functions between humans and machines. Rasmussen (1981) defines three types of human behaviors: skill based, rule based, and knowledge based. This classification provides a basis for considering human–machine functional allocation. Machines operate more consistently than humans. Therefore, if consistency is important, as it is in safety-critical systems, functions should be assigned to machines whenever they behave at least as well as humans. Conventional machines consistently perform rule-based behaviors because they are imperative in nature, i.e., they describe a sequence of steps necessary to solve a specific problem. Furthermore, conventional machines are deterministic in that the same set of inputs always produces the same outputs. These virtues, however, make conventional systems completely incapable of knowledge-based actions. Additionally, while skill-based behavior might be broken down into procedural steps, many skill-based behaviors, such as riding a bicycle, are too poorly understood or too complicated to be translated into procedural rules.

Systems that incorporate emerging paradigms offer the possibility for machines to exhibit skill- and
knowledge-based behavior. Such systems promise considerable advantages. They can be more reliable than humans, less expensive than humans, and implement functions more simply and less expensively than procedural machines.

On the other hand, emerging paradigm systems have properties that disturb safety engineers. Such systems may be neither deterministic nor locally stationary, i.e., the transformation they perform on inputs may not completely depend on a limited set of previous inputs and system states. It may also be more difficult to ensure that emerging paradigm systems have completely covered the possible system input spaces. Neural networks and geometric algorithms, for example, are heuristic systems that are "trained" by observing a finite set of input conditions, but the rules for selecting an adequate set of training inputs are not yet clear. Acceptance of conventional software systems in highly critical systems (nuclear power plant protection, for example) is based on much more than software testing with a finite number of test cases. It is difficult to envision the safety community accepting heuristic or knowledge-based systems for highly critical applications at the present time.

2.3 What is Verification and Validation?

There is some consensus among software engineers that the activities of verification and validation (V&V) are focused on determining whether the software performs its intended function and has the required quality attributes. A large body of opinion agrees that V&V is one of the techniques used to help identify, assess, and manage risks in software development projects and can be carried out at varying levels of rigor, depending on the nature of the application and the risks involved in the development activity. The utmost rigor is required when one of the risks is safety.

In essence, verification asks the question, "Are we building the system right?" and validation asks the question, "Did we build the correct system?" In general, the V&V process comprises two distinct types of activities: (1) software analysis (reviews, inspections, and walkthroughs), and (2) software testing. Together they form a process that encapsulates the management and technical activities to analyze and evaluate all aspects of the software development process. As used herein, analysis is defined as an examination for the purpose of understanding and evaluation is the determination of fitness for use (IEEE 1074). For more detailed information on safety, reliability, testing, and V&V see (Lawrence, 1992; Persons, 1994a; Scott and Lawrence, 1994; and IEEE 1012, 1986).

3.0 COMPARING CONVENTIONAL AND EMERGING SOFTWARE PARADIGMS

There are many similarities among the emerging paradigms and conventional software systems, but there are also many startling differences. As these paradigms begin to move into the domain of safety-critical software, the issues of software testing, V&V, and software quality assurance take on much greater importance. To use these emerging paradigms in safety-critical software applications, significant evidence must be unearthed to support the claim that they will not contribute to a catastrophic failure. The state of the practice for the development of systems using these emerging paradigms does not typically include the use of disciplined V&V activities. Many components of software V&V activities exist, but because of the infancy and fragmentation of these paradigms, the V&V activities have not been assembled into an integral process.

Several issues to be addressed include, but are not limited to, the following: (1) What is the correct development process? (2) How are users' needs determined? (3) What components of these systems need to undergo V&V? and (4) Was the development supported by sound V&V activities?

John Rushby makes the following points in the SRI International Report entitled, *Quality Measures and Assurance for AI Software*, "The best way to develop credible and effective quality assurance and evaluation techniques for AI software will be to identify the facets of such software that are inherently, or essentially, different from conventional software, and to distinguish them from those facets that are only accidentally or inessentially different. Inherent differences demand the development of new techniques; accidental differences—those due simply to culture, history and bias—require only the adaptation of established techniques (and possibly the elimination or reduction of those differences)."

The application of testing and evaluation techniques is a function of "how expert" the systems are. Rushby identifies two qualities for evaluating AI software—competency and service. Competency requirements involve the "Dimensions of the overall requirements that concern 'knowledge' or appeal to comparison with human skills." Service requirements can be specified in a similar fashion to most types of conventional software. This consists of the following types of characteristics: input and output requirements, help (explanation facilities), interfaces to other systems, required performance, and throughput. Depending on where an application falls on the spectrum of "how expert" it is, the emphasis shifts from competency to services as the dominant force. It should be noted that as this balance is
achieved, any application—especially safety-critical software—must be both competent and serviceable. The only real difference is the relative focus of the V&V effort as a function of the expertise embedded in the application and the safety and reliability expected from the application.

The software development paradigm used to develop conventional software systems is different from the paradigm typically used for development in emerging paradigm software systems. This difference is summarized by the fact that emerging paradigms tend to use an iterative and/or recursive development process, while conventional systems tend to use a more linear development process. This fact has a substantial impact on the V&V activities associated with each software development process. Emerging paradigms have some similarity to conventional software used in safety-critical systems, but simultaneously have significant differences from conventional software systems. Some major characteristics of each type of software system are summarized below.

3.1 Characteristics of Conventional Systems

Conventional software systems are typically deterministic in nature, i.e., given the same set of inputs during a cycle, they produce the same set of outputs using a well-defined sequence of steps. Conventional software systems are also inherently imperative (procedural) in nature, i.e., conventional programs describe a sequence of steps used to obtain the solution to a specific problem. In particular, conventional software represents mostly “how to” knowledge. The software systems are generally created using a traditional life cycle model and are typically documented with a software requirements specification, a software design description, and are implemented using a third- or fourth-generation language or through evolving CASE tool environments, visually based or otherwise.

3.2 Characteristics of Emerging Systems

In the most general terms, the emerging paradigms are AI-based systems that attempt to simulate as closely as possible the behavior of a highly experienced and very knowledgeable human problem solver. Emerging software paradigms typically perform in human-like fashion, i.e., they are not procedural (imperative) in nature, and they are declarative and non-deterministic in that the order of the inputs is sometimes important and can result in different recommendations and decisions being reached. Emerging software paradigms typically use declarative knowledge and paradigms that describe the declarations of the solutions rather than the solutions of the problems themselves. Some well-known declarative paradigms are logic programming, object-oriented techniques, rule-based approaches, and inheritance.

These systems typically contain a knowledge base composed of factual information, problem-solving strategies, “rules of thumb,” and other types of domain-specific data which must be verified for consistency, correctness, and completeness. This information is codified and is accessible to a logic mechanism housed in an “inference engine” that performs operations on the knowledge base to reach conclusions and to make decisions.

4.0 EMERGING PARADIGM V&V ISSUES AND CHALLENGES

Some issues regarding the emerging software paradigms and software created within these paradigms are software life cycles, insufficient software documentation, complex knowledge bases, reasoning under uncertainty, V&V activities, and software testing activities.

The Software Life Cycle Problem. One possible software life cycle used for conventional software development is the traditional waterfall model; another is the spiral model publicized by Boehm (1988). Many AI software developers reject the traditional specify-design-implement paradigm used by many conventional software development processes because they claim that their problem structure is low and software requirements are typically not known in advance. Instead, they propose to use a prototyping paradigm that permits and supports the use of heuristic or evolutionary strategies to address their ill-defined problems. One such process can be developed from a modification to the spiral life cycle model.

Developers of safety-critical software systems must use a software life cycle model that contains well-defined steps to which V&V activities can be applied. This is true despite differences with conventional software development processes. As with conventional software systems, the verification activities should be integrated with the development effort to find errors at the earliest possible time. Note that many components of the emerging software paradigms are actually conventional software systems, e.g., the user interface, the inference engine, the support tools, shells and utilities, and the proposed knowledge representation schemes.

The Software Requirements Specification Problem. The state-of-the-practice in the emerging paradigms is that there is almost never any form of documented software requirements for the software system being
constructed. The heart of any safety-critical conventional software project should be a software requirements specification. As described in (IEEE 830, 1984), the requirements specification serves as a basis for the V&V activities to be performed during software development.

What is the problem? Green (1988) points out that the documentation of software requirements for KBSs would improve the testability, correctness, and reliability of these systems. However, the conventional software requirements methods would be perceived by AI software developers to impose unacceptable constraints and demands as to how the software is developed.

**The Knowledge Base Problem.** The most difficult part of the V&V process for some of the emerging paradigms is the verification of the knowledge base. A KB requirements specification should be written and maintained for use in the V&V process. Moreover, it is recommended that formal methods be used to verify the KB. For additional information on this topic see (Williams, 1992; Persons, 1994b).

Some problems associated with the KB concern the structure, syntax, and semantics of the knowledge contained therein. The task is to verify that the knowledge is complete, consistent, and can be used to reach correct judgments and conclusions. In verifying the consistency quality of the KB, one should establish that there are no redundant rules, conflicting rules, subsumed rules, circular rules, unnecessary "if conditions" or illegal attribute values. When addressing completeness, one should establish that there are only referenced attributes, reachable conclusions, no dead ends, and that all portions of the problem domain are covered.

**The Verification and Validation Problem.** One of the most critical activities in the analysis and evaluation of emerging paradigm systems is gathering convincing evidence that the system is reliable for the intended safety-critical application. Part of the V&V dilemma for these systems is that AI software developers tend to define the processes of V&V differently from conventional software engineers. As a case in point, Zlatareva and Preece (1994) define verification as the activities intended to assure that the KBS is structurally correct, and validation as the activities intended to demonstrate the KBS's ability to reach correct conclusions.

In addition to this, O'Leary (ND) defines knowledge validation as analyzing the KB and decision-making capability of the system. This process requires four steps: (1) determining what the system knows, does not know, or knows incorrectly, (2) determining the expertise level of the system, (3) ascertaining if the system is based on a decision-making theory that is generally well accepted in the specific problem domain, and (4) ascertaining the reliability of the system.

However, the developer thinks of the V&V activity as a non-binary decision, i.e., a system is neither absolutely "perfect" nor absolutely "imperfect." Since these emerging paradigm systems model the problem-solving capability of a human expert, their performance is human-like—namely, not perfect, but in many cases "good enough." In all cases there is risk associated with the evaluation of the system. This is true for both conventional and emerging paradigm systems, and consists of well-defined statistical errors that address the hypotheses related to accepting a system as valid or declaring a system invalid. These are known as Type I and Type II errors in the discipline of statistical inferencing. A Type I statistical error means that a decision error was made in the validation process in that a valid system was rejected as being invalid. A Type II statistical error results if the system is accepted as valid when it is really invalid. O'Keefe et al. (1987) call the probability of a Type I Error the "builder's risk" and the probability of a Type II error the "user's risk." Type I errors are not particularly serious in safety-critical software since they only increase the system builder's cost. However, Type II errors can have serious consequences in safety-critical software, so the user's risk must be reduced for safety-critical applications. These concerns should be carefully addressed in the V&V activities of both conventional and emerging paradigm systems.

**V&V Challenges.** Some special concerns should be addressed for knowledge bases: (1) assessing the access adequacy of the knowledge representation structure, (2) comparing the system with the human expert's knowledge, (3) performing an analysis of the completeness and consistency of the KB, and (4) conducting a verification of the reasoning process.

Experience based on the V&V of conventional safety-critical software systems suggests the following major challenges are associated with the V&V activity for systems constructed using emerging software paradigms:

1. What to V&V? Any intermediate result(s), the final result (conclusion), the reasoning of the system, and all combinations of the above.
2. What to V&V against? Use known results and human expert judgment, the software requirements specification, and the KB requirements specification.
3. What to V&V with? Combine review, analysis, inspections, and testing techniques. Test cases should be drawn from several experts, taken from many similar problems in the specific domain, and test cases should be selected by ran-
dom sampling techniques within each identifiable result class.

(4) When to V&V? It is important that systems exhibit acceptable performance early in the development process. This implies there should be a well-defined development software life cycle and that V&V activities should be planned, performed, and documented throughout the development process using a V&V plan (IEEE 1012, 1986).

5.0 KNOWLEDGE-BASED SYSTEM PARADIGM

Knowledge-based systems are used to perform a variety of complicated activities that until recently could only be performed by a limited number of highly trained human experts. One of the most interesting and powerful characteristics of KBSs is their capability to apply human "judgment" and "intuition" in the problem-solving activity. KBSs have been used in a wide variety of application domains ranging from planning, status monitoring, control, and diagnosis to instructional assistance.

KBSs are first and foremost software systems. From a functional perspective, they are software systems that exhibit human-expert-like performance in a particular discipline or field called a problem domain. KBSs are intended to solve problems in small, unstructured domains where knowledge rather than procedure is used by the human expert. KBSs share the following capabilities and characteristics: (1) solving difficult problems as well as or even better than some human experts, (2) reasoning using what human experts consider to be effective heuristics, (3) interacting with human users in appropriate ways, including natural language, (4) manipulating and reasoning about symbolic descriptions, (5) reasoning with data which contain errors and using rules which can handle uncertainty, (6) contemplating multiple, competing hypotheses simultaneously, (7) rationalizing why they are asking for specific information or asking a given question, and (7) justifying their conclusions using an explanation facility.

5.1 Components of a Knowledge-Based System

A KBS is based on artificial intelligence concepts that can be thought of as "machine problem solving." They contain expert knowledge of the specific problem domain in a KB, which allows the KBS to "reason" about specific problems, accommodate new knowledge as it evolves, follow various lines of reasoning, use fragmentary and heuristic knowledge, make judgments, and provide advice and guidance to the user of the KBS.

5.2 Knowledge-Based System Specific V&V Issues

The process of determining whether or not a KBS is robust, reliable, and accurate can be divided into two main activities: (1) evaluating the reasoning mechanisms used to determine that they accurately and correctly interpret and apply the data in the KB to solve domain-specific problems, and (2) determining that the KB is complete, consistent, and correct. Experience has shown that for both conventional software systems and KBSs, it is helpful to perform both qualitative and quantitative V&V activities. Several qualitative and quantitative V&V techniques appropriate for knowledge-based systems are adapted from O'Keefe (1987), and each technique has potential to improve the quality of the fielded KBS:

Qualitative V&V Techniques. These techniques include (1) face validation, (2) predictive validation, (3) Turing test, (4) field testing, (5) subsystem validation, (6) sensitivity analysis, and (7) visual interaction.

Quantitative V&V Techniques. Quantitative V&V activities use classical statistical techniques to compare the performance of a KBS with the human expert or various test cases with known results. Standard statistical tests can be applied to the validation portion of the V&V activity for KBSs. These include statistical tests such as (1) paired t-test, which uses a statistic to compare the difference between KBS performance and a human expert performance or known results, and (2) Hotelling's one sample T^2 test, which is a statistical technique that is most useful when the KBS produces a number of intermediate or final results.

The quantitative measures typically fit into one of two separate categories. Either subjective judgment is used to compare a KBS's performance with an acceptable performance range, or a formal statistical hypothesis test is used to compare measurements with a predetermined acceptable performance range.

6.0 NEURAL NETWORK PARADIGM

The use of neural network (NN) paradigms is increasing rapidly in both safety-critical and non-safety-critical applications. An increasing number of NN applications have been developed in several major industries, including financial, defense, manufacturing, aerospace, and business. Michal (1994) estimates what the future holds: "The use of artificial neural networks for monitoring of equipment and components in nuclear power plants could be commonplace before the turn of the century. Within five years, the relative cost of neural networks could usher in a technology that will be
used to detect incipient faults in machinery and increase effectiveness of maintenance scheduling.” Developing NN applications is currently considered an art form; however, enough experience exists to propose a preliminary software development model to be used for NN applications.

Neural networks consist of many artificial neurons connected together in a structure inspired by a model of the cerebral cortex of the human brain. Structurally, the individual processing units, called neurons, are typically organized into layers in such a way that every neuron in a given layer receives inputs from the outputs of every neuron in the previous layer.

Every connection between neurons has a “synaptic weight” that is adjusted during the “training” of the NN. Thus, a neural network is a dynamic system consisting of layers of neurons connected to each other. Each neuron’s state is determined by messages sent to it from all the neurons connected to it. These messages tell the neuron the state of each neuron sending it a message and the weight or strength of the connection among the neurons. The basic processing element of a NN is the individual neuron. Each neuron receives inputs over all of its connections, but each has only one output. A threshold function is sometimes used to help determine the output of each neuron. The value outputted from the neuron’s threshold function characterizes whether the neuron has “fired” or not. Most frequently, there are three layers in the neural network model: an input layer, an output layer, and a middle (or hidden) layer that serves as a “feature detector.”

The input layer consists of the problem to solve or the “pattern to match.” The hidden layer consists of one or more layers that represent the internal connections in the NN. The output layer presents a resulting solution or classification presented to the user of the NN system.

NN operation involves the two processes of learning and recall. Learning means that the connection weights adapt in response to external stimuli presented to the network, i.e., they adjust themselves to the stimuli. The network is trained to “learn” using the learning rule governing the adjustments for the connection weights. The recall process accepts an input and produces a response according to the NN topology and connection weights.

Neural net computing is one of the fastest-growing fields in the history of AI because NN can be trained to identify non-linear patterns that exist between input and output values, and can solve complex problems much faster than digital computers and numerical algorithms. They can learn complex non-linear relationships even if the input information is noisy, imprecise, or incomplete. NN models can be “verified” against a statistically designed set of experiments. One of the significant distinctions between conventional software and NN software is the potential to perform parallel distributed processing.

NN memory is both distributive and associative. A distributive memory implies that the storage of a unit of knowledge is distributed among virtually all of the connection weights in the network. Associative memory means that when the trained NNs are presented with a partial or distorted input, the network will choose the closest matching solution, in a least-squares sense. Since the memory of a NN is distributive, the destruction of one or a few connection weights or artificial neurons does not significantly influence the behavior of the NN, making it very fault-tolerant.

6.1 Components of a Neural Network

The components of a NN are the individual neurons, the various connections that exist within a layer, the number and topology of the layers, the learning algorithm used, the activation function selected, the training algorithms, and the selected neural network model. Note that the individual components of standard NNs are typically implemented using conventional software development practices and third- or fourth-generation programming languages.

One could argue that a statistical model could be formulated to “know” how to guess characteristics based on “known” features of the problem domain. In a simplistic way, a statistical model is an indirect way of learning a correlation. However, the advantages that NNs provide are that they can analyze data presented to them and determine which data is relevant. The irrelevant data has very low connection strength to the output neurons so that they are effectively removed from influencing the output neuron. An additional advantage of NNs is that they have many factors with which to experiment.

6.2 Neural Network Specific V&V Issues

Since a NN application is essentially a system made up of conventional computer subsystems, much of the development process parallels that used in conventional software and KBSs. Like KBSs, the development process for NNs requires experimentation, iterative refinement of network parameters, problem redesign, and reformulations—specifically, refining a general approach to a specific solution. During the V&V activities, special attention should be focused on the learning and recall processes used by the NN-based system.
Questions to Consider:
- How many patterns will be input to the network?
- How many neurons are in each layer?
- How do we describe error and learning rates?
- How do we describe connection weights?

An approach to validating a NN application is to compare its characteristics to other successful NN applications. For example, (1) Is the application data-intensive and dependent upon multiple interactive parameters? (2) Is the problem domain rich in historical data or examples? (3) Is the data set incomplete? and (4) Does the data set contain errors? A NN is potentially useful when these questions are answered in the affirmative. However, for safety-critical software system applications, all of the conventional safety-critical software V&V activities should be applied to the NN-based system.

7.0 GENETIC ALGORITHM PARADIGM

Genetic algorithms (GAs) can be used as part of machine learning, search, or optimization applications. A GA emulates classical biological evolution theory as it attempts to solve problems. As part of the search for solutions, a GA maps a given problem onto a set of (binary) strings. The set of strings forms a population in which each of the strings represents a potential solution to the problem at hand. The GA then manipulates these populations and evaluates each potential solution. Finally, a GA searches for the “best” candidate solutions and uses them to create new generations of potential solutions. GAs contain properties that make them inviting as a technique to select high-performance membership functions for fuzzy systems or to use in the training process for NNs. Also, GAs can be used to facilitate the derivation of the optimal fuzzy relationship matrix and fuzzy membership function (Park et al., 1994).

GAs may be characterized as follows: (1) GAs use a scheme to encode solutions to the problem; (2) GAs use an evaluation function to rate each potential solution relative to the others in the current set of solutions; (3) An initialization procedure generates the first set of potential solutions; (4) A set of operators is used to manipulate the composition of the potential solutions; and (5) A set of parameters specifies the initial settings for the algorithm and operators, as well as the algorithm’s termination condition (Montana and Davis, 1989).

GAs have their own terminology based on the biological genetic model of life. A chromosome is a candidate solution and consists of a linear list of genes, where each gene can assume a finite number of values. A population consists of a finite number of chromosomes or potential solutions to the problem at hand. The GA evaluates a population (the current set of candidate solutions) and generates a new one iteratively, with each successive population called a generation. In nature, individuals best suited to competition for scarce resources tend to survive, and adapting to a changing environment is essential for the survival of members of each species. In a GA, each solution has associated with it a measure of “goodness,” called its fitness value. This allows comparisons to be made with other solutions in the current generation. Recombination of genetic material is simulated through a mechanism that exchanges portions of genetic material between chromosomes. An additional operation causes random alteration of the genes in the chromosomes.

In a GA, each new generation is created using three basic genetic operators: reproduction, crossover, and mutation. Reproduction is a process by which the most highly rated chromosomes in the current generation enter into the “mating pool” to await reproduction (that is, copying to the new generation). Crossover provides a mechanism for chromosomes (potential solutions) to mix and match attributes through random processes. Mutation is the sporadic alteration of some gene values within a chromosome. Every gene in each chromosome is a candidate for mutation, and the candidates for mutation are selected by the mutation probability.

7.1 Components of a GA

The components of the simple GA described above include a population of binary strings, control parameters, a fitness function, genetic operators, a selection mechanism, and a mechanism to encode the solutions as a binary string.

7.2 GA Specific V&V Issues

GAs are based on the natural genetic model, but are almost always implemented using conventional software development practices. Thus, all of the traditional software V&V activities and testing activities apply to the individual components of the typical GA—namely, the population initialization procedure, the population evaluation procedure, the next-generation selection procedure, the crossover procedure, and the mutation procedure. GAs therefore share many attributes of conventional software systems and, from a V&V perspective, can be treated similarly to conventional software.

Special attention should be paid to determining the techniques used for the encoding process that maps the problem domain to the chromosomes in the ini-
8.0 FUZZY SYSTEM PARADIGM

Many of the KBSs are developed in an attempt to solve problems for which conventional software systems have proven inadequate. Available data in these problem domains is sometimes "noisy," incomplete, imprecise, and uncertain. The notion of "dealing with uncertainty" comes into play when the knowledge base is incomplete, inconsistent, or changes over time. Driving the use of the emerging paradigms for safety-critical software is the fact that the reasoning that must be performed is often based on approximate or inexact evidence.

In such a system, the knowledge that is used comes from multiple sources and has various levels of uncertainty. Fuzzy systems have been used to develop risk analysis techniques that have applications in several problem domains. Currently, fuzzy systems are finding applications in the commercial marketplace in appliances and video cameras.

Fuzzy systems are rule-based systems that use fuzzy linguistic variables to model the human expert’s rule-of-thumb approach to problem solving. These "fuzzy expert systems" feature rules that direct the decision process, and membership functions that convert linguistic variables into the precise numeric values needed for a software implementation on a computer system. The rule set is gleaned from a human expert’s knowledge, which is based on experience, and can usually be written more easily than the crisp rule set required by most conventional KBSs.

Thus, fuzzy systems present an additional method of modeling a problem or the characteristics of a system using fuzzy inference rules. The application of some of the paradigms and techniques discussed previously is complicated by the real-world situation that both the data and the expertise are quite often uncertain themselves. Several methods for dealing with uncertainty have evolved over the years, including plausible inference, Bayesian probability theory, certainty theory, possibility theory, and the Dempster/Shafer theory of evidence. Fuzzy systems attempt to encode the knowledge of the human expert in a computer program together with methods for reasoning. It is important to state explicitly the assumptions underlying the various models and algorithms that are used in these systems.

Uncertainty in information theory has been defined as the lack of exact knowledge that would allow a perfectly reliable conclusion to be reached in a deterministic system. Classical logic (first-order predicate calculus) allows only exact reasoning that is based on perfect knowledge of the truth or falsity of facts. In addition, it uses the law of the excluded middle—it presumes that each and every proposition is either true or false, but not both. Uncertainty and fuzzy systems, however, relax these assumptions and try to perform similar "reasoning."

Reasoning under uncertainty differs from both statistical and probabilistic reasoning. The goal of statistical reasoning is to consider properties of populations and the samples drawn from them. Probabilistic reasoning allows one to determine whether or not the samples are representative of the populations from which they were drawn.

8.1 Components of a Fuzzy System

Fuzzy systems include fuzzy sets, fuzzy logic, fuzzy inferencing techniques, and fuzzy control structures. The primary difference between fuzzy systems and more conventional knowledge-based systems is that fuzzy systems support a gradual and continuous transition rather than a discrete, binary change. The components of KBSs and fuzzy systems are quite similar. In fact, they differ mainly in that fuzzy systems have "fuzzifiers" which convert inputs into their fuzzy representatives and "defuzzifiers" which transform the output of a fuzzy logic process within a fuzzy system into the "crisp" (numerically precise) solution variable. In a fuzzy system, the values of a fuzzified input set execute all the rules in the KB that have the fuzzified input state as part of their premise.

Most fuzzy systems are designed without using a mathematical model of the underlying process. These systems and controllers are generally based on an expert’s understanding of a given process. A fuzzy system typically consists of the following: (1) a KB which contains a number of fuzzy control rules, (2) a KB which defines the fuzzy membership functions of the linguistic terms used in the rule base, (3) an inference mechanism based on the control rules, (4) a fuzzification unit to map real inputs from sensors into fuzzy terms, and (5) a defuzzification unit to map fuzzy output of the fuzzy logic inference mechanism to real numbers.

8.2 Fuzzy System Specific V&V Issues

For the purposes of software V&V, fuzzy systems are similar to the KBS, but they have been enhanced to permit the use of knowledge representation schemes and the inferencing methods to process uncertain data. As such, most of the V&V techniques discussed for KBSs apply to fuzzy systems.
However, special emphasis during the V&V activities should be focused on the use of uncertainty. At least three issues surround this topic: there may be uncertainty in (1) the incoming information that is used in the reasoning process, (2) the information or rules that are in the KB, or (3) the working facts that have been deduced thus far in the reasoning process. Additional information on various approaches to deal with uncertainty are found in (Prade, 1985).

9.0 SUMMARY

9.1 What Is Known?

Part of the concern and fear associated with the introduction of emerging-paradigm-based software systems into safety-critical applications is that many professionals do not perceive these software systems as simply computer programs. Each emerging paradigm has its own vocabulary and jargon, making it difficult to understand. Human nature suggests that these “unknown” systems should be suspect. The myriad of new terms leads many practitioners to conclude that V&V must be different for these systems. Many others disagree with this claim.

Since the emerging software paradigms are computer software programs, many of the activities of the V&V process from conventional software development techniques are applicable to the verification and validation of these systems. Granted that V&V activities should be adapted to the specific paradigm as required, software systems without V&V are neither safe nor cost-effective solutions to safety-critical applications.

One of the major problems with these emerging paradigms is that their software development practices are immature and continuing to evolve. In many cases the development processes do not support effective V&V activities. It is believed that if appropriate techniques are used to document the system and knowledge requirements in a software requirements specification, then V&V activities can be performed in ways similar to V&V activities for conventional software systems.

Table 9-1 provides a sample of emerging paradigm applications that have been discussed in recent literature. Which of these would the safety community consider acceptable today? What more needs to be known to expand the envelope of acceptable applications? Today we can only speculate about answers to these questions.

The safety community is likely to accept emerging paradigm systems in applications where an effective safety backup can be provided by operators, hardware, or dependable procedural systems. Examples of such systems might be operator advisors, control systems that are backed by safety interlocks, or anticipatory trip systems.

The safety community is not likely to accept emerging paradigm systems in applications where simple procedural or hardware interlocks are adequate to take a system to a known safe state. While emerging paradigm systems may have economic advantages in these cases (lower development costs or greater operating margins) safety engineers are unlikely to accept such systems until they are convinced that the emerging paradigm system offers dependability at least equivalent to that of conventional systems. At least two exceptions to this might be considered: emerging paradigm implementation of well defined physical rules as an alternative to procedural programming (Nilsen, 1990), or emerging paradigms that recognize only safe patterns and force the system to a safe state when an unrecognized pattern exists.

The safety community is not likely to know how to treat applications for which a safe default state cannot be defined in advance. The fire alarm system described by Nakanishi is a readily understandable example of such a system. Failure to give a valid alarm may directly lead to death, injury, or property loss. False alarms, however, cause fire fighters to be temporarily unavailable to respond to a subsequent valid alarm. Emerging paradigm systems offer the opportunity to reduce the false-alarm rate by enabling

<table>
<thead>
<tr>
<th>Application</th>
<th>Paradigm</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steam leak detection</td>
<td>Fuzzy system</td>
<td>Sutton</td>
</tr>
<tr>
<td>Reactor rod control</td>
<td>Fuzzy system</td>
<td>Sutton</td>
</tr>
<tr>
<td>Safety parameter display</td>
<td>Knowledge-based system</td>
<td>Nilsen</td>
</tr>
<tr>
<td>Fire alarm system</td>
<td>Fuzzy system</td>
<td>Nakanishi et al.</td>
</tr>
<tr>
<td>Accident identification</td>
<td>Genetic algorithm coupled to neural net</td>
<td>Guo and Uhrig</td>
</tr>
<tr>
<td>Electrical distribution</td>
<td>Knowledge-based system</td>
<td>Enns et al.</td>
</tr>
<tr>
<td>Protective relay coordination</td>
<td>Knowledge-based system</td>
<td></td>
</tr>
<tr>
<td>Reactor core protection calculator</td>
<td>Neural net</td>
<td>Koo, Kim, and Chang</td>
</tr>
</tbody>
</table>
more sophisticated criteria to be used for triggering an alarm.

9.2 What Should Be Learned?

In dealing with the emerging software paradigms, one typically finds that very few requirements are initially specified by the user or requester of these systems. In traditional software development, precise requirements definition and analysis is considered to be a necessity for safety-grade systems. Thus, in addition to getting the correct result, it is important to verify that the knowledge and reasoning used in solving a problem is correct.

Open Issues. Some open issues to be addressed when planning to use these emerging software paradigms in safety applications are listed below. This list is neither inclusive nor exhaustive. Much additional research needs to be performed to fully understand how design errors are introduced into these systems, and what techniques may be applied to produce required levels of safety and reliability.

- Does the method used for combining multiple sources of expertise and opinions improve the consistency of the knowledge base and the resulting guidance or recommendations?
- How should consistency of human expert judgment be measured, and should there be a threshold for inclusion of an expert's opinion(s) in the KB?
- Is the use of homogeneous subgroups of experts a help or hindrance in creating the KBSs?
- How can the impact on existing knowledge in the KB be identified when additional rules are added?
- How can we ensure that KBSs degrade gracefully as they attempt to use their knowledge and reasoning capability to provide conclusions and decisions even though they may be operating near or beyond the boundary of their expertise?
- How should requirements tracing be performed, since to perform a trace from the software requirements specification to the software design description to code requires that "functional units" be identified at some level?
- How should test cases be constructed, since the execution sequence of these systems is difficult to follow?
- How should V&V be performed, given that the different software paradigms have significantly different intermediate software products?
- Is this new technology; that is, the use of KBSs, NNs, GAs, or fuzziness, a way to achieve software diversity in safety-critical systems?
- Could these new paradigms offer a sufficiently reliable way to achieve fault tolerance in safety-critical applications?

10.0 CONCLUSION

Despite an agreed-upon need for V&V of systems based on the emerging paradigms, much confusion exists as to what is necessary for their verification and validation activities. Some professionals doubt that V&V activities are even possible for these systems, while others ask what should be done to perform the V&V for these systems, and how these activities should be performed.

To accept emerging paradigms in safety-critical systems, safety engineers must better understand what is gained and what is lost by their use. Several important issues must be addressed. What is the proper functional allocation between humans, emerging paradigm machines, and conventional paradigm machines? How do we ensure the completeness of the input conditions covered by the emerging paradigm systems, particularly heuristic systems? Can the non-deterministic behavior of emerging paradigm systems be bounded or probabilistically described? Can the risks and benefits of using emerging paradigm systems be described in a way that allows tradeoffs?

It may be appropriate to consider an analogy between the evaluation of emerging-paradigm software-based systems and the evaluation of conventional software systems. One perspective is that performing V&V on emerging paradigm software systems is like grading an essay examination, while performing V&V on conventional software systems is more like grading a multiple-choice examination. Both can be performed; however, the grading of the essay examination requires a much deeper understanding of the subject area, more experience in the grading process, and typically much more effort.

As part of our research, a software evaluation process has been developed that can be used to evaluate safety-critical conventional software. It is believed that this process, currently being automated on a laptop computer, can be tailored to help with the evaluation of emerging software paradigms used to create safety-critical software applications.
REFERENCES


