SAND--96-2100C SAND96-2100C
CONF-960869--17

Achieving Strategic Surety for High Consequence Software

Guylaine M. Pollock[1], Ph.D.; Sandia National Laboratories; Albuquerque, NM

Abstract

A strategic surety roadmap for high consequence software systems developed under the High Integrity Software (HIS) Program at Sandia National Laboratories guides research in identifying methodologies to improve software surety (Ref. 13). Selected research tracks within this roadmap are identified and described detailing current technology and outlining advancements to be pursued over the coming decade to reach HIS goals. The tracks discussed herein focus on Correctness by Design, and System Immunology™. Specific projects are discussed with greater detail given on projects involving Correct Specification via Visualization, Synthesis, & Analysis; Visualization of Abstract Objects; and Correct Implementation of Components.

Introduction

The development of software for use in high-consequence systems mandates rigorous (formal) processes, methods, and techniques to improve the safety characteristics of those systems. To address this need, research efforts must progress in several areas over the next few decades to allow us to reach, with greater certainty, the higher levels of reliability required by software used in high-consequence systems (Ref. 6,12). This paper describes a strategic surety program developed for high—consequence software under a new initiative at Sandia National Laboratories (SNL)—to identify how we will develop ultra-reliable software in the 2010 time frame.

This initiative, the High Integrity Software Program (HIS), is tasked with guiding strategic investments in the development of new capabilities and technologies in the domain of high consequence software at SNL. The program sponsors research within the strategic surety backbone of the defense sector to establish predictive confidence that a system is safe, secure, and under control through the exploration, extension and application of the science of software systems (Ref. 13). The program emphasizes high-risk, high payoff research through a correctness research track focused on a "correctness by design," and more immediate lower-risk, medium payoff applications research through a systems immunology™ track. This track produces methods and techniques to render today's systems safer, more secure and more reliable. (Other tracks have been defined but are not currently staffed.)

[1] This work was supported by the United States Department of Energy under contract DE-AC04-94AL85000.

The motivation behind this initiative is briefly reviewed before outlining the development of the strategic surety roadmap guiding this program. Subsequently, a more detailed discussion of several of the research projects will be presented. Projects within the Correctness by Design track and the Systems Immunologyø track will be discussed.

Motivation

The High Integrity Software Program (HIS) at Sandia National Laboratories was established to provide a crucial role in guiding internal research efforts to improve technologies that enhance surety aspects of high-consequence systems. This program strives to develop better technologies within the software industry enabling us to increase our confidence in the correctness of high-consequence systems, many of which may become life-threatening if flawed.

Examining this industry in general, we see software becoming more complex and being relied upon more often for an ever-widening variety of applications. In fact, our dependence on software is exploding quietly-"The amount of code in most consumer products is doubling every two years ... televisions may contain up to 500 kilobytes of software; an electric shaver, two kilobytes; while the power trains in new General Motors cars run 30,000 lines of computer code." (Ref. 9)-and yet software is not reliable in most systems. As a result, software irregularities, in some instances, have taken or degraded people's lives in various system accidents.

Notwithstanding, new types of applications continue to appear on the technological horizon, generating continued cause for concern regarding current abilities to evaluate software surety. For example, Andy White, Director of Los Alamos National Laboratories Advanced Computing Laboratory, has stated that an important goal for new software applications is to solve large problems (such as helping the Forest Service fight fires, helping doctors determine which flu vaccines to use, and making sure that U.S. nuclear bombs do not go off accidentally) that, in short, require us to trust computers to predict the future (Ref. 1).

While some have encouraged expansion of these types of applications, many others have cited this proliferation as a potential powder-keg for our society: "These days we adopt innovations in large numbers, and put them to extensive use, faster than we can ever hope to know their consequences ... which tragically removes our ability to control the course of events" (Ref. 11).

Even more alarming, this increase in numbers and types of software applications has increased our vulnerability as a nation to information warfare. (This is a problem for other nations as well.) In fact, last year the Joint Security Commission stated that "The U.S. vulnerability to infowar may be the major security challenge of this decade and possibly the next century" (Ref. 7). Not surprisingly, Pentagon officials have reported an attempt at such warfare was actually suggested to U.S. adversaries during the Gulf war when a group of Dutch hackers offered to disrupt the U.S. military's deployment to the

Middle East for $1 Million. If current trends continue, this type of vulnerability will only increase unless we work to ameliorate our skills in assessing software surety.
Clearly software integrity and surety (safety, security, reliability) issues are a major concern for U.S. industries; as such, they are also a concern for Sandia National Laboratories. Current surety technologies just are not good enough for industries' increasing needs.

Consequently, the HIS program initiative was formulated to address high integrity and surety software issues. Sponsors of the program include the Strategic Surety Backbone of the Defense Programs Sector and the Vice President of Defense Programs. The HIS objective is to establish predictive confidence that a system is safe, secure, and under control.

Roadmap Tracks

One of the first actions taken under this program was the establishment of a steering committee to guide research activities. After establishing the general framework of a strategic plan identifying the strategies, technologies, and capabilities needed for achieving goals of improved software surety and integrity, we began working to define stronger, more specific guidelines to identify measurable milestones and produce specific products, to meet our needs. Consequently, the steering group identified a number of roadmap tracks for development. Six different tracks have been identified to date. (This is an on-going process so we anticipate that additional tracks may be identified in the future.)

Once a track was identified, appropriate staff were assigned to aid in the further development of that track. Please note, each track was not expected to be a complete solution in and of itself. The tracks complement one another to address a complex problem. Numerous projects may be conducted with a particular track. Due to space constraints, only two of the roadmap tracks are discussed in the following sections. They are the Correctness by Design, and the System Immunologyø tracks.

These tracks are viable research directions that will enable Sandia to reach a higher level of software integrity by allowing surety to be engineered into software systems. Research efforts within the tracks will drive software development to be a much more disciplined engineering activity, supported with modeling and measurement and reuse of proven components. Several of the tracks are derived from a vision of how software will be built, verified, and understood in the future as suggested by Dalton. (His view is depicted in figure 1.)

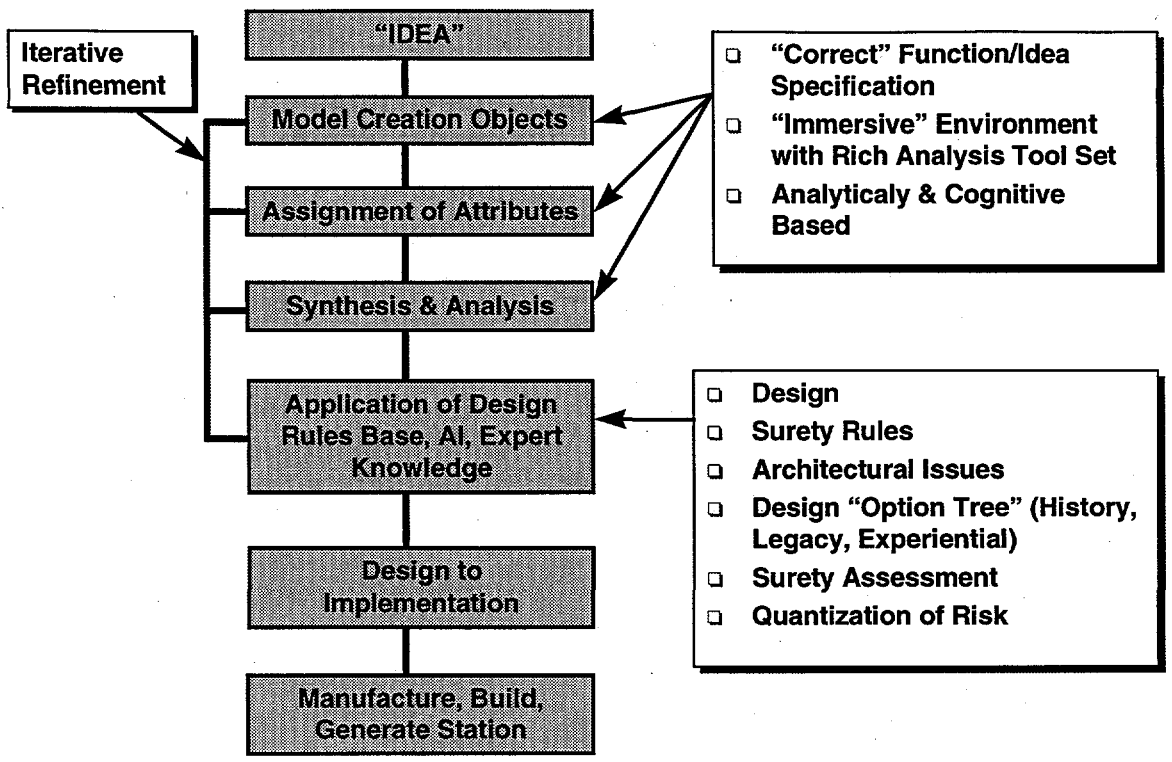
Iterative Refinement
"IDEA"
Model Creation Objects
Assignment of Attributes
Synthesis & Analysis
Application of Design Rules Base, AI, Expert Knowledge
Design to Implementation
Manufacture, Build, Generate Station
"Correct" Function/Idea Specification
"Immersive" Environment with Rich Analysis Tool Set
Analyticaly & Cognitive Based
Design
Surety Rules
Architectural Issues
Design "Option Tree" (History, Legacy, Experiential)
Surety Assessment
Quantization of Risk

Figure 1 - HIS Total Software Approach

Each track encompasses a major area for selected research, and must be refined into subtracks representing different aspects, approaches, features, or options within their respective research areas. Some, but not all, of that breakdown has occurred and is reflected in the roadmap track descriptions in this section.

Three projects within the first track are discussed in some detail. A brief review of the remainder of the projects in the first track and the projects within the second track, that are currently under development, follows the detailed discussion. The three projects discussed in greater detail include:

1. Correct Specification via Visualization, Synthesis, & Analysis;
2. Correct Implementation of Components; and
3. Visualization of Abstract Objects.

Furthermore, as work progresses, we expect further refinement to occur, new subtracks to be identified, and some subtracks to be abandoned. We also expect that new tracks will be identified in the future, and old tracks may split. Each track will have a champion, who will keep the vision for that track alive, and who will steer the "coming and going" of subtracks to assure progress toward the vision. For now, we will review the track on Correctness by Design before proceeding with the System Immunology™ track.

Correctness By Design

Five projects are currently funded under this track. They include projects related to: Correct Specification via Visualization, Synthesis, & Analysis, Correct Implementation of Components, Visualization of Abstract Objects, and Software Testing for High Consequence Automated Systems. The last project, the Software Testing for High Consequence Automated Systems project, investigates the use of production supervisory control software to test large complex dynamic electromechanical systems and is not discussed here. Efforts underway on the first three projects are described in the following subsections.

Correct Specification via Visualization, Synthesis, & Analysis: This track of Sandia National Laboratories High Integrity Software (HIS) Initiative is to develop a software/hardware system specification, design, and implementation methodology along with tools that will guide the developer to intrinsically "sure" designs. Initial efforts by Yakhnis and Yakhnis have identified the following goals for methodologies and tools to be developed within this track:

1. The system specification will accurately reflect the true customer intent; also, the methodology will help the customer to reveal and evaluate all of the information included in his or her original idea;

2. All system analysis and design documents will have a precise semantics (*e.g.,* as in the Object-Oriented System Analysis(OSA) model (Ref. 8) or the Business Object Notation (BON) model (Ref. 15)). The semantics will serve as a basis for prototyping and visualization at every stage of system creation, from requirements capture to design;

3. The system will conform to the specification and design via computer generated mathematical proofs, so that each layer of the design or code will conform to the element of the design or specification positioned immediately above within the specification/design hierarchy (Refs. 18, 19);
4. At each analysis of a design step, the customer will be provided with persuasive demonstrations (*e.g.,* via computer visualization) that the system behaves as desired;

5. The system will maintain traceability of requirements in the sense of an automated ability to locate the respective customer requirements for every element of the design and/or code;

6. Maintainability will be sound in the sense that specification, design, and code will be continuously maintained to be mutually consistent; and

7. System surety (in respect to safety, security, etc.) will be enhanced by guaranteeing predetermined system behaviors with respect to a list of unusual

circumstances provided by the requirements (*e.g.,* hardware malfunctions). Specifically, the system will be able to either undertake a protective action or gracefully degrade its performance while giving sufficient warnings to users (Ref.17).

Yakhnis and Yakhnis have proposed that these goals can be achieved via seamless (Ref. 15) integration of OOA, OOD, code generation, visualization, and automated correctness proofs. Specifically, they suggest pursuing the following steps: 1) making specifications transparent and easily accessible; 2) insuring that the specification captures the original idea; and 3) enabling the specification to govern the design and implementation.

The first step identifies how to make the specifications transparent and easily accessible. To begin, they distinguish three ways to represent a specification:

1. An informal specification in a natural language. This approach stems directly from the requirements, and thus, is at least partially understood by the customer. However, without conversion into the two other forms, this method is usually not conducive to systematic design and implementation;

2. A formal specification. This approach is usually not understood by the customer; although, this technique may be conducive to automated development of correct systems; and

3. An object-oriented analysis model. This tactic allows the specification to closely model the real world, possibly serving as a common ground for communication between the customer and the developers.

Note that a non-hierarchical moderately complex specification of any of the above is usually not understood in its entirety by either the customer or the developers. Thus, the following six actions are suggested for improved specifications:

1. develop hierarchical specifications limiting each observable element to no more than seven subordinate entities;

2. represent the specification as three documents consisting of: an informal specification, a formal specification, and an object-oriented analysis model;

3. for the object-oriented analysis model, choose an object model (e.g., OSA or BON) which does not include any elements of design (Ref. 8, 15, 19). Doing so will prevent developers from distorting the requirements analysis stage by making design decisions too early;

4. coordinate the hierarchical structure of the three documents (For example, each object within the object-oriented model should correspond to its description within the informal specification document);

5. provide hypertext-like links between corresponding elements among the three specification documents; and finally,

6. do not use the formal specification document to communicate with the customer as they may not have the necessary skills to fully understand the formal notation.

The second step in the Yakhnis methodology focuses on insuring that the specification captures the original ideas expressed by the requirements. A "simultaneous iterative refinement" procedure should be used to capture the specification from the original customer requirements (SIRC). The customer should have control over the capture/extraction process at all times since the feedback from the developers will be provided in several transparent forms, including visualization.

Finally, the third step is to enable the specification to govern the design and implementation. The "simultaneous iterative refinement" procedure should be extended via the object-oriented stepwise refinement process to obtain a "simultaneous iterative refinement" procedure of design (SIRD). Under this SIRD procedure, each object is treated as a new system to be analyzed and specified. Thus, the design is viewed as a continuing application of methods for analyzing requirements, albeit with smaller granularity of objects.

The hardware and software should be developed jointly, with their separation only occurring for appropriate granularity of objects when needed. Further, at each step of the design process, a mathematical proof verifying that the internal design of each object (i.e., subsystem) conforms to its external specification, will be computer-generated (Ref. 19). Finally, a target code that has been proven correct mathematically will be generated automatically (Ref. 16).

Correct Implementation of Components: This track focuses on achieving advancements in program transformations to ensure correct implementation of components. Several goals have been identified for this research area and initial investigations are focusing on the "task scheduling" problem using algorithms developed in the WALS and APP pit handling projects at Pantex.

Program transformation can be a means to formally and correctly bridge the gap that exists between the specification of a problem in some domain specific language, and a realization of the specification in some programming language. Exactly what constitutes a domain specific language and what constitutes a programming language is more or less irrelevant from a theoretical point of view.

Given a specification, *s*, that is expressed in some domain specific language, a transformation sequence *T* can be constructed that will transform *s* into *s'*, where *s'* is an executable program belonging to some previously selected target language. Furthermore, if *T* has been shown (through formal proofs) to be "correctness preserving," then one can conclude that the program *s'* is correct with respect to the specification *s*.

Ideally, a correct formal specification, *S*, of the problem would be produced by research efforts under the correct specification via visualization, synthesis and analysis track described in the previous section. This formal specification would be in a domain specific language whose formal semantics would also be defined.

It should be noted that the specification, *S*, might be in a language that is not directly executable by a computer, or *S* might be inefficiently executable. At this stage, transformations can be applied to *S* with the goal of producing a program, satisfying *S*, that can be efficiently executed by a computer.

In order to accomplish this, one needs to: 1) define the source and target language in a common semantic framework; 2) write a transformation sequence, *T*, that is capable of transforming *S* into a program *P*; and 3) prove the correctness of the transformation sequence, *T*. *TAMPR*, a transformation system created by James Boyle at Argonne National Laboratory, provides suitable functionality needed for this effort. *TAMPR* views specifications, programs, and transformations in terms of syntax derivation trees (SDT's). In this paradigm, a transformation consists of a rewrite rule stating that one SDT should be rewritten into another.

Typically, syntax derivation trees associated with transformations tend to be quite massive; consequently, researchers at Sandia continue to investigate environments and tools that will facilitate manipulating, constructing, and reasoning about transformations. Because Pad++ allows syntax derivation trees to be presented in a form more amenable to human understanding, it is under review for this program.

Once a series of transformations have been developed and applied, the correctness of that transformation sequence can be proven with the assistance of an automated reasoning system. Winter has developed an approach that uses the automated reasoning system OTTER; however, an extension to this system is necessary in order to make the transformation proofs more manageable.

With current technology, transformation proofs require several passes, each providing a portion of the overall proof. Search strategies and inference rules may vary from one pass to the next. Ideally, the overall strategy of a complex proof would be defined within the automated reasoning system itself, eliminating the need for these separate passes.

Finally, a significant amount of research needs to be done in order to expand the class of transformations about which current methodologies are capable of reasoning. These areas of research are mostly near term (3-5 years) goals. A list of the general areas with a brief description of what is needed is given below:

1. Automatic deduction of delta-functions. Delta functions are essentially the semantic manifestation of syntactic variables that can occur within transformation schemas. It is because of these variables that transformations obtain a general applicability. Reasoning about such variables requires knowledge of their semantics. A delta-function captures the semantics of such variables; currently, these delta-functions are constructed by the user, a situation that is unacceptable if one desires to produce high integrity software.

2. Reasoning about subtransformations. A subtransformation is a transformation within the body of another transformation. Research needs to be conducted on how such information can be adequately expressed and exploited in a correctness proof.

3. Formally deducing and incorporating preconditions (canonical form properties). Research in this area centers around the development of a theory enabling one to reason about properties other than correctness that are established by transformations and transformation sequences.

There are many areas that need to be researched and developed further as long term goals in order to produce a usable production strength methodology. With the present technology, it is quite difficult to prove the correctness of transformations that introduce significant algorithmic implementation decisions. Dramatic improvements can (and need to) be made in this area.

Currently, Winter suggests further investigation of refinement calculus. In addition, he believes efforts to make algorithmic implementation decisions (to some extent) automatically deducible by computer through observation of human solutions to "example" problem instances would be likely to improve current technology.

Development of a methodology that is capable of quantitatively computing the reliability of arbitrary analysis techniques (e.g., risk analysis, formal verification, etc.) will follow earlier efforts. This idea is based on measuring the resiliency of an analysis technique to typos and other errors. Essentially, we will measure the chaotic nature of the analysis technique, relating this work to the predictive measurement track discussed in Ref. 13.
There are several critical issues or show stoppers for this track. In order for this technology to succeed, a specification language and a specification must be produced in the parallel track on correct specification via visualization, synthesis, & analysis that is amenable to the transformation process. This requires a frequent exchange of ideas between these two tracks.

Further, it is extremely desirable, at some future point, to be able to extend reasoning about transformations to properties other than correctness (*e.g.,* safety); of course, those properties need to be defined first. Currently, some preliminary theoretical research has been done with respect to reasoning about general properties that are established by *transformations* and transformation sequences. We envision that properties other than correctness (*e.g.,* safety) can be handled within this theoretical framework.

Visualization of Abstract Objects: The previous two tracks deal with earlier phases of the software life cycle and progress to code development. However, once a software system has been developed, the problem still remains of assessing software surety status--rigorous processes and methods applied to early phases of the software life cycle alone cannot assure software integrity, safety, security, and reliability in the final end product. The implementation itself must be verified, with particular focus on surety aspects for high-consequence systems. In that regard, several key issues include whether or not the executing software properly incorporates specified constraints, and whether or not all necessary constraints and their interactions have been considered, understood, and correctly implemented to avoid loss of life or other undesirable effects. How do we verify the surety attributes of a system implementation?

Traditionally, there have been three areas of research for verification of system implementations: logical verification, mathematical verification, and statistical verification. However, Berztiss (Ref. 3) has advocated that every possible technique and method should be utilized to address safety concerns, as current methods to address this problem are inadequate. While testing the actual system code does provide substantial information regarding the correctness of the system, generally this is an incomplete method for assessing surety aspects as economic and scheduling restraints prohibit the level of testing required to achieve the necessary confidence in the surety of real-world systems. Further, tested programs may correctly execute their specifications, but with current textual and limited graphical documentation, it is difficult to ascertain whether a code does what is needed.

Mathematical models can be considered for this task. However although rigorous, they can only prove that the implementation meets the specific requirements. They do not allow support for identifying any cases that have not been considered within the requirements and specifications--a drawback of mathematical techniques, they only work if the right cases are proven. Reliability models are also useful, but again, they can only provide statistical confidence at levels that are clearly beneath those required for these high-consequence systems, and they, generally, are making predictions about future failures of the systems without addressing the types of errors or their significance. Finally, none of these existing methods of research address the difficulty of assessing whether all necessary constraints have been specified. This is an area visualization can address.

Therefore, it is time to consider a fourth category, the use of visualization, in addressing the issue of verification. Accordingly, several such efforts are underway in various laboratories and universities (Ref. 2, 14, 10), including an investigation of software attributes visualization within the High Integrity Software program at Sandia National Laboratories.

Visualization techniques have been used quite successfully within the scientific community for some time; and not surprisingly, many researchers feel the utility of visualization as a means of illustrating the properties of multiple objects, or as a means of demonstrating properties of supersets of discrete items, may be considered a given (Ref. 4). Fortunately, this benefit of improved comprehension through visualization can be achieved in other application areas as long as the appropriate visual model is selected. Correspondingly, although system verification is a new context, visualization provides the capability of increased system comprehension, thereby facilitating discoveries that are not otherwise possible. This is a major benefit of using visualization in a formal method to investigate surety aspects of a system implementation. However, little work currently has been undertaken to apply multi-dimensional visualization techniques to software analysis (Ref. 5), while a number of projects have focused on algorithm animation, at least in two dimensional formats (Ref. 20). (It is only fairly recently that hardware support has been sufficient to allow work on information visualization for analysis of software.)

Projects are just beginning to investigate the use of this methodology for enhancing understanding of system software. Initial successes have resulted in recommendations of investigating the use of virtual reality technology to map multiple-layer software systems onto expansive 3-dimensional terrains and providing more direct means for traversal as a more effective facility for software visualization (Ref. 10). We are investigating such a use of visualization and virtual reality techniques, with our efforts going further in utilizing these technologies in assessing surety factors for high-consequence software through the verification of system software (Ref. 13), as current visualization models do not evaluate or portray surety issues. A multi-dimensional abstract model is used to reduce system complexities associated with the conceptual mapping of a problem domain into a software solution space.

The goal of this track is to improve cognition of software systems behavior and improve software surety confidence by providing an environment that allows visualization of abstract objects and animation of program behavior incorporating requirement constraints. The project focuses on a multi-dimensional visualization of software abstractions that incorporates a technique for assessing the correct implementation of select requirement constraints during the execution phase of the life-cycle process.

The prototype software attribute visualization tool is developed on Eigen/VR, a multi-dimensional user-oriented synthetic environment developed at Sandia National Laboratories. The tool incorporates the use of requirement constraints, expressed in a requirements constraint language, in the visualization of an executing program. As the program executes, selected requirement constraints are monitored and if violated, the abstract visual model indicates those errors have occurred.

Systems Immunology™ Track

Projects in this track use current technology to provide lower-risk, medium payoff research addressing surety issues for high integrity software. The projects and their goals will be briefly addressed. Currently, there are three projects underway: Digital Isolation and Incompatibility; Path Expressions; and System Fault Analysis.

The Path Expressions project extends capabilities of existing path expression methodologies to include event sequence and timing concerns in high consequence software. The Digital Isolation and Incompatibility project uses software path expressions and hardware state machine monitoring to provide highly reliable software solutions. And, the System Fault Analysis project extends a top-down fault analysis methodology, based on Fault Tree Analysis to identify high consequence hardware failures in software-controlled systems using microprocessors.

Conclusion

We have presented the beginnings of a strategic plan for high integrity software. We have identified necessary attributes, strategies and supporting technologies and capabilities needed to achieve our goal of improved software surety. Further, we have specified and described several research tracks to accomplish our stated goals. Improvements will need to be achieved within several different tracks to attain success.

Our next step is to continue development of the research tracks, expanding into a more complete strategic plan containing specific goals, strategies objectives and tasks. In addition, we need to continue assessing the research tracks to determine project priorities and their expected impact and contributions towards HIS goals.

References and Notes

1. Albuquerque Journal, Sunday, November 12, 1995.

2. Ball, T., S.G. Eick, "Software Visualization in the Large," *Computer*, April 1996, pp. 33-43.

3. Berztiss, A.T., "Safety-Critical Software: a research agenda," *International Journal of Software Engineering and Knowledge Engineering*, Vol. 4 No. 2, 1994, pp. 165-181.

4. Braham, R., "Math & Visualization: new tools, new frontiers," *IEEE Spectrum*, November 1995, pp. 19-37.

5. Clifford, C., Huff, M. Klein, S. Stevens, "The State of the Art in Scientific Visualization," Technical Report, CMU/SEI-95-SR-Visualization, Software Engineering Institute Carnegie Mellon University, September 1995.

6. Collins, E., L. Dalton, D. Peercy, G. Pollock, and C. Sicking, "A Review of Research and Methods for Producing High-Consequence Software," *1995 IEEE Aerospace Applications Conference*, Vol. 1, January 1995, pp. 197-245.

7. "Cyberware," Time, August 21, 1995.

8. Embley, D., Kurtz, B., Woodfield, S., *Object-Oriented Systems Analysis (A Model-Driven Approach)*, Yourdon Press, 1992.

9. Gibbs, W., "Software's Chronic Crisis", *Scientific American*, September 1994.

10. Kimelman, D., B. Rosenburg, T. Roth, "Strata-Various: Multi-Layer Visualization of Dynamics in Software System Behavior," IBM Thomas J. Watson Research Center, June 1994.

11. Lagedec, P., "Major Technological Risk", Quoted in *Safeware, System Safety and Computers*, Nancy Leveson, University of Washington, Addison-Wesley, 1995.

12. Musa, J.D., A. Iannino, K. Okumoto, *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, Inc., 1987

13. Pollock, G.M., L.J. Dalton, "A Strategic Surety Roadmap for High Consequence Software," *1996 Aerospace Applications Conference*, Snowmass CO, Vol. 4, February 1996, pp. 351-370

14. Steven P. Reiss, "An Engine for the 3D Visualization of Program Information," Dept. of Computer Science, Brown University, May 1995.

15. Walden, K., J. Nerson, *Seamless Object-Oriented Software Architecture*, Prentice Hall, 1995.

16. Winter, V., A. Yakhnis, V. Yakhnis, High Integrity Software: Automated Software Design via Refinement Transformations, submitted to 8th Annual STC'96.

17. Yakhnis, A., V. Yakhnis, High Integrity Software: Capture and Analysis of Requirements on Safety via Strategic Multiagent Approach, submitted to the 8th Annual STC'96.

18. Yakhnis, V., J. Farrell, S. Shultz, "Deriving Programs Using Generic Algorithms," *IBM Systems Journal*, Vol. 33, no. 1, pp. 158-181, 1994.

19. Yakhnis, V., A. Yakhnis, "A Model of Object-Oriented Analysis and Design Tailored Toward Stepwise Refinement," to appear as a technical report, Mathematical Sciences Institute, Cornell University.

20. Zeus, DEC Systems Research Center.

Biography

Guylaine M. Pollock, Ph.D.
Sandia National Laboratories
Computer Sciences Dept. 9224
MS 1109
PO Box 5800
Albuquerque, NM 87185-1109

Guylaine M. Pollock, a Senior Member of the Technical Staff at Sandia National Laboratories, received a Ph.D. in Computer Science from Texas A & M University and a BS in Computer Science and Mathematics from East Texas State University, graduating with Academic Distinction and Highest Honors. She has served on Software Capability Evaluation Teams for the Battle Management Defense Organization of the Department of Defense. She has investigated software reliability for massively parallel codes and is a member of the Sandia Reliability Working Group. Dr. Pollock is a member of the Board of Governors of the IEEE Computer Society currently serving as Treasurer, and has previously lectured with the Distinguished Visitors Program for the society. She has received several awards including the Richard E. Merwin Scholarship and Notable Women of Texas.