SAND95-3065C
CONF-960248--1

A Strategic Surety Roadmap for High Consequence Software

Guylaine M. Pollock
Sandia National Laboratories, MS 1109
Computer Science Department
P.O. Box 5800
Albuquerque, NM 87185-1109
505-845-7463
gmpollo@cs.sandia.gov

Larry J. Dalton
Sandia National Laboratories, MS 0535
Command and Control Software Department
P.O. Box 5800
Albuquerque, NM 87185-0535
505-844-2520
ljdalto@sandia.gov

Abstract—A strategic surety roadmap for high consequence software systems developed under the High Integrity Software (HIS) Program at Sandia National Laboratories is presented. Selected research tracks are identified and described detailing current technology and outlining advancements to be pursued over the coming decade to reach HIS goals.

TABLE OF CONTENTS

1. 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. This paper describes a strategic surety roadmap 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 initiative has focussed initial efforts on the development of a roadmap to form the foundation for strategic decision making. The roadmap also provides a framework for characterization of the High Integrity Software (HIS) problem in Sandia's context and identifies key technologies of importance to the development of high integrity software through the next ten to fifteen years.

We will summarize and present the most pertinent aspects of the roadmap and identify and discuss the most relevant technology tracks. Specifically we will describe each of the selected technology tracks and identify why it is important, what is the current state of that technology and what advancements should be pursued over the current(1-3 years), near term (3-5 years), and long term (5-15 years) timeframe to enhance factors affecting software integrity.

Various tracks discussed herein include tracks on: Predictive Measurement; Correct Specification via Visualization, Synthesis, & Analysis; Correct Implementation of Components; and System Composability. The Predictive Measurement track focuses on identifying key areas for research and proposing various projects that complement each other in advancing expertise in this area. In parallel, research efforts in the Correct Specification via Visualization, Synthesis & Analysis track focus on the exploration of new and innovative methods for the capture, synthesis, simulation, evaluation and verification of the behavior of modeled software systems; while the Correct Implementation of Components track examines ways to improve capabilities to automatically derive programs from specifications through the use of program transformations that have been formally verified to preserve correctness. Finally, the System Composability track addresses the ability to build systems out of components with known properties with the ability to state something about the surety properties of the resultant system. (Other tracks not discussed in this paper include Visualization of Abstract Objects, and Architecting & Design using Surety Principles & Risk Assessment. These tracks will be discussed at a later time as they become more fully developed.)

Next, we review the motivation behind this new program initiative, present the current status of the High Integrity Software Web developed by the HIS Software Roadmap Team, and explain how the roadmap tracks map back to the web. Afterwards, we proceed to a more detailed description of the selected roadmap tracks before concluding.

2. 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. Further, this program strives to develop better technologies within the software industry that will enable 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, The power trains in new General Motors cars run 30,000 lines of computer code." [5]—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 the 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 figure out 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. [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" [6].

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" [3]. 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. Accordingly, one of the first actions taken under this program was the establishment of a steering committee to guide research activities.

3. HIGH INTEGRITY SOFTWARE WEB

The formation of a steering committee for this initiative was formed with members from various groups including Sandia, Allied Signal Inc., Microelectronics and Technology Center in Columbia, MD, the Nuclear Regulatory Commission, and the Software Engineering Institute (SEI) at Carnegie Mellon University, The committee was established in March of 1995.

Figure 1 depicts the high integrity software web that has been established to date by this group. The HIS Web is used to identify the attributes, strategies, necessary technologies and capabilities needed to achieve high integrity software. Consequently, it forms the initial framework for a strategic plan being developed to broaden surety focus and understanding through secure design, surety management, ultra-reliable engineering, and predictive stewardship.

The strategies identified by the web include: Engineering for Change, Composing Systems, Satisfying Surety Requirements, and Analyzing and Predicting. Other strategies may be identified later. We will briefly discuss each of

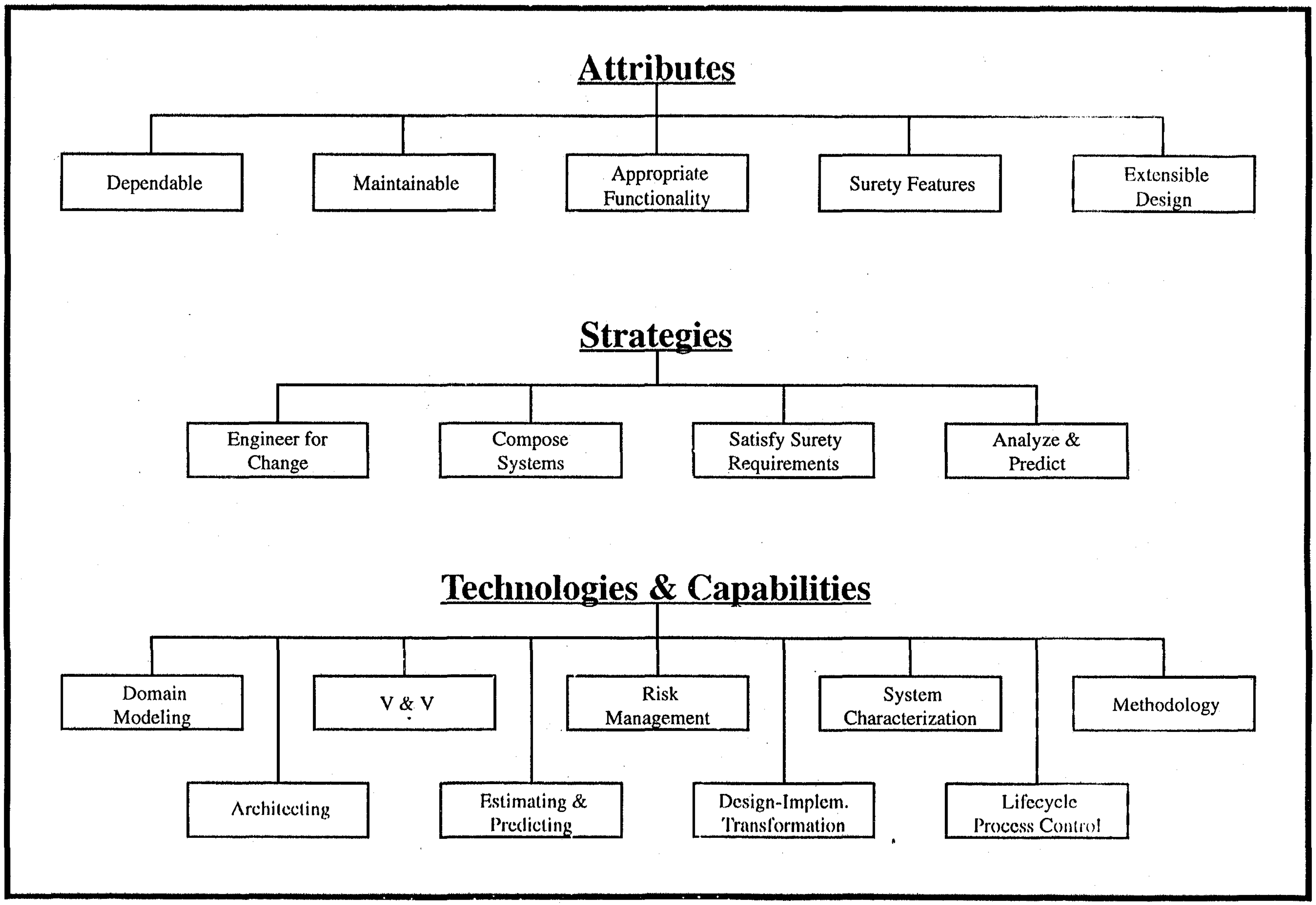


Figure 1. High Integrity Software Web

the specified strategies before continuing with the roadmap tracks; however, the technologies and capabilities identified in figure 1 should be clear to the reader and thus will not be defined further within this paper.

The first strategy, Engineering for Change, is important because many changes, especially changes upon changes, can produce fragile systems. And unfortunately, changes are a fact of life for most software systems—both from the aspect of changing requirements and from the aspect of correcting identified errors in the code. While current lifecycle development paradigms do not deal well with changes, newer paradigms such as model-based engineering may have advantages for managing change. Accordingly, research efforts should incorporate this strategy whenever possible.

The second strategy, Composing Systems, focuses on building systems out of components with known properties, so that one may be able to characterize something about the properties of the resultant system. In fact, there are many issues involved with this approach. For example, discovering properties to measure and compose, achieving the measurement, modeling how the composition works, and approaching domain modeling and system architecture with reuse as a goal. Obviously, incorporating COTS (commercial off the shelf) components will be a challenge.

The third strategy identified by the HIS Web is Satisfying Surety Requirements. This clearly is an important strategy to address in accomplishing the goals of HIS. But first we must clarify what exactly is meant by the phrase *surety requirements*. Sandia defines surety to mean a system that is safe, secure, reliable, and under control. Thus this phrase refers to any requirements that affect system attributes relating to surety or integrity issues. Clearly, system surety requirements need to be explicitly specified and satisfied. In doing so, there are issues of trade-offs, residual risk, and the need to understand the effects of system changes on surety that must be examined. Further, there is also an aspect of ensuring that the system has no unwanted effects to consider. In implementing this strategy, there may be certain architectures, or techniques that can be engineered into systems that have a positive impact on surety.

The fourth and last strategy currently included in the web, Analyzing and Predicting, is needed to provide the capability to model, analyze, and predict software behaviors in order to produce reliable systems that implement intended functionality, have known properties, and remain under control. Selected behaviors must be analyzed more rigorously in order to understand characteristic responses for properties of interest. If a "watchdog" is to be able to recognize unacceptable system behavior, for example, one must define what behavioral properties to watch. All four of these strategies will be considered by research efforts reviewed in the next section.

4. Roadmap Tracks

After establishing the general framework of the web 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. The roadmap tracks currently identified are:

- Predictive Measurement;
- Correct Specification via Visualization, Synthesis, & Analysis;
- Visualization of Abstract Objects;
- Architecting & Design using Surety Principles & Risk Assessment;
- Correct Implementation of Components; and
- System Composability.

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 2.)

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.

In order to save space, we will only present the web mapping in full for the predictive measurement track to illustrate what was done. In actuality however, all of the tracks map back to the web. This allows assessment of our organization's coverage and strengths associated with the necessary technologies and capabilities that must be achieved to improve software integrity. This method pinpoints areas where additional efforts should be focused. In addition, these mappings still need to be rated to establish each task's level of impact on those technologies and capabilities specified in the web. Additional work needs to be done to map specific tracks back to the strategies and attributes. Each strategy may be accomplished through a variety of approaches and the mappings are needed to assess the coverage achieved by on-going projects.

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.

A complete track definition consists of the following information—a track description, a list of specific tasks and milestones for current (1-3 years), near term (3-5 years), and long term(5-15 years) timelines; the, expected benefits of pursuing the track, critical needs for pursuing the track, and critical issues affecting the success of the work.

Each of the tracks are in various stages of development; therefore, we will only present the information that is available for the tracks that are discussed. We will discuss the following tracks briefly: Predictive Measurement; Correct Specification via Visualization, Synthesis, & Analysis; Correct Implementation of Components; and System Composability. The two newest tracks—Visualization of Abstract Objects, and Architecting & Design using Surety Principles & Risk Assessment—will be presented at a later.

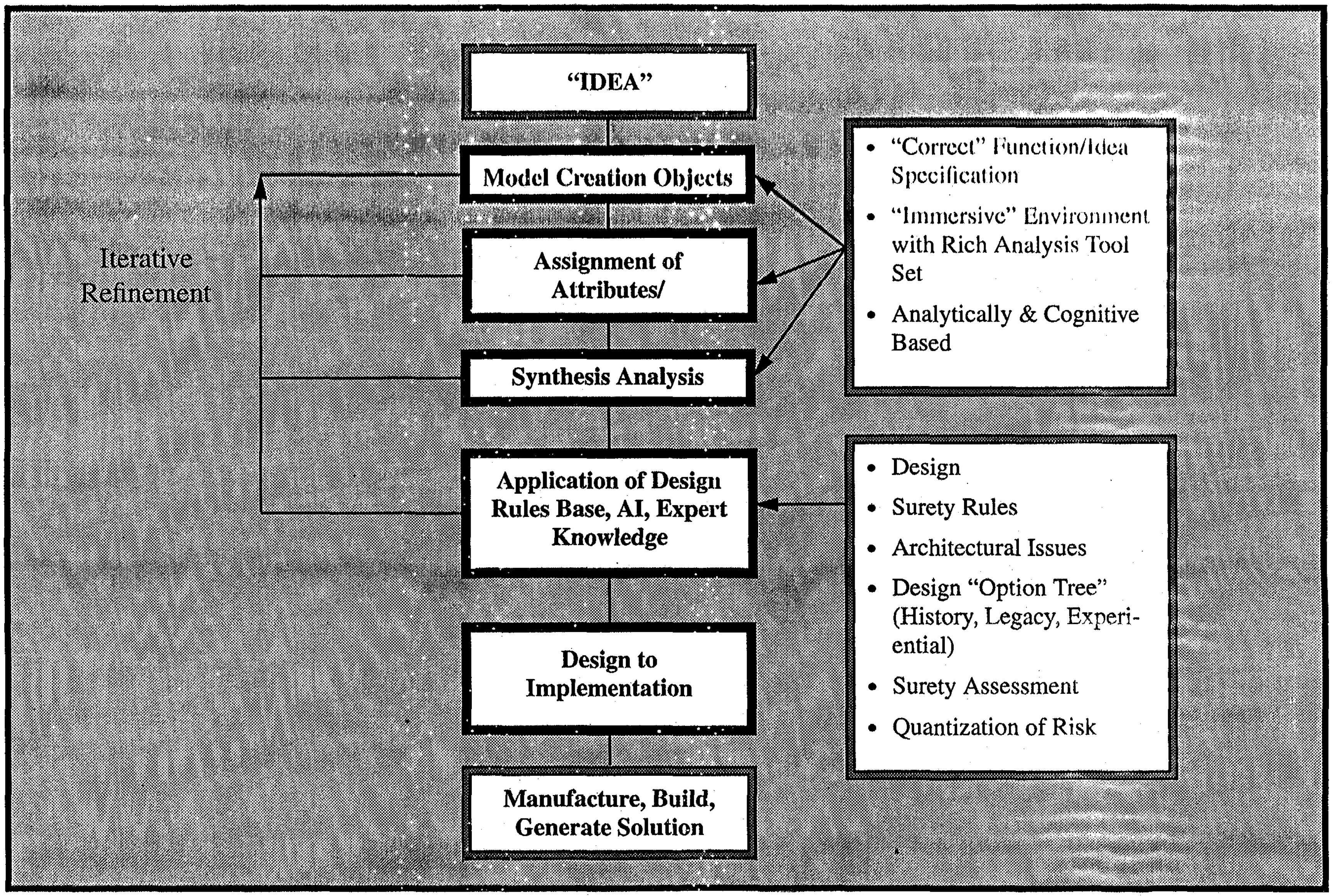

"IDEA"
Model Creation Objects
Assignment of Attributes/
Synthesis Analysis
Application of Design Rules Base, AI, Expert Knowledge
Design to Implementation
Manufacture, Build, Generate Solution
Iterative Refinement
• "Correct" Function/Idea Specification
• "Immersive" Environment with Rich Analysis Tool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

Figure 2. HIS Total Software Approach

Predictive Measurement

If software is to be used in high integrity applications, predictive measures that quantify the risk associated with using software for specific functions are highly desirable. A key aspect of work in this area is determining which of the many different aspects and characteristics of software should be measured. For example, a measure that predicts, estimates, and quantifies the rate of failure occurrences in software could be employed to begin to increase one's understanding of the associated functional risks. This particular type of predictive measurement is referred to as software reliability.

Increased understanding of reliability aspects is a first step towards improving predictive measurements for high integrity software; however, other software properties clearly contribute to the overall integrity of the software. Thus, research in predictive measurement must also consider and examine additional attributes besides reliability. Some of the additional software properties that contribute to high integrity include performance, safety, and security attributes. Ultimately, formal models will need to be developed to assist in predicting these software attributes. As the reliability of software contributes to the safety and security "rating" of any particular system, and consequently must be part of any models predicting safety and security levels, we will focus our subsequent discussion on the advancement of reliability issues as the example framework for initiating activities/research in this track.

Software reliability can be defined as the probability of failure-free operation of a computer program for a specified time in a specified environment [7]. To estimate or predict software reliability, a model that accurately characterizes the behavior of a software program over time is required. This model is usually derived from data collected about defects detected during development testing. As research efforts grow in this discipline, the models should progress to providing information earlier in the development process, hopefully during the specification phase. Further, there are many additional aspects such a measure/model must consider, including the assessment of criticality for the types of failures that may occur.

Currently, software reliability models are essentially reliability growth models. These models are based on a test, find, and fix approach. A computer program is tested until it fails. The failure is investigated and the fault is corrected. The program then undergoes further testing to uncover the next failure. This process repeats until sufficient data is available to predict the reliability of the final "corrected" program based on the observed failure data from the previous versions.

Various models utilize different approaches in addressing the problem of reliability prediction. However, most assume that all software faults are independent of others, and many are not advanced enough to differentiate between the severity of failure classes. Work in this area must increase our understanding of the types of failures within ~~the~~ various stages of the software lifecycle process and their respective consequential impacts. Furthermore, many other questions need to be addressed. Typical kinds of questions that should be explored include: "How do functional changes affect the mode?," "Is a software failure distinguished from a *requirements* failure?," "How are non-function requirements considered?," and "Are all failures equivalent?."

Despite the drawbacks of current models, software reliability techniques are used by many commercial firms with reputations for producing high reliability systems. A few examples include AT&T for the 5ESS Telephone Switch System; Loral Federal Systems for the space shuttle mission software, air traffic control systems, and data management systems for the IRS and FBI; and Motorola cellular telephones. These companies deliver software with a predicted reliability to their users. [2] provides an overview of work being done in this area.

Application of the proposed software reliability methodology to SNL Software development projects can provide a baseline assessment of the reliability of SNL software products. Further, it can provide vital feedback on the success of process improvement efforts. Without a baseline reliability assessment measure, the cost versus benefit trade-off of additional process improvement or testing to produce high reliability software cannot be evaluated. Process improvements derived from analysis of the software defects database may yield a higher quality software that is less likely to fail in an unsafe mode. Accurate estimates of reliability growth during test of software provide additional assurance that the software under operational conditions will meet or exceed reliability expectations.

Track Timeline—The track for predictive measurement focussing initially on a software reliability methodology is provided in the six main tasks described below. Three of the tasks, Characterization of Software Integrity, Software Defects Database and Root Cause Analysis, are identified as current tasks, to be accomplished in the next three years. The first task is a planning task that essentially expands the predictive measurement track into measurement and prediction of selected characteristics that contribute to overall software integrity. Two tasks, Quantification of Reliability and a Software Environment for Defects Data Collection, are near term tasks. The final two tasks, Correlation of Software Reliability to Non-test Related Metrics, and Correlation of Software Reliability to Safety/Security Metrics are long term research-oriented tasks. These tasks establish a basic framework for research in this area and identify the types of projects that should be considered. Additional projects may be considered as needed.

Current (1-3 years). Three tasks are proposed for current efforts:

Task 0: Characterization of Software Integrity

This task is a planning effort focussed on identifying the various software characteristics and aspects that contribute to software integrity. The goal is to identify characteristics that would benefit from a predictive measurement model.

- Subtask 1: Identify software issues, aspects, and characteristics that contribute to software integrity.
- Subtask 2: For each issue/characteristic/aspect, identify the types of information that must be gathered to measure or rank the characteristic.
- Subtask 3: Rank the characteristics according to their impact on the integrity of software.
- Subtask 4: For each characteristic/aspect of import, develop a schedule of tasks similar to the following tasks (tasks 1-6) outlined in detail for reliability with the goal of producing a similar predictive model for each desired characteristic measurement model.
- Subtask 5: Initiate activities for as many of the task schedules outlined in subtask 4 as time and resources permit.

Task 1: Software Defects Database

Information about defects detected during software development are to be collected into a database for analysis. The Software Defects Database is the most important task in the Software Reliability methodology. Task 2 through 6 cannot be started without the defects data collected during this task. This task is further described by the five Subtask listed below.

- Subtask 1: Identify the goals of the data collection program, i.e. quantification of the risk of software in a specific usage.
- Subtask 2: Identify critical core metrics and the necessary data to be collected, such as, mean time-between-failures, defect density, customer reported problems, defect removal effectiveness, etc.
- Subtask 3: Identify/enlist High Integrity Software development programs for data collection.
- Subtask 4: Identify/develop a software database and supporting analysis tools.
- Subtask 5: Populate software defects database with defects detected during high level design inspections, low level design inspections, code inspections, unit tests, integration tests, systems tests, and field tests from ongoing software development programs. Also, collect available data from any operational usage.

Task 2: Root Cause Analysis

A Root Cause Analysis examines software defects detected during software development to determine what are the underlying causes for the defects.

- Subtask 1: characterize software development processes for the different SNL software development programs from which defects data are collected.
- Subtask 2: Examine defects to determine if any general trends for defect injection into software products are indicated by the defects data.
- Subtask 3: Identify/evaluate/implement corrective actions to improve SNL software development processes.

Near term (3-5 years). An additional two tasks have been identified for the near term timeline:

Task 3: Quantification of Reliability for SNL Software Products

A software reliability model is selected and validated to assess/predict the reliability of SNL software products.

- Subtask 1: Select/develop software reliability models(s) from statistical analysis of the defects data in the database.
- Subtask 2: Predict reliability for a SNL software product that is being fielded.
- Subtask 3: Collect field failure data for the software product. Compare reliability prediction with field experience to validate reliability prediction model(s).
- Subtask 4: Evaluate cost/benefit of application of reliability prediction model(s).
- Subtask 5: Improve/refine reliability prediction models and identify lessons learned.

Task 4: Software Environment to Facilitate Defect Data Collection

Based on lessons learned during the prior three tasks, software tools to assist data collection, data storage, and data analysis are developed/procured.

- Subtask 1: Identify impediments to defects data collection based on lessons learned.
- Subtask 2: Explore techniques to automate/simplify data collection process.

- Subtask 3: Develop/procure software tools to assist in defects data collection, data storage, configuration management, and data presentation.

Long term (5-15 years). Two task were identified as long term activities:

<u>Task 5: Correlation of Software Reliability to Non-test Related Metrics</u>

Software reliability models are based on test data from the software product. Software is expensive to test. A more cost effective approach to reliability prediction could entail predictive measures based on metrics that are not test related. This task is to identify and explore other factors that may influence software reliability.

- Subtask 1: Analyze defects data to identify factors that may influence final product reliability.
- Subtask 2: Develop reliability prediction model(s) that are not driven by test data.
- Subtask 3: Validate reliability prediction model(s).

<u>Task 6: Correlation of Software Reliability to Safety/Security Metrics</u>

Software reliability models should be incorporated into newer more expansive models that examine and assess safety and security levels of software in order to increase our abilities to produce high integrity software with confidence.

- Subtask 1: Examine defects data and software reliability predictions for specific systems to identify factors that may influence the safety/security attributes of a system. Determine if higher reliability correlates with enhanced safety/security.
- Subtask 2: Define metrics (qualitative and/or quantitative) for the safety and security attributes for the types of systems that Sandia National Laboratories develops.
- Subtask 3: Define desirable or acceptable levels of risk for systems in terms of the metrics.
- Subtask 4: Correlate the defects data to quantifiable safety and/or security metrics. Analyze the data to determine what processes may enhance the safety/security attributes of the system.

Benefits—Investment in this technology will provide a number of benefits. Only a few of the most important are briefly outlined here. These benefits are both internal and external and provide advancements both in technical and procedural aspects of the software lifecycle.

From a technical aspect, advancements in this area will allow users of the technology to deliver a product with an expected reliability. In addition, it will allow the evaluation of non-test related characteristics of software to predict reliability. This will allow an earlier assessment of the reliability within the software lifecyle, thereby improving potential cost savings. Current models expect a 5% reduction in the cost of the overall project if these techniques are applied. We expect to see an improvement in that figure if reliability assessments can be made earlier in the lifecycle process.

Further, the improved ability to assess domain specific attributes through objective data collection and analysis will allow more software decisions to be made based on domain characteristics and ultimately provide the basis for leading edge research in reliability assessment of critical software. With a strong historical database, we will develop better metrics and models for our domain that are more meaningful, accurate, and easier to use; and the database can also be used for more objective benchmarking of alternative methodologies.

The tasks as outlined above in this predictive measurement track will also provide advancements for the overall software process. In particular, the database will provide for better management of test assets and other resources such as scheduling and cost issues. It will provide an objective opportunity for continued process improvement; and, it will allow objective analysis of lessons learned.

In the area of process improvement, successful completion of these tasks would provide innovative and successful tools for the collection and technical analysis of data. The established metrics database would also provide a strong testbed for support of future programs.

Table 1 (described in the next section) further identifies the prospective benefits of each outlined task in relation to the High Integrity Software Web. In considering options for investment of resources within this track, tasks 1 and 2 are minimum requirements. The Software Defects Database is a necessary foundation for any substantive progress in the area of predictive measurement. The actual scope of the database may be revised initially according to available resources. If a subset of the tasks are pursued, a subset of the benefits will apply and should be assessed accordingly when determining investment focus.

Critical Competencies—The three main critical competencies related to this track include: Software Engineering, Database Design, and Statistical Analysis. (Other competencies are needed for work in this track, but these three are the most critical.) Expertise is needed in each of these three areas to ensure successful investment within this proposed track. First, the various components and critical aspects of software engineering must be clearly understood in order to drive the measurement tasks. Second, the design of the database must support as yet undetermined analysis; therefore, maximum functionality must be incorporated into the design to support future studies requiring critical knowledge of current architectures, tools, and capabilities. Finally, the third critical competency requires an excellent understanding of statistical analysis. This is needed in order to verify that appropriate assumptions are made and observed in collecting, processing and analyzing the data. Otherwise, there will be no technical validity to this work.

In addition to requiring competency in these three areas, several other critical issues affect work in this area. For instance, the selected metrics must be germaine to the unique aspects of the selected projects. These projects must be judiciously hand-picked to meet specified requirements. The database will be most useful for the types of projects selected—thus the need for careful selection. (In other words, the analysis will apply to the domains of the selected projects, thus they must be characteristic of the types of projects that will be utilizing this technology.)

Considering further issues, staff must support the collection process with rigorous compliance. Spotty collection will invalidate all efforts. Consequently, staff training should be provided on how to use the statistical analysis tools and proper collection procedures as they are developed. This process may also help to identify training deficiencies. Finally, continued analysis of the data must occur to reap the benefits of the predictive process.Table 1 provides a mapping of the tasks described above to the technologies and capabilities identified in the High Integrity Software Web illustrated in Figure 1. This mapping illustrates the educational needs associated with each task and also illustrates the different technologies that will benefit from successful completion of each task.

TABLE 1. Mapping of Technologies and Capabilities to Tasks

	Task 0	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
Lifecycle Process control	X		X		X	X	
Methodology	X	X	X	X	X	X	
V & V	X			X		X	X
Risk Management	X		X	X		X	X
Estimating & Predicting	X			X		X	X
System Characterization	X		X		X		X

Critical Issues—The major critical issue to this track is that it does not provide any short term payoff to SNL software development processes. For example, the defects database must be established before objective analysis can begin, and the process itself must be iterated over several projects before some of the noted process and technical benefits will occur. Therefore, the SNL software development programs must implement the defects data collection as a minimum task, or this track cannot progress. Some analysis can begin once data has been collected for at least one project; although the results will be stronger based on the number of projects that are considered.

Also, the perceived benefit of this technology is often not accurate as the defects collection requires an initial front end effort with the expected benefit to occur at the end of the process through reduced testing efforts to achieve desired levels of predictive reliability. This perception should improve as objective evidence is obtained demonstrating validity of the techniques. Other organizations have clearly documented these improvements through such methods [7].

Because the projects selected for defect data collection are important for the domain characterization, minimal control over the selection of projects could significantly limit the analysis process and restrict the functionality (or expected payoff) of the proposed work. The critical issue here is that management must support the defects collection on those projects deemed to benefit most from the technology even though the collection process may induce additional schedule overhead.

In addition, corrective actions identified to improve SNL software development processes may not be cost effective for all projects and may be domain specific in some cases. Further, ultra-high reliability levels (0.999999) cannot be demonstrated with current reliability growth model(s), and thus cannot be achieved within the short term. Newer models will have to be developed and tested based on the objective data obtained within the defect database.

Another critical issue that could prohibit advancement in this area would be the failure to achieve staff compliance with the defect collection process and analysis methods.Frequently, staff view these activities as pointless overhead. Efforts outlined in this track are focussed on how to help staff improve the integrity of the software they produce and thus, staff should view these tasks as value-added effort for their benefit with an expected time-trade-off in reduced testing time.

Finally, if the data collection effort is not properly controlled, statistical projections will be false. And, we may not know today what information needs to be collected for the proposed predictive models. Therefore, careful consideration needs to be taken in the early phases of the predictive measurement track to optimize the potential utility of the measured data.

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 tolls to be developed within this track:

- 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;
- All system analysis and design documents will have a precise semantics (e.g., as in the Object-Oriented System Analysis(OSA) model [4] or the Business Object Notation (BON) model [8]). The semantics will serve as a basis for prototyping and visualization at every stage of system creation, from requirements capture to design;
- 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 [11], [12];
- 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;
- 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;
- Maintainability will be sound in the sense that specification, design, and code will be continuously maintained to be mutually consistent; and
- 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 [10].

Yakhnis and Yakhnis have proposed that these goals can be achieved via seamless (also see [8]) integration of OOA, OOD, code generation, visualization, and automated correctness proofs. Specifically, they suggest pursuing the following steps:

(A) Making specifications transparent and easily accessible:

Presently 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 kinds 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 such that each observable element will not contain more than seven subordinate entities;

2. represent the specification as three documents that consist 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 [4], [8], [12]. Doing so will prevent developers from distorting the requirements analysis stage by making design decisions too early;

4. make the hierarchical structure of the three documents similar. 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.

(B) Insuring that the specification captures the original idea:

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 processed all times since the feedback from the developers will be provided in several transparent forms, including visualization.

(C) Enabling 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, that the internal design of each object (i.e., subsystem) conforms to its external specification, will be computer-generated [12]. Finally, a target code that has been mathematically proven correct will be automatically generated [9].

Correct Implementation of Components

This track focuses on achieving advancements in program transformations to ensure correct implementation of components. 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 into *s'* where *s'* is an executable program belonging to some previously selected target language. Furthermore, if *T* has been shown (through formal proof) to be "correctness preserving", then we can conclude that the program *s'* is correct with respect to the specification *s*.

Victor Winter has identified several goals for this research area. The remainder of this track is broken down into two sections. The first section discusses short term and near term objectives and the second section discusses long term objectives.

Current and Near Term Goals—The "task scheduling" problem will be investigated for demonstration of our desired HIS methodology. The specific instance of task scheduling of interest to Sandia to be considered are the algorithms found in the WALS and APP projects for pit handling at Pantex.

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*.

A transformation system is suggested for this effort. A suitable choice is TAMPR, created by James Boyle at Argonne National Laboratory. TAMPR is a transformation system that views specifications, programs, and transformations in terms of syntax derivation trees (SDT's). In this paradigm, a transformation is simply a rewrite rule stating that one SDT should be rewritten into another.

At Sandia, Victor Winter is currently investigating environments and tools that will facilitate manipulating, constructing, and reasoning about transformations. Because syntax derivation trees associated with transformations tend to be quite massive, he is looking at Pad++ as a possible environment for presenting syntax derivation trees in a form more amenable to human understanding.

The correctness of a 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 created by Larry Wos and Bill McCune. Based on his experience, Vic suggests that an extension to the automated reasoning system OTTER is necessary in order to make transformation proofs more manageable.

Currently, transformation proofs require several passes, with each pass concerning itself with providing a certain portion of the overall proof. Search strategies and inference rules can vary from one pass to the next. Ideally the overall strategy of a complex proof could be defined within the automated reasoning system itself, eliminating the need for 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:

- 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.
- 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.
- 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.

Long Term Goals—There are many areas that need to be researched and further developed 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.

Critical Issues/Show Stoppers—In order for this technology to succeed, a specification language and a specification must be produced in the 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.

System Composability

System Composability is the ability to combine components to achieve a specified functionality and to understand properties of the whole from the properties of the parts. Building and maintaining large systems becomes much more manageable when systems can be composed from well understood parts.

Interest in applying this approach to software systems is evident in ARPA, the Component Ware Consortium, the computer security community, and elsewhere. As software systems have grown immensely in size and complexity, methods of producing those systems have not kept pace. Composability is a whole new approach to producing systems. It works in the hardware industry, and many are asking why not for software too?

Our interest in System Composability is not just in the efficiency of producing systems, but in how Composability can contribute to very high integrity software. At one level, simply re-using software modules that have been well proven provides some benefit; but we need to achieve a higher level of benefit, by understanding properties of the system architecture resulting from the composition.

For example, if modules have certain characterizable security, safety, or correctness attributes, is there a model that tells how to characterize the system as a whole? (Note that this is not the case for today's security implementations: connecting a B1 system to a B1 system does not yield a B1 network!) If not, we need to map out an approach for developing such a model. We also wish to learn whether there are architectures that can yield a system with higher integrity than the parts it is made of; this would be especially valuable when using COTS products.

This track should address three situations:

- Within a mature application area where domain models, architectures, and components exist, the emphasis is to appropriately adapt and reuse components, and to determine composite system properties from the properties of validated components. It may be possible to identify correctness preserving composition rules, or other desired property preserving rules. The Correct Implementation roadmap track* may yield some insights that apply here.
- When "foreign" components, e.g., COTS or modules from different domains, must be integrated into an architecture, the emphasis is on understanding mismatched properties and assumptions. Composing properties becomes very challenging, as information on component properties may not be in the desired form, and mismatches may be hard to remedy.
- When unproven components are used in a system, the emphasis is on validating them, and meanwhile protecting against them. Runtime measurement and validation approaches would allow early use of such components while confidence is being gained. Defensive architectures should also be sought, such that the system can contain the damage from an unproven component, and essentially deliver higher system surety than the "sum" of the components.

Benefits—One aspect of composability is building systems from parts, with the use of COTS products and the reuse of software modules as goals. Of course, this has great efficiency implications. Regardless, composability is seldom achievable today because of mismatched functionality and unknown assumptions. Even if a system is initially built from parts, maintaining the modular architecture in light of changes and enhancements during the maintenance phase is difficult.

Another aspect is understanding behaviors of the parts and the whole. If the surety and integrity properties for parts can be understood and measured, then we also need a model of how to compose those into a system measurement. This track emphasizes the latter, while the Measurement & Prediction track emphasizes the former. If the goals of this track can be achieved, the benefits will include:

- Efficiency in system development,
- leveraging the surety records of proven components,
- maintaining system surety in the face of low integrity components,
- maintaining system surety as parts of the system evolve, and
- leveraging and amplifying the other roadmap tracks.

Critical Issues/Show Stoppers—The goal of defining and composing characteristics of software modules is very ambitious. Little is known today about how to characterize software properties (especially surety properties), or what to measure about software, let alone what might make sense to compose. The success of this track depends on advances in the Web Technologies/Capabilities of domain modeling, architecting, risk management, most especially system characterization, and estimating & predicting. Appropriate statistical techniques must be developed with great care paid to the appropriate assumptions that will need to be made to in the development of models to guide in assessing system composability. This effort will also be greatly influenced by the success or difficulties encountered in the other roadmap tracks.

5. 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 web, expanding it 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.

Acknowledgments

We wish to thank our colleagues at Sandia National Laboratories who have contributed to the strategic surety roadmap. They include Victor Winter, Alex Yakhnis, Vladimir Yakhnis, Sharon Fletcher, Elmer Collins and the Sandia High Integrity Software Roadmap Team.

References and Notes

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

[2] Collins, Dalton, Peercy, Pollock, and Sicking, "A Review of Research and Methods for Producing High-Consequence Software", 1995 IEEE Aerospace Applications Conference, Vol 1, January 1995, pp. 197-245.

[3] "Cyberware." Time, August 21, 1995.

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

[5] Gibbs, W., "Software's Chronic Crisis", Scientific American, September 1994.

[6] Lagedec, P., "Major Technological Risk", Quoted in Safeware, System Safety and Computers, Nancy Leveson, Univ. of Washington, Addison-Wesley, 1995.

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

[8] Walden, K., Nerson, J., Seamless Object-Oriented Software Architecture, Prentice Hall, 1995.

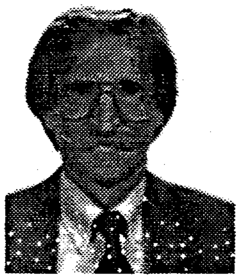
[9] Winter, V., Yakhnis, A., Yakhnis, V., High Integrity Software: Automated Software Design via Refinement Transformations, submitted to 8th Annual STC'96.

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

[11] Yakhnis, V., Farrell, J., Shultz, S., Deriving Programs Using Generic Algorithms, IBM Systems Journal, vol. 33, no. 1, pp. 158-181, 1994.

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

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 and currently is 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.



Larry J. Dalton holds a BS in Applied Mathematics and an MS in Electrical Engineering from the University of New Mexico. Larry has spent the past 18 years at Sandia National Laboratories in Albuquerque, New Mexico engaged in high-consequence systems development. He has developed personnel entry control systems for DOE high value facilities, and Aircraft Monitor and Control System for gravity nuclear weapons on board the B-52, high performance airborne multiprocessing systems for advanced concepts demonstrating and gravity nuclear weapons programmers. He currently manages the Command and Control Software Department at Sandia that develops software and systems safety solutions for high-consequence operations. These activities span nuclear weapons, biomedical applications, transportation, and information security.

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