NIST Special Publication 500-190

Computer Systems Technology

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology

NIST

Proceedings of the Workshop on High Integrity Software; Gaithersburg, MD; Jan. 22-23, 1991

Dolores R. Wallace
D. Richard Kuhn
John C. Cherniavsky

NIST Special Publication 500-190

Proceedings of the Workshop on High Integrity Software; Gaithersburg, MD; Jan. 22-23, 1991

Dolores R. Wallace
D. Richard Kuhn
John C. Cherniavsky*

Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

* National Science Foundadtion

August 1991

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
John W. Lyons, Director

Reports on Computer Systems Technology

National Institute of Standards and Technology Special Publication 500-190
Natl. Inst. Stand. Technol. Spec. Publ. 500-190, 85 pages (Aug. 1991)
CODEN: NSPUE2

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1991

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402

ABSTRACT

This paper provides information related to the National Institute of Standards and Technology (NIST) effort to coordinate an effort to produce a comprehensive set of standards and guidelines for the assurance of high integrity software. The effort may include adapting or adopting existing standards as appropriate. In particular, the paper presents the results of a Workshop on the Assurance of High Integrity Software held at NIST on January 22-23, 1991. Workshop participants addressed techniques, costs and benefits of assurance, controlled and encouraged practices, and hazard analysis. A preliminary set of recommendations was prepared and future directions for NIST activities in this area were proposed.

Keywords: assurance; computer security; controlled practices; cost-benefit; criticality assessment; formal methods; hazard analyses; high integrity systems; software safety; standards.

TABLE of CONTENTS

LIST of FIGURES

EXECUTIVE SUMMARY

The Workshop on Assurance of High Integrity Software was held at the National Institute of Standards and Technology (NIST) on January 22-23, 1991. The purpose of the workshop was to address problems related to dependence by individuals and organizations on computer systems for their physical health and safety, their financial welfare, and their quality of life. There is a proliferation of standards activity for topics such as software safety, computer security, software certification, and software quality management. The standards being developed may overlap in areas of application, and make varying or conflicting demands on software producers. The workshop explored the development of a consistent framework for standards that will be useful in assuring that critical software can be trusted to work as required. This report contains the proceedings and recommendations of the workshop.

High integrity software must be trusted to work dependably in some critical function, and whose failure to do so may have catastrophic results, such as serious injury, loss of life or property, business failure, or breach of security. Some examples include software used in automated manufacturing, avionics, air traffic control, corporate decisions and management, electronic banking, medical devices, military communications, and nuclear power plants.

Many organizations are currently developing standards to address software safety or computer security. In the United States, standards have been developed for military security, electronic banking, nuclear power, and other applications. The twelve nations of the European Community are planning an integrated market at the end of 1992 and have recognized the need for quality standards. It is expected that the European nations will require certification of quality systems employed by computer and software vendors.

It is important to understand how this proliferation of standards will affect the ability of U.S. companies to compete in the international marketplace. While many standards address individual concerns (e.g., safety or security) or specific application domains (e.g., weapons software), there is little consistency in levels of assurance or methods of showing conformance. Some standards may have conflicting requirements, leading to increased costs for companies doing business in different areas. An effort must be undertaken to ensure there is an integrated framework of standards whose requirements are technically feasible and economically practical. The NIST workshop identified technical issues that need to be resolved.

Opening presentations at the workshop described the goals of the workshop. Four working groups then addressed separate issues:

- techniques for assuring high integrity software,
- development of a cost-benefit framework of techniques,
- criticality assessment and hazard analyses, and
- controlled and encouraged practices for designing and building high integrity software.

The four working groups presented their results at a closing session. Each group arrived at some consensus for its specific issues; on the more general topics discussed in groups or at the final plenary session, the four groups had similar perspectives. Principal results included:

- NIST should undertake coordination of a framework of standards on high integrity software.
- NIST should organize liaison activities with other standards organizations.
- Similarities and differences between security and safety issues need to be studied.
- Research is needed to identify technology that is sufficiently mature for standardization.
- Methods, techniques and tools need to be put together in a framework that enables selection according to the benefits of using one or more for a particular application.
- NIST should administer trial use of a draft standard to ensure that its requirements may be implemented successfully in a cost-effective manner.
- NIST should prepare a bibliography of relevant standards and guidelines.
- NIST should conduct additional workshops to study particular issues, to address problems with coordinating standards, and to discuss the contents of a standards framework.
- Supporting research in specific techniques and trial use of the techniques to study their effectiveness and cost will be needed.

1. INTRODUCTION

In today's world, both individuals and organizations have become dependent on computer systems for their physical health and safety, their financial welfare, and their quality of life. Today there is a proliferation of standards activity for topics such as software safety, computer security, software certification, and software quality management. The standards being developed may overlap in areas of application, and make varying or conflicting demands on producers. To address the problems that may result from this situation, the Workshop on the Assurance of High Integrity Software was held at the National Institute of Standards and Technology (NIST) on January 22-23, 1991. The workshop explored the possibility of developing a consistent framework for standards that will be useful in assuring that critical software can be trusted to work as required. This report contains the proceedings and recommendations of the workshop.

High integrity software is software that must be trusted to work dependably in some critical function, and whose failure to do so may have catastrophic results, such as serious injury, loss of life or property, or disclosure of secrets. Examples include software used in automated manufacturing, avionics, air traffic control, electronic banking, military communications, nuclear power plants, and medical devices. Failure of *any* software which is *essential to the success of an organization* can be catastrophic; such software should be considered critical software whether it is a stand-alone system or a part of another system. Such software requires the assurance of high integrity [1].

High integrity software specifications are complete and correct and the software operates exactly as intended without any adverse consequences, including when circumstances outside the software cause other system failures. High integrity software will have firm adherence to the principles of its application (e.g., principles and algorithms of civil engineering for building technology, nuclear engineering for the nuclear industry). The direct user will have a complete description of the software and its descriptions (e.g., architect using software for building design has full description of algorithms and limits of the software). High integrity software is incorruptible (e.g., performs correctly, reliably, responds appropriately to incorrect stimuli; if external circumstances cause system failure, software operation shutdown does not cause any physical damage or loss of data). The indirect user (e.g., patient receiving medical treatment via automated devices, airline passenger) has reasonable assurance that the software will behave "properly," and will not cause problems due to any external malfunction.

The National Institute of Standards and Technology (NIST) has developed and adopted standards for software verification and validation [2-4] and standards for computer security [5-11]. NIST has been monitoring the development of other standards for computer security, software safety, and related disciplines. In the United States, particular attention has been placed on systems handling classified data, military weapons systems, and nuclear reactor control systems. Some standards address the integrity of these systems [11,12]; research has begun in the area known as software safety [13].

The European nations, through the Esprit program, are active in research for producing high integrity software and in standardization, both at the international level and in preparation for the European Community (EC) in 1992, of methods for producing such software. The standardization efforts range from the very specific proposals embodied in the DEF STAN 00-55 and DEF STAN 00-56 [14] to generic quality standards embodied in the ISO 9000 series of quality standards [15].

Many standards address specific application domains or categories of concerns. There is a need to bring all these efforts together in a comprehensive framework of standards to address requirements for assuring high integrity software. The framework would reduce duplication of effort, improve the understandability of the requirements, and enable demonstration of conformance to standards in the framework.

The NIST Workshop on the Assurance of High Integrity Software on January 22-23, 1991, involved a broad community of interested parties in the development of guidance for assuring high integrity software. The purpose of this, and future, workshops is to provide a forum for the discussion of technical issues and activities that NIST plans to undertake in the development of guidance. Participants in workshops will be asked to comment on technical contributions as these evolve.

NIST will distribute the workshop report to appropriate communities (e.g., standards bodies, industries that use critical software, Federal agencies with regulatory authority, developers of software). NIST will participate either directly or indirectly in related standards activities.

2. THE FIRST NIST WORKSHOP

Participants at the first workshop represented Federal agencies, the Canadian government, academia, and industry. In the opening plenary session the participants learned of the scope of efforts undertaken at NIST toward the evolution of guidance addressing the assurance of high integrity software. After the opening session, four parallel working groups addressed technical topics.

The participants in the four working groups discussed issues concerning techniques for developing and assuring high integrity software, a cost-benefit framework for selecting techniques, criticality assessment and hazard analysis, and controlled and encouraged practices for use in development. The Techniques working group was charged to determine a method for describing techniques and to identify candidate techniques. The Cost-Benefit group was charged to investigate means for selecting techniques and assurance methods. The Hazard Analysis group was charged to identify both criticality assessment and hazard analysis techniques. This group was asked to discuss differences and similarities that would be encountered if the assurance concerns were primarily security-related or safety-related. The Controlled and Encouraged Practices working group was charged to study the forbidden practices of DEF STAN 00-55 [14] and identify how best to handle them. The list of participants is provided in Appendix A; the opening remarks of the moderator of each session are provided in Appendix B.

The consensus of workshop participants was that no existing or evolving standard will satisfy the breadth of needed standards and guidance and that NIST should coordinate an effort to produce an integrated body of guidance. NIST should adapt or adopt existing standards as appropriate. Questions arose regarding the sequence of development of standards and guidance based on information the community needs:

- what the requirements are,

- demonstration that the requirements can be achieved reasonably,

- how to demonstrate conformance and to certify software.

Workshop participants will help frame the requirements for the needed standards. To help demonstrate that the requirements can be achieved, one group has proposed that NIST oversee an experiment of applying a draft standard on a real project to record feasibility and associated needs. The draft standard would be modified accordingly. Participants would include academia to help develop the experiment and industry to perform development and assurance activities. NIST is looking into the possibility and costs of conducting this experiment.

A possible vehicle for addressing some of the conformance and certification questions is the National Voluntary Laboratory Accreditation Program (NVLAP) administered by NIST. Although originally intended for accrediting laboratories that do testing of materials, NVLAP has been expanded to accredit laboratories for testing conformance to Government Open System Interconnection Profile (GOSIP) [33] and portable operating system interface (POSIX) [34] software standards. NVLAP-accredited laboratories may be a cost-effective means of ensuring that software products conform to a standard for high integrity software.

Other activities in which workshop attendees recommended NIST should have an active role included:

- Coordination of standards bodies with computer science departments to ensure that graduates are trained in assurance techniques required by standards and guidelines.

- Interaction with international standards bodies and Computer Aided Software Engineering (CASE) vendors.

- Coordination of standards bodies with vendors to enable development of tools that support requirements of standards.

- Increased visibility for high integrity software through a presentation at COMPASS '91 and by speaking to Federal agencies and at other conferences regarding these efforts for high integrity software.

- Production of a bibliography of standards and references.

- Clarification of the scope of guidance of high integrity software by establishing definitions of the basic terms.

The workshop expressed concern that a new standard may be used to tie many issues together and may also deviate from some international work.

There was consensus that this workshop did not adequately identify differences and similarities between security and safety issues. This topic required too much detail for the first meeting; the task remains an objective for future workshops. Other important topics that will be addressed in future workshops and in guidance include software process assessment, software certification, the role of accredited testing laboratories, and risk assessment.

3. THE TECHNIQUES SESSION

3.1. Overview

The Techniques working group considered development and verification techniques that can be effective in the production of high integrity systems. A diversity of experiences and interests among participants helped make this an interesting and productive session. A survey of participants showed the following interests and application areas: security, communication protocols, nuclear power systems, weapons systems, formal methods and tools research, railway systems, avionics, independent validation and verification, and quality assurance. There were no Techniques session participants from medical or financial application domains, or from CASE tool vendors, although these are equally relevant areas.

Session leader Dr. Susan Gerhart proposed a template for describing the characteristics of various techniques (fig 1). In addition to describing features of the various techniques, the template looks at how a technique fits into a development organization by considering the personnel roles involved in its use (e.g., specifier, verifier, coder). Advantages and disadvantages of tools are also considered. Members of the working group discussed the template categories and suggested a few modifications, leading to the template shown in figure 1. To evaluate the effectiveness of the template, small groups were formed to review seven methods considered useful for high integrity systems: Harlan Mills' Cleanroom method [16]; the formal specification languages EHDM [35], FDM/Inajo [36], Estelle [37], and Larch [38]; the Petri-net based tool IDEF0 [39]; and traces [18], which can be used to formally describe a system by specifying externally observable events.

Detailed evaluations of these techniques are included in Appendix C. Working group discussions of several techniques are given in section 3.2.

3.2. Review of Techniques

Small groups completed templates on seven techniques which are given in Appendix C. After completing the templates, some of the techniques were discussed by the full working group, although time did not permit a discussion of all techniques.

Verification and Validation: Software verification and validation (V&V) was discussed relative to experience with it in the nuclear power industry. The IEEE Std. 1012-1986, "Software Verification and Validation Plans," [3] is used as guidance. The main objectives are to ensure that requirements are met, to minimize common mode errors, and to detect unintended functions. An independent organization uses a requirements matrix to trace requirements to implementation, conducts tests, and prepares discrepancy reports and test reports. Any discrepancies must be resolved before a certification document can be issued. The V&V effort often involves independent generation of code by two separate teams, then a comparison of the two implementations using a large number of tests. In some cases, only object code is generated independently, using two different compilers. In others, two source programs are prepared, but the same algorithms are used in each.

Verification and Validation Assessment: V&V is thought to be effective in the nuclear industry. V&V has been conducted on very large systems, some in excess of 1,000,000 lines of non-comment source code. Its main disadvantage is its high cost.

HOW IT WORKS
Conceptual basis
Representations used
- Text, graphics, etc.
- Executable

Steps performed
- Mechanics - "transform this to that"
- Synthesis and analysis steps
- Tools used

Artifacts produced
- Documents
- Data
- Representations

Roles involved
- Person to task mapping - example: specifier, verifier
- Skills required

WHAT IT ACHIEVES WITH RESPECT TO HIGH-INTEGRITY
Positive
- Errors identified
- Evaluation data produced
- Reuse possibilities

Negative
- Fallibility - common failures, gaps in knowledge, ...
- Bottlenecks - sequential steps, limited resources, skills, ...
- Technical barriers

Other techniques
- Required
- Supported

CURRENT APPLICABILITY OF TECHNIQUE
- Domain of application?
- Where is it being used? How? Where is it taught?
- Who is researching it? Why are they doing this?
- If not in use but has potential, then what changes are needed?
- Maturity:
  Adapt/deal with change? How well does it scale?
  Who can use it? How does it fit with, e.g., prototyping?

Figure 1. Proposed template for describing techniques.

Cleanroom: The objective of Cleanroom software development [16] is to create high quality software with certifiable reliability. The term "Cleanroom" is derived from the cleanrooms used in integrated circuit fabrication, where the production process must be free from all traces of dust or dirt. Cleanroom software development attempts to prevent errors from entering the development process at all phases, from design through operation. Three roles are involved: specifiers, programmers, and testers. A specification is prepared in formal or semi-formal notation. Programmers prepare software from the specification. A separate team prepares tests that duplicate the statistical distribution of operational use. Programmers are not permitted to conduct tests; all testing is done by the test team.

Cleanroom Assessment: NASA Goddard's experience with Cleanroom has been successful, with higher quality software produced at lower cost than previous projects [17]. The initial project was approximately 30,000 lines of non-comment source code. The technique is now being used on several other projects, including one with 100,000 lines of source code. The power of the method results from having a separate party do statistical testing, plus the design simplification that results from the use of formality and from the prohibition on testing by programmers. The primary disadvantage of Cleanroom is the cost of educating programmers and testers.

Traces: A trace [18, 19] for specifications is a history of external events, in text form or logic table form. Traces are implementation independent; internal states are not specified. Traces are prepared by identifying external events and the possible sequences in which they are allowed. A set of "canonical" or non-reducible traces is used to specify the behavior of a module or function. Internal consistency is shown by verifying that all event sequences allowed by the module can be reduced to one of the canonical traces. The ability to show soundness and completeness of a specification helps to remove errors at the specification stage.

Traces Assessment: Traces were successfully used on the evaluation of the Ontario Hydro plant for the Atomic Energy Control Board of Canada. Participants also noted that they have been used at Naval Research Lab and Bell Northern Research. The Ontario Hydro project was a small shutdown system, approximately 7,000 - 10,000 lines of code. Traces have also been used to specify a communication application. Traces work well with an information hiding design and are useful for testing. Work is needed to deal with timing issues, and better notation and tools are needed as well. It is not clear how well the technique would scale up for larger projects, but participants familiar with traces believed it would be effective. Traces seem to work best at a fairly low level. Some people questioned whether it is sufficient to specify only external events, since it may sometimes be necessary to indicate internal states. While this may be a limitation in some cases, it was noted that by ignoring internal states, the representation can be changed easily, and that traces can be mapped into a state-based representation. Traces are understandable to some users, and most programmers can be taught to use them for specifications.

Statecharts: Participants were familiar with the use of statecharts [32] at ARINC, the University of Maryland, NIST, and Rail Transportation Systems. Projects included a transaction processing protocol and railway system, although the project abandoned the use of statecharts before the project was complete. Statecharts can be used to specify and design systems using a graphical interface to show control flow, data flow, and conditions. They seem to be effective for clustering states. A commercial product implementing statecharts includes a graphical editor, reachability analyzer, and some code generation capability. A flexible document producer is also included.

Statecharts Assessment: Statecharts appear to be a good tool for initial design, but are not effective for detailed specifications. They are easy to learn and available tools are easy to use for prototyping. Statechart specifications are adaptable to changes in requirements. Statecharts do not seem to scale up well since large specifications become difficult to read. They are hard to use for applications that require multiple copies of the same type of object because a copy must be built of each instance.

IDEF0: IDEF0 [39] is a Petri-net tool that is useful for specifying real-time systems. It was developed for the U.S. Air Force and has been used primarily on nuclear weapons systems. It can be used for design of concurrent and distributed systems. A graphical editor is used to prepare Petri net diagrams. Reachability analyses can be conducted on the nets to look for timing problems and race conditions. It includes a report generator that can produce reports required by MIL-STD 2167A [20], diagrams, and error reports. A data dictionary that can be exported to other tools is also included.

IDEF0 Assessment: IDEF0 is useful for identifying race conditions and incompleteness. It assists in reuse of specifications by identifying "like" portions of the nets. IDEF0 does not scale up well because of the complexity of its diagrams.

3.2.1. Discussion

The tools and techniques discussed have different strengths. All are useful for assurance of high integrity software, although none is comprehensive enough to be used alone. Proper matching of techniques to problems is needed. Application domain, project organization and personnel skills must also be considered. A high integrity software assurance standard could identify a set of techniques and associate them with the problems the techniques are considered acceptable for addressing. Participants did not believe that any particular set of techniques should be required for all high integrity software. Technologies are not equally applicable to all types of applications, so application domain specific standards may be useful. Working group participants sought to make the template categories sufficiently detailed for intelligent selection of techniques, either by developers or for application specific standards.

3.3. An Assurance Model

A model of assurance levels was proposed, shown in figure 2. Working group participants agreed that the proposed model does a good job of structuring assurance levels based on formal methods. But no claim is made that increased integrity is guaranteed by higher levels of the model, since the model represents only one axis of a many dimensional problem.

3.4. Economics and Practicality of Techniques

Working group members pointed out that techniques that are not economically practical today may become so with improvements in technology and education. An economically profitable field such as computing evolves rapidly. Advances in technology such as improved software tools, faster processors, and better graphics may make some techniques more practical. Education is perhaps an even more important

Level	Technique	Examples
3	mechanical verification	ASOS, LOCK, FM8502
2	formal spec + hand proof	VIPER, Leveson's method, traces
1	formal spec only	Z, VDM, control law diagrams
1/2	pseudo-formal spec	statecharts
0	static code analysis	SPADE, MALPAS

Figure 2. Assurance levels with formal methods.

determinant of the usefulness of a technique. Many programmers today do not have computer science educations, and often even those who do may not have the necessary background to use techniques such as formal verification. As more people become available with the necessary skills, developers with undergraduate educations may be able to use techniques that often require graduate level education today.

These facts have several implications for a high integrity software standard. The standard must be written to accommodate improvements in technology and education. It would be a mistake to prescribe only a limited set of techniques that are in use today. Instead, advanced techniques can be included as options to be selected as the user determines necessary. The standard must be coordinated with university curricula as well. Appropriate education must be available for techniques specified in the standard.

3.5. Safety vs. Security

Many people appear to be unhappy with the Trusted Computer Security Evaluation Criteria (TCSEC), or "Orange Book" [11]. It was noted that much of the dissatisfaction with the TCSEC results from its rather "technology specific" approach to assurance. Designed for evaluating multi-level security in mainframe operating systems, the TCSEC is becoming outdated now that many systems are distributed and network-based. (The Trusted Network Interpretation does not address all problems of distributed systems.) Also, multi-level security is not relevant to many commercial applications. As a result, the TCSEC is inadequate for evaluating security in many commercial systems.

The group considered this experience relevant to development of high integrity software standards, since any such standard might have similar limitations. The standard must be flexible to deal with advances in technology. The second lesson to be drawn from the TCSEC experience is that probably no standard could be applied to all application areas. A "Framework" proposal developed by the U.K. Department of Trade and Industry [21] describes an approach to developing a set of standards for high integrity software. Industry-specific components of the standards set will need to be developed because different applications have different needs and different approaches to assuring integrity may be necessary.

The aircraft industry safety standard DO178A [22] is now being revised to DO178B. The nuclear industry has relatively little in safety standards applicable to software but is in the process of developing standards and recommendations on tools. German and French CASE tools for specification and requirements were mentioned.

The railroad industry is looking for standards. The industry distinguishes between safety-critical and failsafe. Currently there are no standards with which a safety assurance level can be judged. There are four types of design for assurance: check redundancy for hardware failure; diverse methods across platforms; different platforms to compare operations; numerical assurance allows errors but calculate the probability that error will have an undesired effect. The industry uses experience to know that a design is right, but there is still a difficulty in knowing if the system will be safe in the event of a hardware failure.

The nuclear industry assumes that no component is fully safe. It doesn't believe any failure probability figure less than 10^{-4} (a practical limit of measurement techniques), so systems must be designed to limit the consequences of failure.

3.6. Recommendations

The group prepared a set of recommendations for inclusion in a standard for high integrity software. The recommendations are necessarily preliminary, but there was a good deal of consensus among participants.

Respect the "practical assurance" limit. With current technology it takes about one year of testing to assure a system of correct operation for 1h with a failure probability of 10^{-4}. It was noted that this can be bettered with N-version programs if one assumes independence of versions. Based on empirical studies, group participants doubt the validity of N-version independence [40].

A standard should state characteristics of techniques and require arguments as to why a technique selected is appropriate. The group felt that techniques are not equally applicable to all application domains. A developer who wishes to claim conformance to a high-integrity software standard will need to describe the characteristics of the application and give a convincing argument as to why the techniques used are appropriate.

A clear implication of this recommendation is that a single all-encompassing standard for high integrity software is not practical unless it is simply a catalog of techniques. Requirements for specific techniques will need to be based on application domain characteristics. This is in line with the "framework" approach of having a standard that gives general requirements, supplemented by standards to an appropriate level of specificity for different application areas.

Evaluate and track on-going application of techniques. It is essential to monitor applications of different techniques to determine which are most cost effective for different applications. An equally important aspect of tracking application is to make techniques more widely known in the industry. Many significant techniques are little used today because practitioners are not aware of them, or because they are perceived as too expensive or impractical. Measuring the costs and benefits associated with various techniques will allow decisions to use techniques to be based on sound data rather than guesswork.

Distinguish between techniques used to eliminate design flaws and techniques used to maintain quality in the presence of physical failures. High integrity systems will require the use of both types of techniques. Determining the optimal tradeoff between fault tolerance and fault elimination for a particular application is a challenging problem.

Experience and empirical research will be necessary for designers to make this tradeoff. A standard should provide a selection of both types of techniques, and guidance should consolidate experience to help developers make choices between the techniques.

It was noted that the most important part of a recommendation on techniques is to point out fallibilities. All techniques have limitations; by noting these, developers will be able to compensate for the limitations or at least attach appropriate caveats for purchasers.

It was also recommended that a notation to express what techniques were used at different stages of the lifecycle be developed. Such a notation would facilitate specification of development requirements, and could also be used to characterize developments to make it easier to compare projects.

3.7. Session Summary

The group selected seven techniques to describe with the template. Group members thought a catalog of techniques described using the template would be essential for a standard for high integrity software. A critical aspect of the template is the description of the limitations and areas of applicability for each technique. Technologies are not equally applicable to all types of applications.

A useful reference list for practitioners and researchers was also prepared. The list, given in section 3.8, includes names of annual conferences and workshops as well as books and articles that address high integrity software. The group requested that NIST prepare a bibliography of safety and security related standards.

A discussion of experiences with safety and security standards was helpful in building recommendations for a high integrity software standard. The set of recommendations given in the previous section will help avoid some of the pitfalls associated with safety and security standards in the past.

3.8. Resources on High Integrity Issues

Members of the working group selected reading material for an initial resource list. Proceedings of at least three annual conferences occurring in the Washington, DC area are usually available in libraries or from the conference sponsors:

- the COMPASS conference series,
- NASA Goddard Software Engineering Laboratory Workshops,
- National Computer Security Conference.

A book that may be of interest is by C. Sennett, *High Integrity Software* [1]. Three IEEE publications were synchronized so that the September 1990 issues of *COMPUTER, IEEE Software, IEEE Transactions on Software Engineering* addressed formal methods. Wallace and Fujii edited the May 1989 issue of *IEEE Software* which addressed software verification and validation.

A report that should be read by anyone interested in high integrity systems is *Computers at Risk,* edited by D. Clark, (National Academy Press, 1991.) Other publications on assurance issues include the *FM89 Proceedings* (Springer-Verlag), from a conference on formal methods, and the IFIP Working Group 1 - Protocol Specification and Verification Series (1981 - present.)

4. THE COST-BENEFIT SESSION

The Cost-Benefit group, chaired by John Knight, outlined a basic set of studies and tasks to support development of a cost-benefit framework for assuring high integrity software. Consensus was reached that such a framework will have to address many application domains as well as specific quality concerns. In this sense, there is concern that no single standard that may evolve from a framework will satisfy the needs of all application domains. The framework will have to provide direction relative to selecting techniques and practices that are appropriate, within a reasonable cost, for levels of assurance.

The Cost-Benefit group selected topics relevant to making more specific the general concepts of such a framework. First, definitions of key words are essential for establishing the scope of a framework. Second, selection of an initial cost-benefit model requires understanding of key elements of the model and the types of contributions from the other workshop groups. Hence, relationships among the groups must be understood. Third, usage of the model and any draft standard(s) that may evolve from it should be shown to be feasible. Discussions of these topics are presented in the remainder of section 4 of this report.

4.1. Definitions

Definitions of the basic terms and concepts need to be established so that the scope and frame of reference for a cost-benefit framework will be clear. Such a framework may eventually be used in standards addressing high integrity software. The working group has suggested definitions for *cost, benefit, high integrity, software, relevant application domains,* and *users of a cost-benefit framework.* While there are several definitions of software safety, one must be chosen that will encompass the scope of any cost-benefit framework and subsequent standard on the assurance of high integrity software. Any new standard must identify terms and their definitions that may already exist with different definitions in other standards.

Cost: The definition of types of costs is more appropriate than expressing costs in dollars. While project data on costs may exist, locating the data and getting companies to release the data will be extremely difficult. Work should progress on an alternative cost-benefit model using general concepts. This approach will describe the positive and negative attributes of techniques in different application domains.

Part of the charter for this group was to determine what should be automated based on the cost of automation. The costs of automating a function are related to the costs of the techniques used to develop and assure those functions. The decision of whether a function can be economically developed as a high integrity function should be based on an understanding of the overall process involving both development and assurance. Some functions may be too expensive to automate without using automated techniques. While many techniques have been used as research tools and in industrial applications, many have not been automated. Some functions currently may be too costly to automate but as the demand for automated functionality increases, so will the demand for automation to build and assure the functionality. When those automated techniques become economically practical, then new functionality also may become practical.

Other costs associated with building and certifying systems may be very difficult to quantify due to their political or long term safety and economical natures. One example is the choice between building a certified nuclear plant which is not affordable versus the risk that the loss of nuclear power in future years may be a greater cost. Another example comes from the business community. Sometimes a company may not even attempt to automate certain functions that would greatly enhance business opportunities because of uncertainty that the confidentiality and integrity requirements can be fulfilled. One approach to the problem of quantifying cost may avoid the issue by identifying positive benefits and negative consequences. Some items in either category may be quantifiable but others may be immeasurable.

Two categories of cost that were itemized are the cost of failure and the cost of failure prevention. Again, cost in these contexts may not always be measurable.

The cost of failure includes:

Immediate costs: loss of property; injury; cost to restore service; cost of failure investigation.

Intermediate Costs: root cause analysis; product improvement; assurance of improved service.

Long Term Costs: loss of reputation; loss of market; political costs; replacement costs; litigation costs.

The cost of failure prevention includes:

Direct Costs: cost of software development (includes assurance activities); cost of training, cost of money.

Indirect Costs: opportunity costs, throughput penalties; development delays.

Benefit: Benefits may not always be measurable. A framework must take into account all benefits, which may be perceived as the avoidance of the cost of failure.

High Integrity: A concise definition of high integrity may not be appropriate. Rather, the sense of high integrity is important and there should be guidelines explaining appropriate, but not exclusive, applications. The group accepted the description of high integrity provided in the introduction of this report.

The approach that "high integrity systems include those whose failure could lead to injury, loss of life or property" is acceptable. The sense of high integrity that is evolving also appears to be very closely related to the definition of dependability suggested by Carter: "trustworthiness of a computer system such that reliance can justifiably be placed on the service it delivers [23]."

While this group's primary concern was in developing a cost-benefit model, it recognized that the model may affect standards on high integrity software. For that reason, it is important to identify a definition of software safety that will be used for this work on high integrity software. Different definitions will determine different scopes of work to be covered by standards.

Software: The definition for software in IEEE Std. 729-1990 Standard of Software Engineering Terms [24] is the following: Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.

While the definition for software in ISO Std. 2832/1 [25] is similar, it contains a clause related to legal aspects: The programs, procedures, and any associated documentation pertaining to the operation of a data processing system. Software is an "intellectual creation" that is independent of the medium on which it is recorded.

A cost-benefit framework for assuring high integrity software should address all types of software (e.g., microcode). Any standard using this framework should also distinguish between software for the application being developed and software support tools (e.g., compiler). There may be a substantial cost factor when the support software is also considered high integrity software.

The consensus was that for now a standard should leave it to the user to determine if support software is covered by the standard. Other draft standards need to be examined with respect to their requirements for support software. This issue raises the question about the requirements for sub-suppliers to the principal suppliers of support software. In particular, the ISO 9000 series [15] addresses some of these issues and needs to be studied carefully.

Relevant Application Domains: Obvious relevant domains are those in which failure may mean loss of life or property. Medical devices, air traffic control systems, nuclear power plants are relevant applications. When loss of mission is the consequence of failure, then criticality of that system may be in the "eyes of the beholder." The consensus of the working group is that a standard should not specify the application domains. Agencies who adopt the standard may specify the application domains for which the standard is required. Other users may determine if high integrity software is appropriate for their systems.

Users of the Cost-Benefit Framework: The cost-benefit framework should be used by both buyer and the supplier. (For this workshop, supplier will be used to refer to anyone who develops systems, maintains systems, or provides assurance functions.) The buyer needs to identify to what degree the proposed system requires high integrity; the guidance on criticality assessment and hazard analysis will be of value. The framework will help the buyer to determine if, dependent on the level of high integrity, the system is affordable or if its requirements should be reconsidered. The buyer may also wish to use the framework to check the proposed assurance approach.

The suppliers may use the framework either because a contract requires them to use it or to determine what approach is suitable for the system they are going to build, maintain, or provide assurance for.

4.2. Relationship of Working Groups to a Framework

The Cost-Benefit working group recognized that their ability to develop a framework for selecting techniques is contingent upon the quality of data provided from the other working groups. The workshop session on Hazard Analysis has provided various perspectives on how to determine if a system needs high integrity software. The application of criticality assessment and hazard analyses can provide a mechanism for determining which parts of the system are especially risky. These parts require strongest assurance. The Cost-Benefit working group recognized that the outcome of the Hazard Analysis session is crucial for establishing a foundation on which any framework selection techniques will be based.

The Techniques session will provide information about specific assurance techniques. In particular, the Cost-Benefit group will rely on the technique templates to identify the types of errors a given technique is most likely to prevent or to uncover. It is important to identify the application domains where these errors are likely to occur. The Controlled and Encouraged Practices session will provide guidance on design methods and code practices. The Cost-Benefit group must identify other parameters and must coordinate results from the various working groups to build a selection framework.

4.3. Model

The Cost-Benefit working group was pessimistic that one standard can satisfy all application domains, development environments, and user environments. A basis for a set of standard for high integrity software may be a cost-benefit framework flexible enough to satisfy many users. The framework would include sufficient guidance on selecting development and assurance techniques that are affordable and suit the assurance needs of a user. A mathematical model was proposed as a foundation for such a framework [26,27,28]. The model, shown in figure 3 as modified by the group, seeks to find the minimum of the total cost of two costs, that is, the cost of failures per unit time plus the cost of development and assurance. The cost components must be assessed over the estimated life of the software or the system, with discounting of the costs which are incurred at a future time (the discounting reduces the sensitivity of the model to errors in the estimation of life.) Alternatively, the cost components can be assessed for a limited period of time, say a year, if the cost of development and assurance can be apportioned to be less than the full life interval. The working group recommends use of this model only as a starting point for identifying the parameters that must be built into a selection framework.

A complete model must associate techniques with error types, application domains, and other items to consider relative to a supplier's environment. One suggestion is to associate a probability of failure with a set of techniques as indicated in figure 3. Considerable research is necessary to determine exactly what those techniques are. The original objective was to associate a required level of assurance with a set of techniques. Is a level of assurance identical with a probability of failure per unit time? That is, is all assurance simply a matter of reliability?

Another issue concerns the grouping of methods in general. For example, will techniques highly suited to locating timing errors, a concern of many critical real time systems, be included in Set A? Will others be included in Set B? Extrapolation of this thought brings up the question: What error classes will be covered by techniques in each

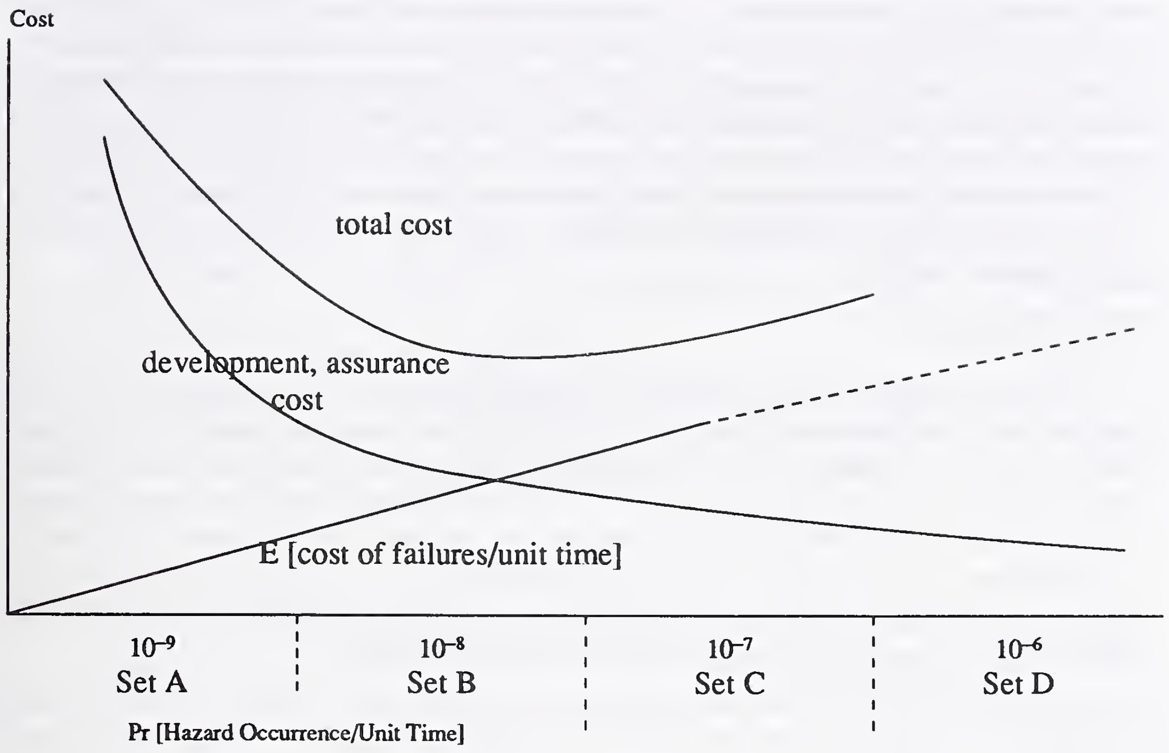


Figure 3. Proposed cost-benefit model for technique selection.

of the sets? Can application domains be characterized by their error classes? Does a set of methods then refer equally to application domains and to error classes? Of course, this model focuses on errors, whereas other requirements for assurance may focus on specific qualities (e.g., maintainability, portability, efficiency). It is not immediately obvious that this model, even with sets of techniques, will accommodate selection of techniques based on the qualities they support. How should applicability of any set of methods be described?

An issue that was barely touched concerns the tradeoffs of techniques. The templates that are to be developed by the techniques working group should provide interesting data as to what a specific technique can do, but someone will have to study all the techniques to further categorize them in terms of their best partners or worst partners according to what their purpose is. The proposed model does not appear to address any of the concerns about the capability of a supplier to perform specific techniques. While the templates may address this concern because the template does have a parameter called training, the group recommends discussion on whether or not the Software Engineering Institute's levels of maturity [29] should be incorporated into the overall sets of techniques.

Further development of this model requires the collection of data on failures of systems, types of techniques for development and assurance and the errors they prevented or discovered, and the costs associated with the failures and successes. One possible source of data is the NASA Goddard Space Flight Center's Software Engineering

Laboratory [30]. It is not clear if the data collection task should be pursued with the intent that such data would result in a "hard-coded" framework or if the objective should be to lay out a model that users may tailor to their own projects. In this case, users would study data from their environment; that data would have to be of the same type already identified but on a much smaller scale. According to Dr. Victor Basili, due to differences in environments, experiences satisfactory in one environment may become unsatisfactory in another [31]. The working group needs to study how problems such as these will affect generic models.

4.4. Experiment

Implementing a standard may mean major changes in the way software is developed and assured. Suppliers may have to provide specialized training for their staffs, and may have to invest in software tools. Training may be needed to help managers understand scheduling for new tasks or new ways of doing traditional tasks and other resource changes. Some of the proposed requirements may be difficult to implement and may not be affordable. The working group strongly recommends that a draft standard should be applied to an industry project. Data from the experiment should influence changes to the draft. The basic structure of the experiment is shown in figure 4.

WHY

- Trial run of the standard to show feasibility

- Acquire performance and cost data on advocated methods

WHAT

- Development according to a draft standard of a realistic sample product in a typical industrial setting

- Measurement of predefined metrics and acquisition of relevant artifacts.

HOW

- NIST, industry, academia form a team
- Find funding
- Prepare strawman draft standard in parallel with planning/preparing experiment

Figure 4. Proposed structure for trial use of draft standards.

4.5. Session Summary

The Cost-Benefit group strongly recommends that the fundamental terms be defined first, especially software safety. The definitions will define the scope of standards that will be proposed by NIST for high integrity software.

The Cost-Benefit group considers the model in figure 3 a starting point for determining selection of techniques. Support of this concept will require the collection of data, much of which may not be easily available. Cost may not be quantifiable or even predictable (i.e., it might be intangible) because of factors like political effects. Cost analysis may require quantification of even intangible items like political fallout. The group will consider other ways of measuring the input to a framework. The concept of a framework itself implies the development of several standards and guidelines.

Development of any standards for high integrity software must also include means of demonstrating conformance to the standards. It must be shown that the requirements of these standards can be met at reasonable cost.

The final point the Cost-Benefit group emphasized is their belief that while a standard may support certification of high integrity software, no procedures or standards can guarantee that a specific system achieves total freedom from defects.

5. THE CONTROLLED AND ENCOURAGED PRACTICES SESSION

This session was charged to review the history and international standing on practices which have been forbidden or discouraged by some software development standards. Examples of these practices include interrupts, floating point arithmetic, and dynamic memory allocation. A well known example of a standard containing such prohibitions is the British Ministry of Defence DEF-STAN-0055.[1] The DEF-STAN-0055 prohibitions were based upon the difficulty of assessing code that uses these programming practices, not because the practices themselves are inherently dangerous. Initial proposals and assumptions consisted of moderator Arch McKinlay's prepared tutorial on international standards and their background, proposed definitions, and proposed forbidden practices based on his experience in software safety engineering.

After review and discussion of other standards and their approach to error-prone practices, the group redefined "forbidden practices." The new definition hinged on the concept of "controlled" versus "forbidden" practices. No one believed that all instances of the "forbidden practices" were in fact unsafe, and those that currently are, may be safe next year if certain technologies develop. This view reflects the comments from the Institution of Electrical Engineers and others concerning the DEF-STAN-0055 standard. Other standards discourage but do not prohibit certain practices.

The group adopted this definition of a Controlled Practice:

> A Controlled Practice is a software development procedure which, when used in safety-critical systems in an inappropriate manner, results in an increased level of risk. This practice is one which can reasonably be expected to result in a safety hazard, or, is not demonstrably analyzable.

[1] At the time of the workshop, the participants had access only to the Draft DEF-STAN-0055, and addressed the "forbidden practices" issue accordingly. The recently-released INTERIM STANDARD DEF-STAN-0055 has relaxed its policies on discouraged or forbidden practices.

5.1. Encouraged Practices

Certain software practices, although not inherently dangerous, are generally recognized as increasing the incidence of software failure and hence the risk in safety-critical systems. These same practices may be less error prone when certain checks and balances are employed in their use. That is, these "risky" practices inject a certain error type and may only be used in conjunction with other practices which have been shown to detect, mitigate, or counter the error type injected.

Initially the group thought that "encouraged practices" could also be required to offset "controlled practices" in certain circumstances. Later discussion showed that some "good" practices (e.g., use of higher order languages), should be encouraged but not forced or controlled as tightly. Thus, group consensus allowed the change to "Controlled and Encouraged Practices." It was later noted that this would allow developers to choose various combinations of techniques (some familiar to them and others not familiar) as long as the error types were "covered."

5.2. Software Integrity versus Controlled and Encouraged Practices

The group decided many factors influenced the risk of the same practice in different domains and applications. A matrix idea was formed in which such factors would allow a developer to select enough countering practices to allow the use of a "controlled" (not forbidden) practice. This matrix concept will be driven by the level of software integrity required. The required software integrity level is further a result of the hazards identified with the system and allocated to the software (and hardware) (sub)system under design.

5.3. Liaisons With Other Working Groups

It was noted that these definitions and matrices depend on other groups for definitions and declarations of hazards, safety-criticality, and integrity level. Required definitions include *hazards, risk,* and *integrity levels.* Originally it was thought that an integrity level would determine the controlled practices. On further reflection, the group reached the consensus that there may be different levels of rigor, or controlled practices, in a technique required for a particular integrity level. The briefing by the Hazard Analysis session moderator at the plenary session implied that the Hazard Analysis session would not address integrity levels.

Given that a controlled practice is used, the next step is to select mitigating techniques. The Techniques Session addressed coverage (and what the technique does not find), input requirements, output, and integrity level rigor. This information may be used to select offsetting techniques when one of the "controlled practices" is to be used in development. To use the Techniques output in this way requires a considerable amount of experience and studies of the various techniques. It may be many years before such an experience base exists.

The Cost-Benefits session must somehow determine the costs of each technique such that the selection of mitigating techniques versus the controlled practice can be constrained by prudent cost considerations. There may be correlation with DEF-STAN-0055 in that "forbidden practices" may be the most expensive practices to implement under "controlled practices," after selection of the techniques required for the integrity level.

5.4. International Issues

There was a consensus that the view of "controlled and encouraged practices" expressed in the working group is different from that in the international standards reviewed. Accommodation of this difference requires two definite steps:

1. The definitions of controlled and encouraged practices must map onto all known standards which have such concepts, and
2. The international community must be made aware of the intent of these redefinitions.

Item 2 raises a particularly strong issue because the DEF STAN 0055 forbids practices while a U.S. standard will probably allow practices with certain rigorous development practices. This seems to be diametrically opposed and may preclude mapping one to the other. While the DEF STAN 0055 is a draft standard with comments being addressed, it is being considered as a baseline for EC standards. Thus, NIST must move earnestly to promote public discussion of differences and coordinate less strict status on certain practices.

5.5. Classification of Controlled Practices

The moderator presented the concept of identifying certain error types found in generic software development processes. These processes are not identified with a development paradigm (waterfall or spiral); rather these processes are required in any paradigm in whatever order or parallelism.

These processes are:

- Management Process
- Concept Definition Process (system level)
- Requirements Specification Process
- Design Process
- Implementation Process
- Integration and Change Process
- Testing/Safety Validation Process
- Use Process (system robustness)

For example, the management process includes practices related to configuration management, quality management, risk management, and training/staff qualifications and requirements.

In addition, the concept of linking controlled or encouraged practices to integrity levels and application domains will define a minimum set of practices required for one to develop high integrity or safety-related software. The life cycle model and other specifics are still left to the developer's choosing due to the generic nature of the processes and the long list of alternating coverages provided in the techniques matrix while still achieving the required integrity level.

5.6. Session Summary

The group advised continuing group efforts on development of concepts, mapping of *integrity level* to *techniques* to *controlled/encouraged practices* and mapping to other standards. A list of controlled/encouraged practices should be prepared for consideration by the Techniques group. Liaisons should be established with other groups. There must be discussion of unresolved issues (e.g., indirect effects of proposed control of listed activities and indirect hazards of such software as finite element analysis tools used on passenger aircraft or highway bridges). The relevant national/international organizations developing related or similar standards need to be contacted. It is recommended that workshop representatives participate in COMPASS '91 and send notices to CASE vendors and universities.

6. THE HAZARD ANALYSIS SESSION

The working group on Hazard Analysis was chaired by Michael Brown of the Naval Surface Warfare Center. This session had the task of defining the terms and techniques for hazard identification and analysis of software for which assurance of high integrity is desired. Experts from both the military and civilian sector were present. Two different documents were used as initial examples of the sort of activities that might be present in dealing with this sort of analysis. The first was the "Orange Book" [11]; the second is MIL-STD-882B, the DoD systems safety handbook [12]. Although the Orange Book does not directly address hazards, the fact that it describes assurance levels for certain types of potential security breaches makes it relevant because, as mentioned in DEF-STAN-0055 [14], security breaches can be viewed as hazards. The initial objective of the session was to identify techniques for

1. identifying hazards,
2. classifying hazards,
3. identifying critical systems,
4. determining how much analysis is necessary,
5. determining where to do analysis, and
6. conducting trade-offs.

In order to accomplish this objective, agreement was needed on many of the terms. In particular, the terms *hazard, risk, and criticality* all needed definition in the context of high integrity software. Analogies were drawn from a number of areas in pursuit of common definitions to apply to high integrity software. From the system safety area hazards and risks are well defined in MIL-STD-882B. The events to be avoided are injury and death; the hazards are elements of the environment that can cause these events. From the perspective of a mission during wartime the events to be avoided are the inability to fulfill a mission and the hazards are elements of the mission environment that can cause these events. From the perspective of security the events to be avoided are security breaches and the hazards are elements of the environment whose presence or absence can allow these events to occur. From the perspective of the manufacturer of consumer products containing embedded software, the events to be avoided are losses caused by insufficiencies in the product that result in financial losses and the hazards are elements of the products environment that could allow these events to occur. These events could be as simple as, say, a ROM error in a chip in a washing machine requiring a recall costing $100/machine. The events to be avoided are called mishaps.

6.1. Basic Notions of Hazard Analysis

Given the wide variation in the events to be avoided, the following definitions of mishap and hazard were adopted.

Mishap - An unintended event that causes an undesirable outcome.

Hazard - A condition that may lead to a mishap.

Given this definition of hazard, the notion of risk is defined to be a function of the hazard, the vulnerability to the hazard, and the opportunity of the associated mishap occurring. The vulnerability and opportunity are assessed together to obtain a probability of the mishap occurring. Once a rough probability has been obtained, decisions are made as to the criticality of the hazards in order to determine whether actions are necessary to mitigate the hazard (or if the consequences are sufficiently severe, not to build the system). As an example, consider a nuclear power reactor and the hazards posed by meteor strikes and earthquakes. The vulnerability of the reactor to a meteor strike is high while the opportunity of the mishap occurring is very low. Thus actions aren't taken to mitigate the hazard (reactors aren't built under a mile of rock) even though the consequences of reactor failure are severe. The vulnerability of a reactor to an earthquake is high and the opportunity for occurrence, particularly on fault lines, is sufficiently high with consequences sufficiently severe (a function of the hazard) that actions are taken, such as building away from fault lines, to mitigate the hazard.

One method of performing this analysis involves building a hazard criticality chart, as illustrated in figure 5. The method is described further in MIL-STD-882B. The letters A-E stand for probabilities of a particular hazard resulting in a mishap. A is the most frequently occurring (nominally ''frequent'') while E is the least frequently occurring (nominally ''improbable''). The Roman numerals I through IV represent the severity of a mishap caused by the hazard. For the safety concerns of MIL-STD-882B, I stands for death or system loss, II stands for a severe occupational illness or major system damage, III stands for minor injury or minor system damage, IV is negligible injury or damage. The regions labeled 1 to 4 are determined by policy. In MIL-STD-882B region 1 is unacceptable, region 2 is undesirable, region 3 is acceptable with approval, and region 4 is acceptable. For each hazard, determined by a careful analysis of the environment in which the system is operating, such a chart is drawn up. Values for the probabilities of a hazard occurring and the severity of a mishap arising from the hazard are determined. If these are in the unacceptable or undesirable range then steps must be taken to mitigate the severity of the mishap and/or reduce the probability of the hazard resulting in a mishap.

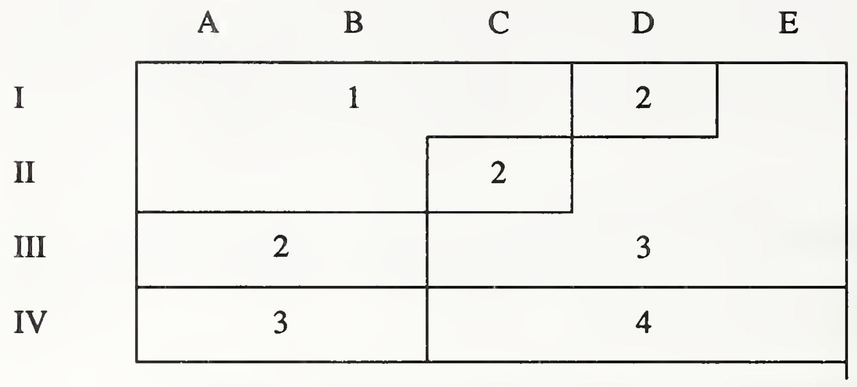


Figure 5. Hazard Criticality Chart.

This same hazard analysis chart can be adapted to high integrity domains outside of safety. What needs to be identified are the roman numeral categories I to IV. For example, to analyze hazards for military missions I would correspond to an inability to fulfill the primary mission capabilities (e.g., field an army in a war zone), II would correspond to an inability to fulfill a secondary mission (e.g., an impaired offensive capability against a collection of targets), III would correspond to an inability to fulfill support functions, and IV would correspond to an inability to fulfill administrative functions. In a consumer product the categories I to IV could correspond to *I: product causes death or injury, II: product causes damage resulting in financial loss to consumer, III: product does not perform its function resulting in financial loss to company, IV: product does not satisfy some consumers in ways unrelated to functionality, cost, or death or injury.*

Techniques for identifying and classifying hazards and for determining the risk associated with hazards is domain dependent. Generic methods include "lessons learned" (historical information about previous mishaps), analysis of energy barriers and the tracing of energy flows (mishaps are frequently associated with energy release or containment), previous system analyses, adverse environment scenarios, general engineering experience, and tiger team attacks (essentially brainstorming).

Specific techniques in tracing possible effects of hazards and isolating those effects have been developed over the last 40 years. These include fault tree analysis, failure modes, effects and criticality analysis, event tree analysis, and hazard and operability studies. At the code level formal proof of correctness and various data and control flow analyses can be performed [13]. Isolating parts of the system responsible for assuring high integrity is an important method of limiting the complexity of the analysis necessary for assurance. This is exemplified in the notion of a Trusted Computer Base that is integral to the TCSEC ("Orange Book") [11] standards. The design techniques for this sort of isolation include the isolation of critical functions in kernels, assurance of module independence ideally through referential transparency of modules, and the general isolation from access of critical software and data.

6.2. Lifecycle Hazard Analysis Activities

The criticality assessment must be carried throughout all of the software development phases, implying a strong traceability requirement for hazards that must be avoided or mitigated. This also implies documentation requirements at all lifecycle phases for hazards being traced. Modifying the language of the software systems safety community, it is necessary to have a Software Integrity Preliminary Plan, Software Integrity Subsystem Plans, Software Integrity Integration Plans, etc. as described in MIL-STD-882B and below.

Taking a typical software product and using lifecycle stages from the above figure we can categorize, according to lifecycle state, the types of analyses that must be performed to assure high integrity.

During initial project planning, two decisions are made. The first is whether to automate a system to satisfy that need. The second is to determine, roughly, the integrity requirements for the product. Both of these decisions involve a high level hazard analysis. For example, in the planning of an airliner and its avionics, a decision must be made concerning the degree to which the pilot's duties should be automated. The consequences of a computer failure during critical flight maneuvers and the capability to actually automate pilot's duties are taken into account at this pre-requirements stage. The early planning largely defines the system and its environment allowing hazards to be identified. It is at this stage that an integrity policy should be put in place. This policy would identify the personnel responsible for the integrity of the system, the level (and thus required activities) of integrity desired, and the resources required for assuring the desired level of integrity.

At the requirements stage an *integrity model* should be put in place. This model should reflect the integrity policy put in place at the conceptual stage. The model should be sufficiently analyzed to ensure that it accurately reflects the desired type and level of integrity. A preliminary integrity analysis would be performed at this level. The need for isolation of system components critical to the integrity of the system is decided at the requirements stage as well. Requirements traceability policy and high level test requirements for hazard coverage are determined at this stage. Test plans are constructed and test cases to stress the system are developed. Formal methods may be used as required by the integrity policy and model.

At the preliminary design stage, components critical to the integrity of the system are identified and traced back to the requirements. It is at this stage that the isolation of components critical to the integrity of the system is enforced. Test cases and test code are again generated to ensure that the hazards are covered. Standard techniques such as software fault tree analysis are used to identify critical software components, and the design is analyzed to ensure that no new hazards are introduced as a result of design decisions. Formal methods may be used as required by the integrity policy and model.

At the detailed design level futher analysis is conducted to evaluate traceability for the identified hazards, and to ensure that no new hazards are introduced at this stage. Again test cases and test code is generated to ensure that the hazards are covered. Formal methods may be used as required by the integrity policy and model.

At the code stage traceability is enforced and coding practices adopted that reduce the possibility of hazards being introduced at this stage. Formal methods may be used as required by the integrity policy and model.

Throughout this process, standard software quality assurance activities are followed. Quality assurance is a prerequisite for high integrity software. Assurance includes checking that the software addresses the hazards, and developing tests that "exercise" the software in response to external events that may lead to a hazard. This testing requires (as does traceability, etc.) that the software addressing hazards be properly isolated. This isolation also allows for more intensive validation activities, such as the use of formal specification or even the formal proof of high integrity properties, to be used.

At the operation and maintenance stage any changes in the environment of the system or in the code itself must be subjected to additional hazard analysis to ensure that the environment changes and/or code changes don't result in unacceptable risk. Records should be kept of the entire development effort in order to build an experience base for the development of high integrity systems.

7. RELATIONSHIPS AMONG THE SESSION TOPICS

While each working group had a specific charter, there are strong relationships among them. Selection and description of techniques is perhaps the most fundamental task. The cost-benefit model will use data on techniques to develop a model in which techniques are classified according to the level of assurance they provide. Hazard analysis and criticality assessment help to determine the needed level of assurance. Risk management analyses will contribute not only to the criticality assessment but also to information regarding controlled practices. Knowledge of practices that should be encouraged or used in controlled environments will need information from data supplied in descriptions of techniques. A topic that lay in the background of workshop sessions is the capability and skills required for various application domains and levels of assurance. The development of guidance will be an iterative process requiring that all information coming from a variety of sources be integrated. Separate standards may be developed for topics shown in figure 4.

8. SUMMARY

NIST's first workshop on the assurance of high integrity software was successful in helping NIST to identify the topics that are most important to the development of federal standards and guidance in this area. There will necessarily be several standards, with one that points to standards addressing specific topics. Any framework for selection will be flexible to allow for differences in application domains and user environments.

NIST will continue its other activities that provide opportunities for information exchange. NIST serves as a co-sponsor of the COMPASS and National Computer Security conferences which address issues of systems security, software safety, and process integrity. NIST has also initiated a lecture series on high integrity systems at which eminent speakers present potential solutions that may be applied to assuring high integrity software. The purpose of these lectures, which are open to the public, is to increase public awareness of the need to consider high integrity as the goal for many kinds of software applications.

To help practitioners become aware of existing standards for high integrity software, NIST will prepare a bibliography of relevant standards and guidelines. NIST plans to strengthen its liaisons with national and international standards bodies, and to work toward adopting or adapting appropriate standards. NIST will coordinate an effort to produce an integrated body of guidance adapting or adopting existing standards as appropriate. Doing this will require identification of topics ready for standardization, and of areas where more research is needed. Workshop participants also recommended conduct of an experiment that implements a draft standard on a real project.

After high integrity software standards are developed, NVLAP-accredited laboratories will be considered as a cost-effective means of providing conformance evaluation.

To conduct these and other activities, NIST will seek funding and will encourage collaborative efforts with other agencies, industry, and academia.

REFERENCES

[1] Sennett, C.T., *High Integrity Software,* Plenum Press, London, 1989.

[2] "Guideline for Lifecycle Validation, Verification and Testing of Computer Software," FIPSPUB101, National Bureau of Standards, Gaithersburg, MD 20899, 1983.

[3] "Guideline for Software Verification and Validation Plans," FIPSPUB132, National Bureau of Standards, Gaithersburg, MD 20899, 1987.

[4] Wallace, Dolores R., and Roger U. Fujii, "Software Verification and Validation: Its Role in Computer Assurance and Its Relationship with Software Project Management Standards," NIST SP 500-165, National Institute of Standards and Technology, 1989.

[5] "Data Encryption Standard," FIPSPUB 46-1, National Bureau of Standards, Gaithersburg, MD 20899, 1988.

[6] "Guidelines for Security of Computer Applications," FIPSPUB73, National Bureau of Standards, Gaithersburg, MD 20899, June 1980.

[7] "Guideline for Computer Security Certification and Accreditation," FIPSPUB102, National Bureau of Standards, Gaithersburg, MD 20899, 1985.

[8] "Standard on Password Usage," FIPSPUB112, National Bureau of Standards, Gaithersburg, MD 20899, 1985.

[9] "Standard on Computer Data Authentication," FIPSPUB113, National Bureau of Standards, Gaithersburg, MD 20899, 1985.

[10] "Security Requirements for Cryptographic Modules," Draft FIPSPUB 140-1, National Institute of Standards and Technology, May 1990.

[11] U.S. DOD Trusted Computer System Evaluation Criteria, DOD 5200.28.STD, December 1985.

[12] MIL-STD-882B with change notice 1, Task 308, Software Safety Requirements Traceability Matrix and Analysis - DOD HBK-SWS, 20 April 1988, Software System Safety, Department of Defense.

[13] Leveson, N.G., "Software safety: Why, what and how," *ACM Computing Surveys* 18, 2 (June 1986), 25-69.

[14] Draft Interim Defence Standards 00-55 and 00-56, Requirements for the Procurement of Safety Critical Software in Defence Equipment, Requirements for the Analysis of Safety Critical Hazards, Ministry of Defence, Room 5150A, Kentigern House 85 Brown Street, Glasgow G2 8EX, May 1989.

[15] Quality Management and Quality Assurance Standards - Guidelines for Selection and Use, ISO 9000 (ANSI /ASQC Q90).

[16] Mills, H. D., M.Dyer, and R.C. Linger, "Cleanroom Software Engineering," *IEEE Software,* September 1987, pp.19-24.

[17] Kouchakdjian, A., V.R. Basili and S. Green, "Evaluation of the cleanroom methodology in the SEL," *Proceedings of the Fourteenth Annual Software Engineering Workshop,* Report SEL-89-007, November, 1989.

[18] Bartussek, W., and D.L. Parnas, "Using Traces to Write Abstract Specifications for Software Modules," Report TR 77-012, U. of North Carolina, Chapel Hill, NC, December 1977.

[19] McLean, J. "A Formal Method for the Abstract Specification of Software," *J. ACM* , Vol. 31, No. 3, pp. 600-627, July 1984.

[20] DOD-STD-2167A Military Standard Defense System Software Development, AMSC No. 4237, Department of Defense, February 29, 1988.

[21] "SafeIt, A framework for safety standards," ICSE Secretariat, Department of Trade & Industry, ITD7a - Room 840, Kingsgate House, 66/74 Victoria Street, London SW1E 6SW, England, June, 1990.

[22] Radio Technical Commission for Aeronautics, "Software Considerations in Airborne Systems and Equipment Certification," DO-178A, RTCA, Washington, DC, 1985.

[23] Anderson, T. and J.C. Knight, "Software Fault Tolerance In Real-Time Systems," *IEEE Transactions on Software Engineering,* 9(3), May, 1983.

[24] IEEE STD. 729 -1990, Revision: A Standard Glossary of Software Engineering Terminology, IEEE, 1990.

[25] ISO/IEC Standard 2832-20:1990, "Information Technology - Vocabulary System Development."

[26] Hecht, H., "Figure of Merit for Fault-Tolerant Space Computers," *IEEE Transactions on Computers,* Vol. C-22, No.3, March, 1973, pp. 246-251.

[27] Hecht, H., "Allocation of Resources for Software Reliability," *Proceedings COMPCON Fall '81,* IEEE, 1981.

[28] Hecht, Herbert, "Effectiveness Measures for Distributed Systems," *Symposium on Reliability of Distributed Software and Database Systems,* IEEE, 1981.

[29] Humphrey, W.S., et al., "A Method for Assessing the Software Engineering Capability of Contractors," CMU/SEI-87-TR-23, Software Engineering Institute, Pittsburgh,

PA, 1987.

[30] *Proceedings of the Annual Software Engineering Workshop,* Volumes 1-15, Goddard Space Flight Center, National Aeronautics and Space Administration, Greenbelt, MD 20771, 1976-1990.

[31] Basili, V.R. "Software Development: A Paradigm for the Future," *Proceedings, 13th Annual International Computer Software & Applications Conference (COMPSAC),* Orlando, FL, IEEE, September, 1989.

[32] Harel, D., "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming 8,* (1987), 231-274.

[33] "GOSIP: Government Open System Interconnection Profile," Federal Information Processing Standards Publication FIPSPUB 146, 1988.

[34] "POSIX: Portable Operating System Interface for Computer Environments," Federal Information Processing Standards Publication FIPSPUB 151-1, 1990.

[35] Crow, J.S., R. Lee, J.M. Rushby, F.W. von Henke, R.A. Whitehurst, "EHDM Verification Environment: An Overview," *Proceedings, 11th National Computer Security Conference,* National Security Agency, October 1988.

[36] Kemmerer, Richard A., "Integrating Formal Methods into the Development Process," *IEEE Software,* V 7, September 1990.

[37] Budkowski, S., and P. Dembinski, "An Introduction to Estelle: A Specification Language for Distributed Systems," *Computer Networks and ISDN Systems,* North Holland, V14, N3, 1987.

[38] Borgida, Alexander, "Features of Languages for the Development of Information Systems at the Conceptual Level," *IEEE Software,* Vol.2 No. 1, pp 63-72, July 1985.

[39] Marca, D.A. and C. L. McGowan, *SADT. Structure Analysis and Design Technique,* McGraw-Hill Co., 1987, ISBN 0-07-040235-3.

[40] Knight, J.C., and N.G. Leveson, "An Experimental Evaluation of the Assumption of Independence in Multi-version Programming," *IEEE Transactions on Software Engineering,* 12(1), January, 1986.

APPENDIX A. WORKSHOP PARTICIPANTS

TECHNIQUES

Susan Gerhart, MCC (Moderator)
Rick Kuhn, NIST (Recorder)
Richard Taylor, Atomic Energy Control Board of Canada
Cynthia Wright, MITRE
Martin Bailey, Naval Surface Warfare Center
Jo Atlee, University of Maryland
Rick Butler, NASA Langley Research Center
John Baumert, Computer Sciences Corp.
Bill Nieh, Logicon
Leo Beltracchi, Nuclear Regulatory Commission
Jon Luedeke, Battelle
Rob Ayers, ARINC
Ian Sutherland, Odyssey Research
John McLean, Naval Research Laboratory
Mark Ardis, Software Engineering Institute
Patty Trellue, Sandia Laboratory
J. Bret Michael, George Mason University
Madhu Rao, Westinghouse
Bill Majurski, NIST
Wayne McCoy, NIST
Scott Green, NASA Goddard Space Flight Center
David Rutherford, Rail Transportation Services

COST-BENEFIT

John Knight, University of Virginia (Moderator)
Dolores Wallace, NIST (Recorder)
Charles Baynard, Food and Drug Administration
Christopher Dabrowski, NIST
Dario DeAngelis, Logicon, Inc.
Herbert Hecht, SoHaR, Inc.
Linda Lauterbach, Research Triangle Institute
Ted Ralston, MCC
Philip Sedgwick, Control Systems Analysis

CONTROLLED AND ENCOURAGED PRACTICES

Arch McKinlay, McDonnell Aircraft Company (Moderator)
Neva Carlson, NIST (Recorder)
Rob Ayers, ARINC
Robert Bagwill, NIST
Protagoras Cutchis, JHU/APL
John Dirks, CSC
Bill Ghrist, Westinghouse Electric
Henry Heffernan, EDPNS
Dan Juttelstad, NUSC
Vince Streckler, Logicon
Laura Strigel, NIST
Gordon Symonds, BRMD (Canada)

HAZARD ANALYSIS AND CRITICALITY ASSESSMENT

Michael Brown, Naval Surface Warfare Center (Moderator)
John C. Cherniavsky, National Science Foundation (Recorder)
Betty Chao, Sandia National Laboratories
Janet Dunham, Research Triangle Institute
Dave Ferraiolo, NIST
Frank Houston, Food and Drug Administration
Grady Lee, Vitro, Incorporated
Bob Pierce, Air Force CSC/SRM
Heinz Rosen, U. L. Inc.
Mel Smyre, FMC Corporation

APPENDIX B. MODERATOR PRESENTATIONS

Working Group Charter

TECHNIQUES for assuring and demonstrating high integrity

Susan L. Gerhart

(Manager, MCC STP Formal Methods Project)

NIST Workshop on
Assurance of High-integrity Software

22-23 January 1991

Assumptions

- Standards and frameworks define objectives and processes
  - 0055, SafeIT, etc.
  - Orange Book, etc.
  - Experience in certification
- US industrial practice and educational system is the target
  - High-tech CASE thru low-tech C
  - Differences among industry sectors (medical, avionics)
  - Common (reviews) thru specialized (security verification) practices
  - Few SE schools, many CS programs, overlap other engineering
- International research perspective should be maintained
  - Lessons from UK for FM and FT, Esprit for PCTE, etc.
  - Technical aspects (tools, theory) are international but US-biased
  - Collaboration is possible, beneficial, on-going

Needs

- Recommended techniques for high-integrity
  - Now -- what is usable and credible?
  - Future -- what changes needed for more usability and credibility?
- Technique "catalog" could be the product
  - Named in various standards and frameworks
  - Common and newer in industrial practice and education
  - Reflect (international) research trends
- Provide a format for description and assessment
  - Basis for technique assessment criteria
- This group "sets up" more complete and detailed technique analysis
  - Technical description and assessment (our charter)
  - Cost-benefits (another group) -- identify factors for them
  - Hazard analysis (another group) -- other techniques, find interface
  - Forbidden practices (another group) -- correlate with techniques
  - Future -- NIST coordination, more detailed analysis of techniques

Tasks

- General input (5 minute max): "where are you coming from?"
  - Background factors -- standards, practice,...
  - Statement of interests -- research, marketplace, national needs,...
- Technique catalog and assessment criteria
  - Pooled experience of group members -- as advocates and critics
  - Techniques for consideration
  - Important criteria
  - Workable format for TECHNICAL aspects
  - Illustrative descriptions (5-10)

Enumeration -- proposal

- "Brand names"
  - Individuals (Jackson, Dijkstra, Leveson)
  - Schools (VDM, Cleanroom, semantic analysis, mutation testing)
  - Generic (refinement, state machine, n-version)
- Current practice per industry sector
  - Medical, Avionics, Security,...
  - Widely taught in CS and engineering programs
- Named in standards
  - See BSI 89/33006

Mode: free associate without filtering for high integrity or practicality

Description -- proposal

- How it works
- What it achieves wrt high integrity
- Current applicability (practice, education, experience)
- References

How it works

- Representations used
  - Text, graphics, etc.
  - Executable, unsalable
- Steps performed
  - Mechanics --
    "transform this to that"
  - Synthesis and analysis steps
  - Tools used
  - Roles involved
- Artifacts produced
  - Documents
  - Data
  - Representations
- Roles involved
  - Person to task mapping
    Example: specifier,
    verification system operator
  - Skills required

What it achieves wrt high-integrity

Positive
- Errors identified
- Evaluation data produced
- Reuse possibilities

Negative
- Fallibility --
  common failures, gaps in knowledge,...
- Bottlenecks --
  sequential steps, limited resources, skills,...
- Technical barriers

Other techniques
- Required
- Supported

Assessment -- proposal

- Current applicability of a technique
  - Where is it being used? How?
  - Where is it taught?
  - Who is researching it?
  - Why are they doing this?
  - If not in use but has potential, then what changes are needed?
- Recommendations to NIST
  - General -- an approach to identifying and assessing techniques
  - Several specific techniques' evaluations

  >> Eliminate from high integrity consideration
  >> Develop more detailed description
  >> Perform cost-benefit analysis (wrt what?)

Schedule

1. Afternoon-1
   - Introductions, position statements
   - Planning
2. Afternoon-2
   - Identification of techniques
   - Description format
3. Morning-1
   - Specific techniques
4. Morning-2
   - Comparison
   - Evaluation of approach
5. Afternoon-3
   - Summary for plenary

Issues

1. Are detailed techniques really needed? How much detail? How will they be used?
2. Is integration with other groups well enough defined
- What biases does our group have?
- Missing -- CASE vendors?
- Too much formal methods?
3. Oversell or undersell tendencies?
4. System (vs. software) considerations -- how to include?

COST BENEFIT ANALYSIS OF ASSURANCE TECHNIQUES

John C. Knight

Department of Computer Science
University of Virginia

DEPENDABLE SOFTWARE

- Safety-Critical Systems:

  A system where the consequences of failure are extremely serious.

- Examples Include:
  - Avionics Systems
  - Nuclear Power System Control
  - Medical Devices

- Safety-Critical Systems Are Required To Be Dependable:

  Dependability is the property of a computing system that allows reliance to be justifiably placed on the service that it delivers (Laprie).

- Dependable vs. High Integrity
- Dependability Requirement May Be Mandated
- "As Good As We Can Make It" Might Not Be Good Enough

UVA
Department of Computer Science

THE PROBLEM

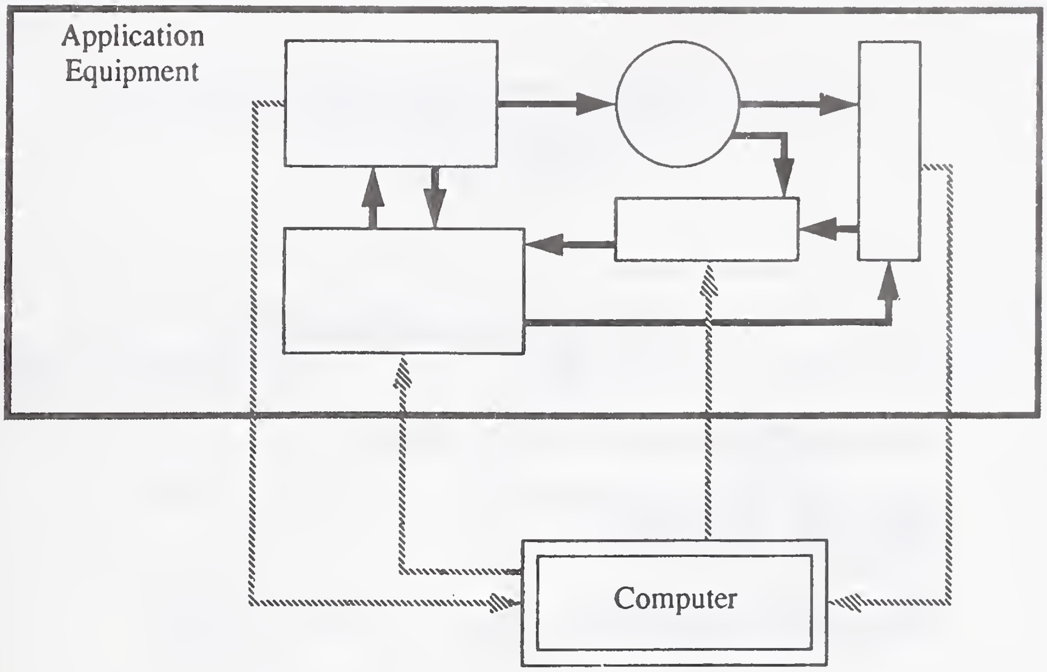
- Is It Worth Computerizing A Given Application?
- Prescribed Dependability Requirements:
  - FAA, NRC, FDA, DoD, NIST
- Development Cost Has To Be Compared With:
  - Operational Benefit
  - Cost Of Operational Failure
  - Prescribed Dependability Levels
- But, Development Costs Change With Time:
  - Better Algorithms, E.g., Theorem Provers
  - Better Equipment, E.g., Much Faster Computers
  - Better Software
  - Different Perceived Or Actual Worth

ASPECTS OF DEPENDABILITY

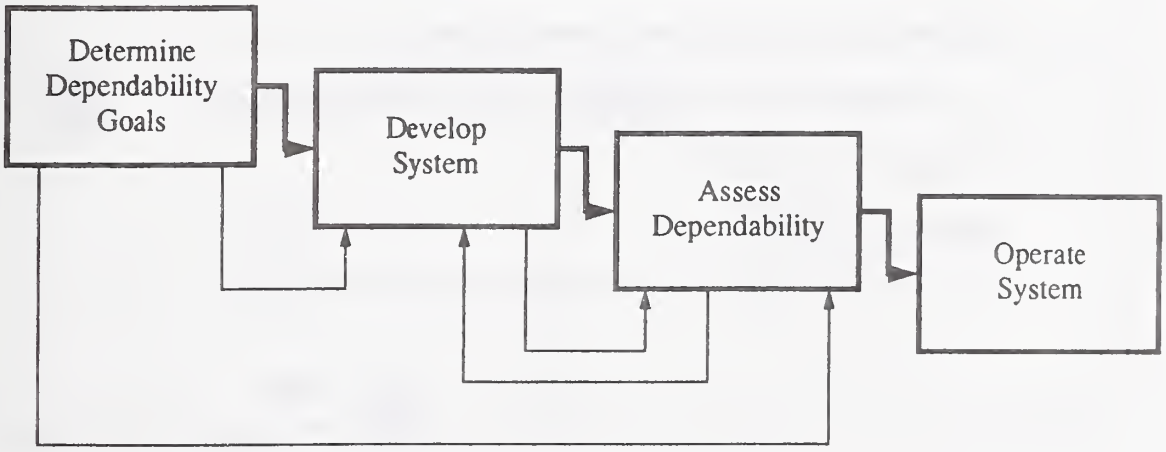
- Reliability:
  - Probability That The System Operates Correctly Up To Time t
- Availability:
  - Probability That System Operates Correctly At Time t
- Software Safety:
  - Property That The Software Does Not Cause A Safety Failure
  - There Are Systems Where No Action Is Preferable To Wrong Action
- System Safety:
  - Property That The System Does Not Suffer A Safety Failure
- A System Might Be Very Safe But Have Low Availability
- A System Might Be Highly Available But Have Low Reliability
- These Are *Fundamentally* Different Concepts

UVA
Department of Computer Science

SOFTWARE SAFETY vs SYSTEM SAFETY



SOURCES OF COST



- Can Dependability Always Be Assessed?
- Can Dependability For The Systems Of Interest *Ever* Be Assessed?
- What Are The Cost Tradeoff's?

UVA
Department of Computer Science

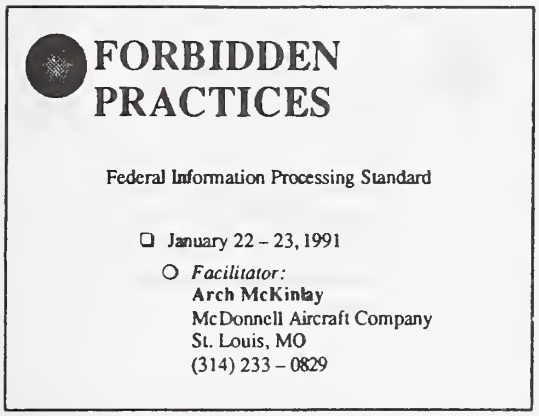
PROPOSED APPROACH

- Define Applications Domains To Be Covered:
  - All ''High Integrity'' Systems?
  - Medical?
  - Commercial Transportation Systems?
- Determine Dependability Requirements For Each Domain:
  - Safety, Reliability, Availability Or What?
- Hypothesize Appropriate Development Processes:
  - Some System Properties Can Be Achieved In Many Ways
- Attempt To Quantify Costs
- Determine Feasibility:

Are There Classes Of Systems That Are Beyond Our Ability To Create?

UVA
Department of Computer Science

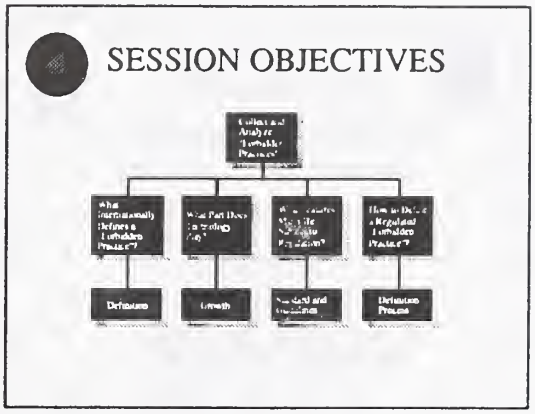
NIST FIPS "FORBIDDEN PRACTICES" SESSION
GOING-IN PROPOSALS
January 1991



FORBIDDEN PRACTICES

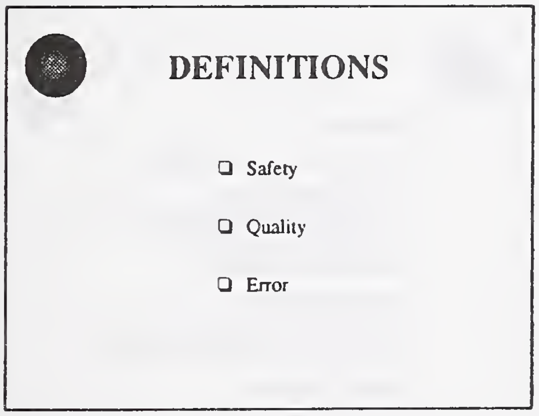
Federal Information Processing Standard

- ❑ January 22 – 23, 1991
  - ○ *Facilitator:*
    Arch McKinlay
    McDonnell Aircraft Company
    St. Louis, MO
    (314) 233 – 0829



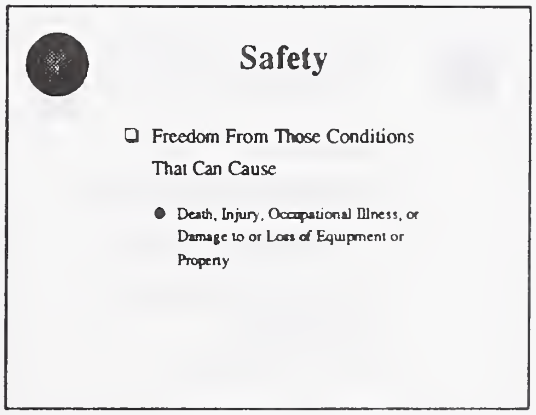
SESSION OBJECTIVES

Collect and Analyze "Forbidden Practices"
What Internationally Defines a Forbidden Practice?
What Part Does Technology Play?
[illegible] Regulation?
How to Define a Regulated Forbidden Practice?
Definitions
Growth
Standard and Guidelines
Definition Process



DEFINITIONS

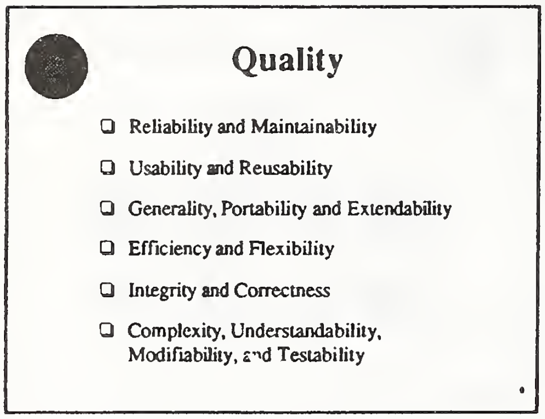
- ❑ Safety
- ❑ Quality
- ❑ Error



Safety

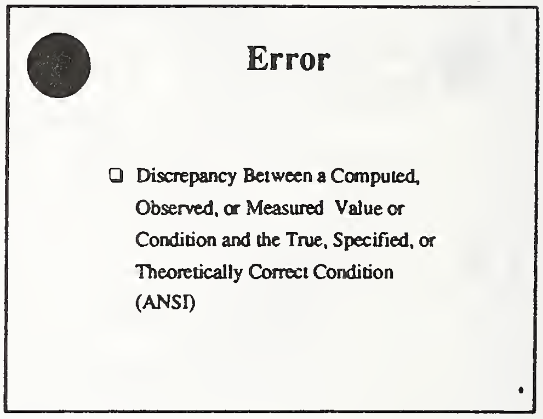
- ❑ Freedom From Those Conditions That Can Cause
  - ● Death, Injury, Occupational Illness, or Damage to or Loss of Equipment or Property

NIST FIPS "FORBIDDEN PRACTICES" SESSION
GOING–IN PROPOSALS
January 1991



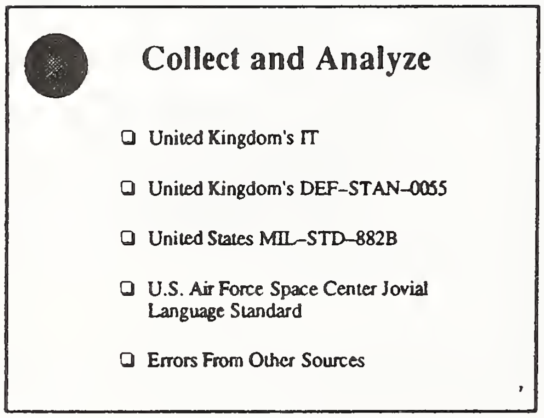
Quality

- Reliability and Maintainability
- Usability and Reusability
- Generality, Portability and Extendability
- Efficiency and Flexibility
- Integrity and Correctness
- Complexity, Understandability, Modifiability, and Testability



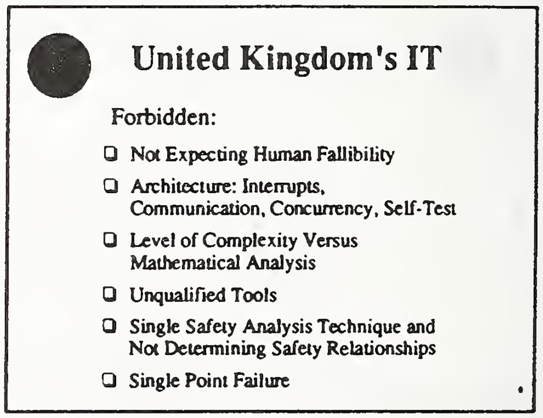
Error

- Discrepancy Between a Computed, Observed, or Measured Value or Condition and the True, Specified, or Theoretically Correct Condition (ANSI)



Collect and Analyze

- United Kingdom's IT
- United Kingdom's DEF–STAN–0055
- United States MIL–STD–882B
- U.S. Air Force Space Center Jovial Language Standard
- Errors From Other Sources

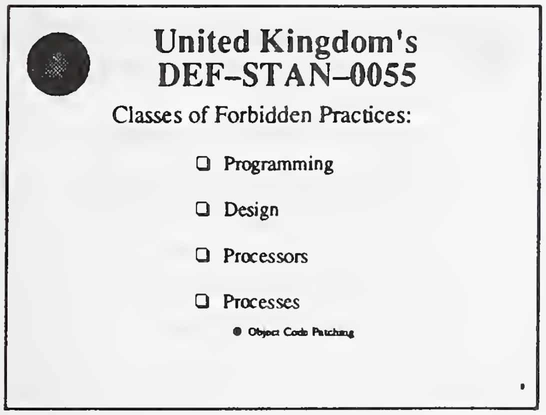


United Kingdom's IT

Forbidden:

- Not Expecting Human Fallibility
- Architecture: Interrupts, Communication, Concurrency, Self-Test
- Level of Complexity Versus Mathematical Analysis
- Unqualified Tools
- Single Safety Analysis Technique and Not Determining Safety Relationships
- Single Point Failure

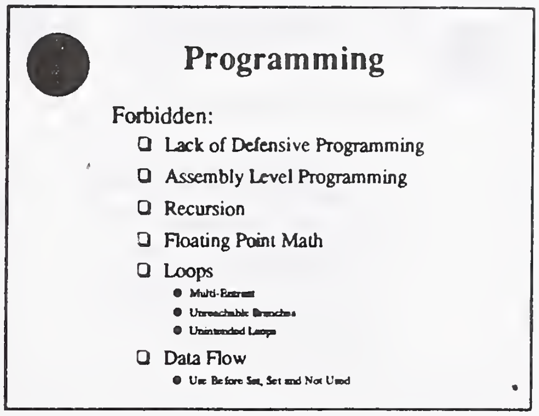
NIST FIPS "FORBIDDEN PRACTICES" SESSION
GOING-IN PROPOSALS
January 1991



United Kingdom's DEF-STAN-0055

Classes of Forbidden Practices:

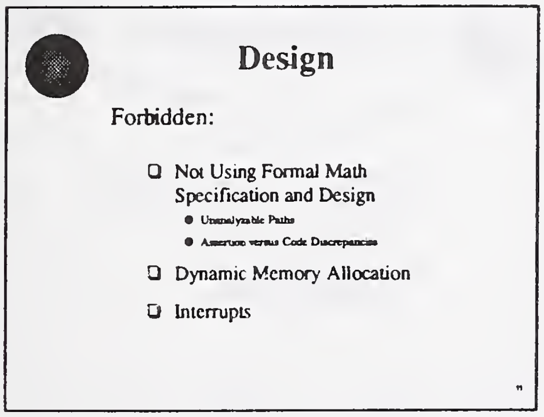
- ❑ Programming
- ❑ Design
- ❑ Processors
- ❑ Processes
  - ● Object Code Patching



Programming

Forbidden:

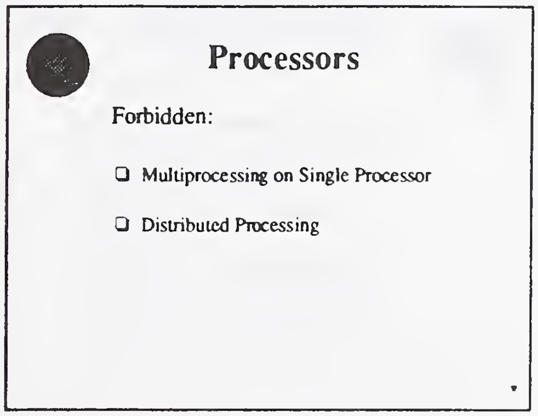
- ❑ Lack of Defensive Programming
- ❑ Assembly Level Programming
- ❑ Recursion
- ❑ Floating Point Math
- ❑ Loops
  - ● Multi-Entrant
  - ● Unreachable Branches
  - ● Unintended Loops
- ❑ Data Flow
  - ● Use Before Set, Set and Not Used



Design

Forbidden:

- ❑ Not Using Formal Math Specification and Design
  - ● Unanalyzable Paths
  - ● Assertion versus Code Discrepancies
- ❑ Dynamic Memory Allocation
- ❑ Interrupts

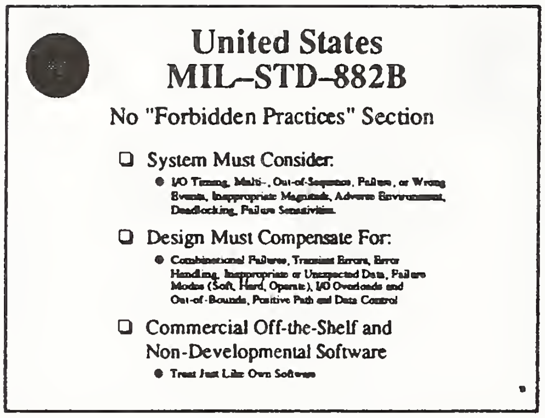


Processors

Forbidden:

- ❑ Multiprocessing on Single Processor
- ❑ Distributed Processing

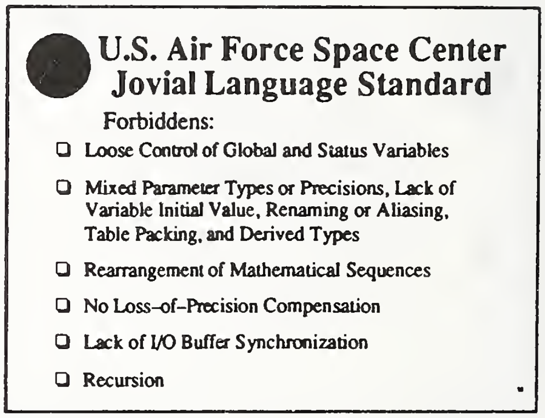
NIST FIPS "FORBIDDEN PRACTICES" SESSION
GOING-IN PROPOSALS
January 1991



United States MIL–STD–882B

No "Forbidden Practices" Section

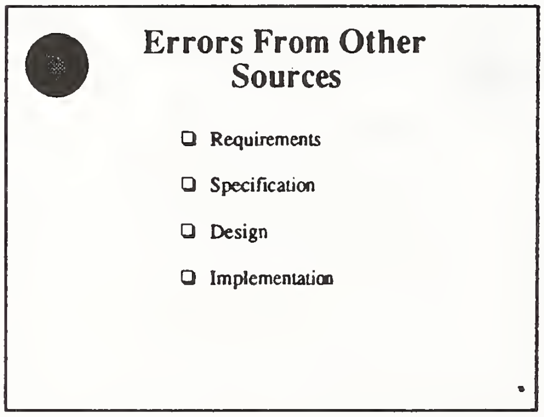
- ❑ System Must Consider:
  - I/O Timing, Multi-, Out-of-Sequence, Failure, or Wrong Events, Inappropriate Magnitude, Adverse Environment, Deadlocking, Failure Sensitivities.
- ❑ Design Must Compensate For:
  - Combinational Failures, Transient Errors, Error Handling, Inappropriate or Unexpected Data, Failure Modes (Soft, Hard, Operate), I/O Overloads and Out-of-Bounds, Positive Path and Data Control
- ❑ Commercial Off-the-Shelf and Non-Developmental Software
  - Treat Just Like Own Software



U.S. Air Force Space Center Jovial Language Standard

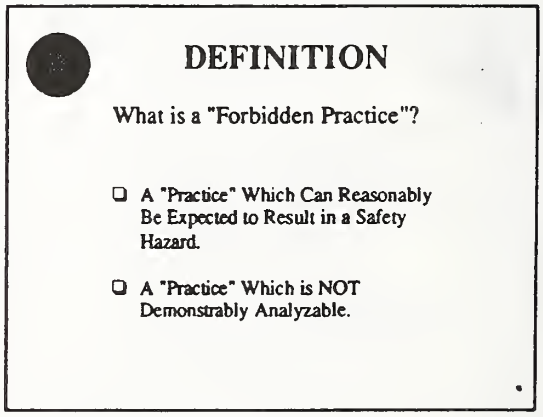
Forbiddens:

- ❑ Loose Control of Global and Status Variables
- ❑ Mixed Parameter Types or Precisions, Lack of Variable Initial Value, Renaming or Aliasing, Table Packing, and Derived Types
- ❑ Rearrangement of Mathematical Sequences
- ❑ No Loss–of–Precision Compensation
- ❑ Lack of I/O Buffer Synchronization
- ❑ Recursion



Errors From Other Sources

- ❑ Requirements
- ❑ Specification
- ❑ Design
- ❑ Implementation

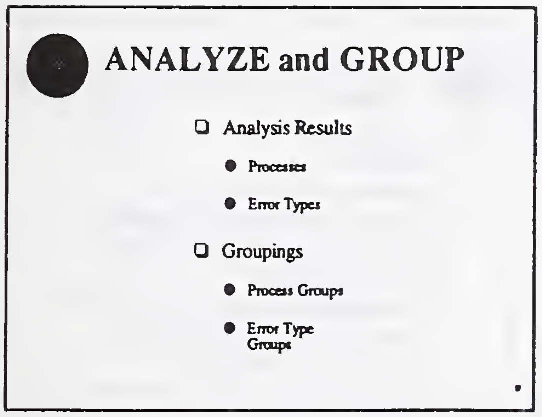


DEFINITION

What is a "Forbidden Practice"?

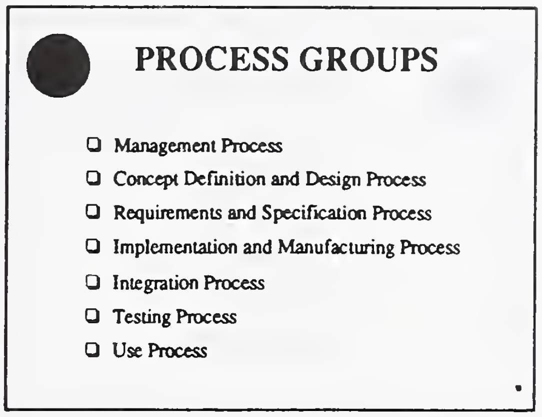
- ❑ A "Practice" Which Can Reasonably Be Expected to Result in a Safety Hazard.
- ❑ A "Practice" Which is NOT Demonstrably Analyzable.

NIST FIPS "FORBIDDEN PRACTICES" SESSION
GOING–IN PROPOSALS
January 1991



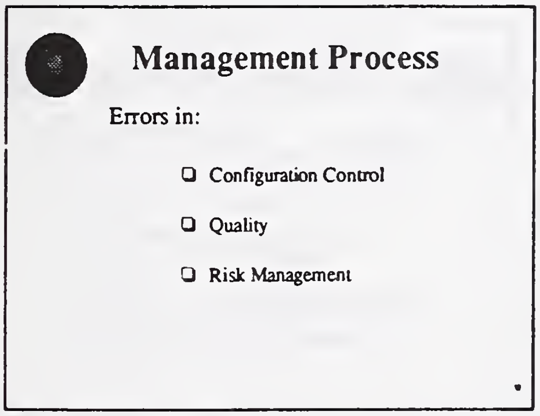
ANALYZE and GROUP

- ❑ Analysis Results
  - Processes
  - Error Types
- ❑ Groupings
  - Process Groups
  - Error Type Groups



PROCESS GROUPS

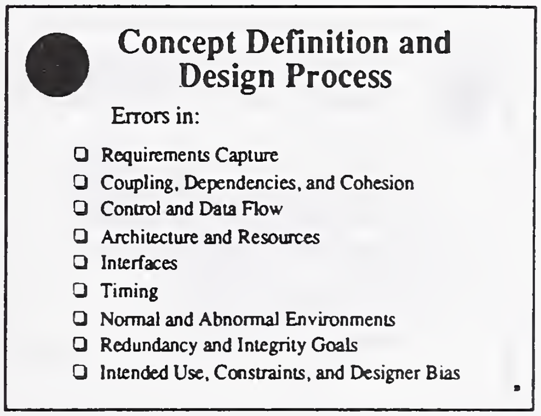
- ❑ Management Process
- ❑ Concept Definition and Design Process
- ❑ Requirements and Specification Process
- ❑ Implementation and Manufacturing Process
- ❑ Integration Process
- ❑ Testing Process
- ❑ Use Process



Management Process

Errors in:

- ❑ Configuration Control
- ❑ Quality
- ❑ Risk Management

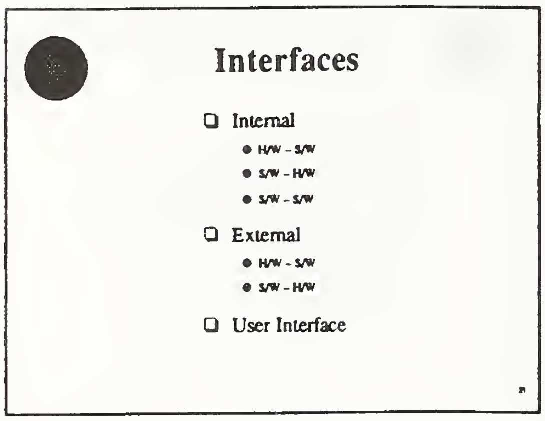


Concept Definition and Design Process

Errors in:

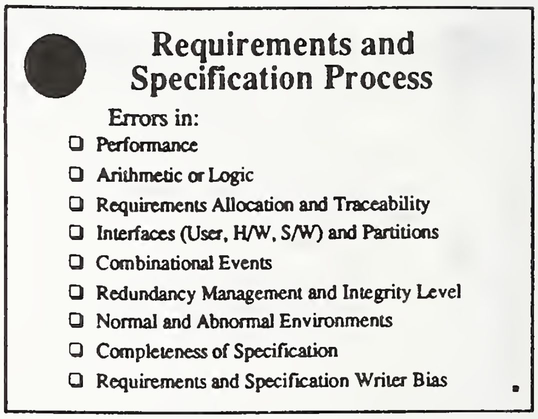
- ❑ Requirements Capture
- ❑ Coupling, Dependencies, and Cohesion
- ❑ Control and Data Flow
- ❑ Architecture and Resources
- ❑ Interfaces
- ❑ Timing
- ❑ Normal and Abnormal Environments
- ❑ Redundancy and Integrity Goals
- ❑ Intended Use, Constraints, and Designer Bias

NIST FIPS "FORBIDDEN PRACTICES" SESSION
GOING-IN PROPOSALS
January 1991



Interfaces

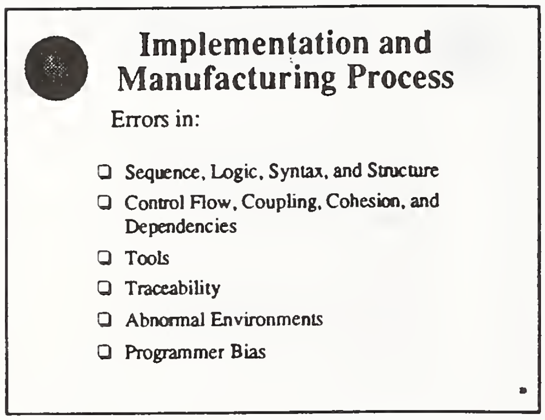
- ❑ Internal
  - H/W - S/W
  - S/W - H/W
  - S/W - S/W
- ❑ External
  - H/W - S/W
  - S/W - H/W
- ❑ User Interface



Requirements and Specification Process

Errors in:

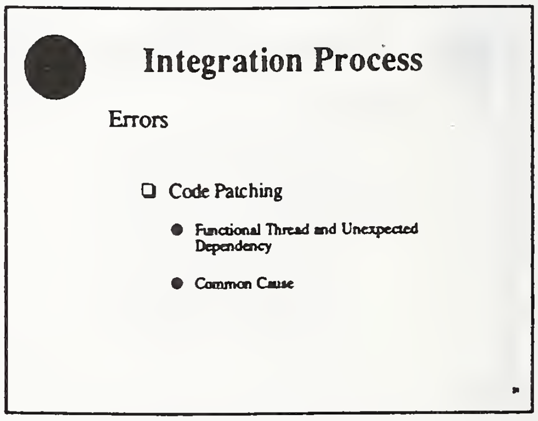
- ❑ Performance
- ❑ Arithmetic or Logic
- ❑ Requirements Allocation and Traceability
- ❑ Interfaces (User, H/W, S/W) and Partitions
- ❑ Combinational Events
- ❑ Redundancy Management and Integrity Level
- ❑ Normal and Abnormal Environments
- ❑ Completeness of Specification
- ❑ Requirements and Specification Writer Bias



Implementation and Manufacturing Process

Errors in:

- ❑ Sequence, Logic, Syntax, and Structure
- ❑ Control Flow, Coupling, Cohesion, and Dependencies
- ❑ Tools
- ❑ Traceability
- ❑ Abnormal Environments
- ❑ Programmer Bias

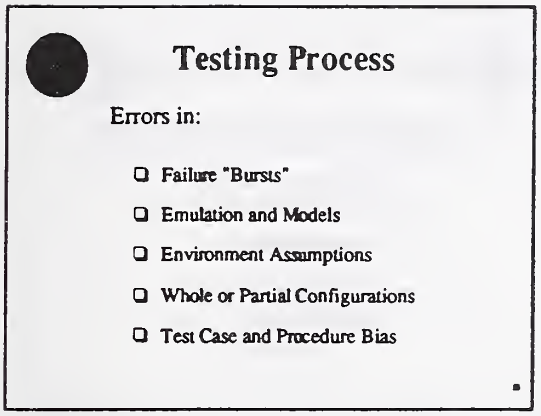


Integration Process

Errors

- ❑ Code Patching
  - Functional Thread and Unexpected Dependency
  - Common Cause

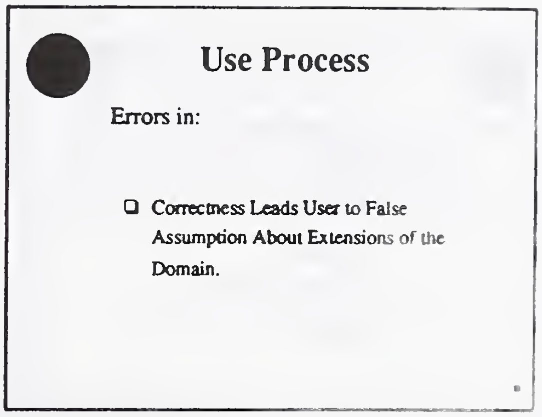
NIST FIPS "FORBIDDEN PRACTICES" SESSION
GOING-IN PROPOSALS
January 1991



Testing Process

Errors in:

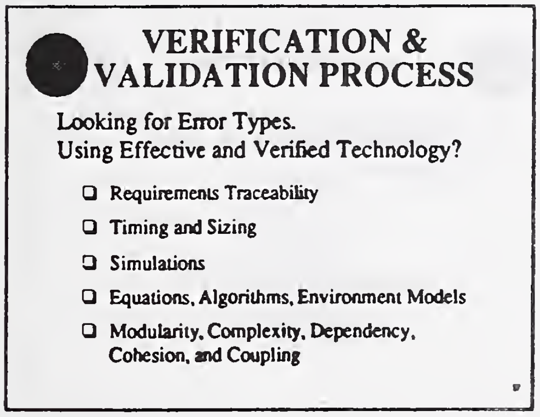
- Failure "Bursts"
- Emulation and Models
- Environment Assumptions
- Whole or Partial Configurations
- Test Case and Procedure Bias



Use Process

Errors in:

- Correctness Leads User to False Assumption About Extensions of the Domain.

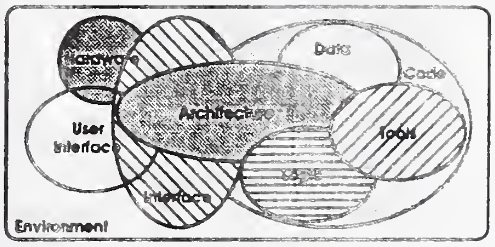


VERIFICATION & VALIDATION PROCESS

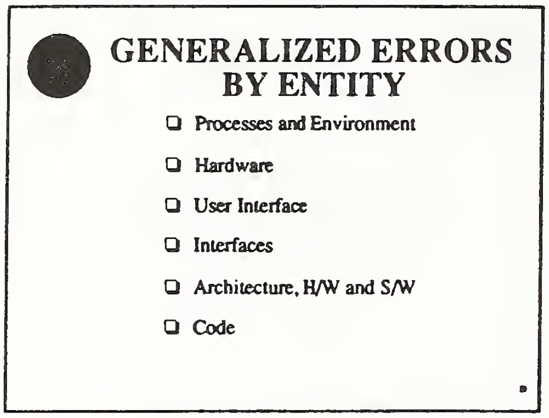
Looking for Error Types.
Using Effective and Verified Technology?

- Requirements Traceability
- Timing and Sizing
- Simulations
- Equations, Algorithms, Environment Models
- Modularity, Complexity, Dependency, Cohesion, and Coupling

ERROR TYPE GROUPS

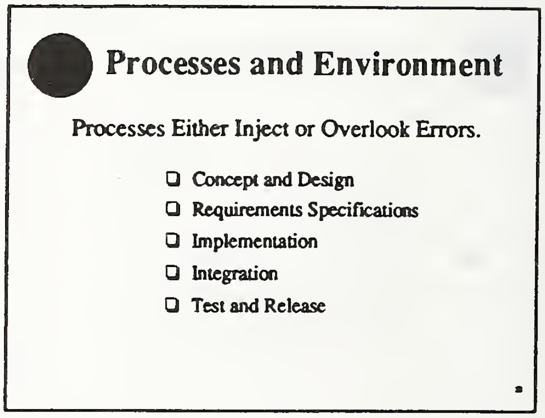


NIST FIPS "FORBIDDEN PRACTICES" SESSION
GOING-IN PROPOSALS
January 1991



GENERALIZED ERRORS BY ENTITY

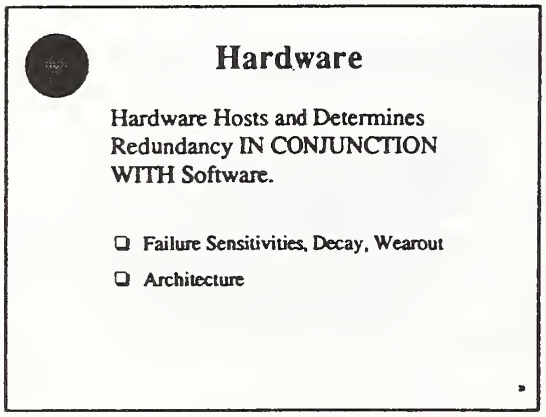
- ❑ Processes and Environment
- ❑ Hardware
- ❑ User Interface
- ❑ Interfaces
- ❑ Architecture, H/W and S/W
- ❑ Code



Processes and Environment

Processes Either Inject or Overlook Errors.

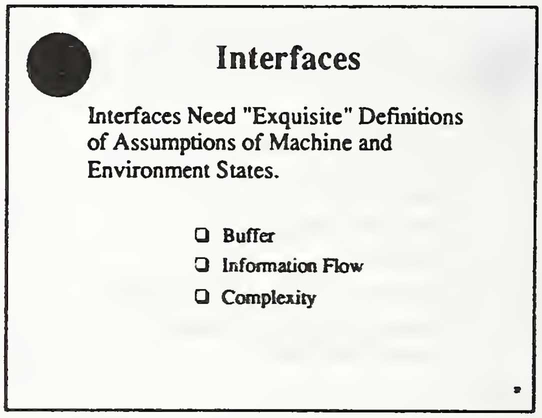
- ❑ Concept and Design
- ❑ Requirements Specifications
- ❑ Implementation
- ❑ Integration
- ❑ Test and Release



Hardware

Hardware Hosts and Determines Redundancy IN CONJUNCTION WITH Software.

- ❑ Failure Sensitivities, Decay, Wearout
- ❑ Architecture

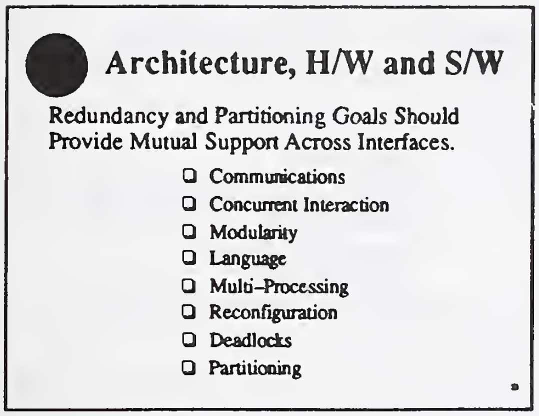


Interfaces

Interfaces Need "Exquisite" Definitions of Assumptions of Machine and Environment States.

- ❑ Buffer
- ❑ Information Flow
- ❑ Complexity

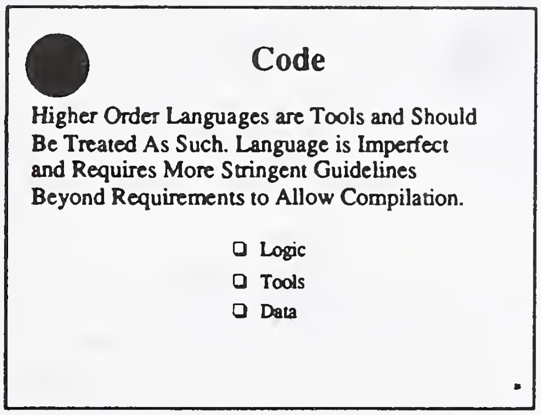
NIST FIPS "FORBIDDEN PRACTICES" SESSION
GOING-IN PROPOSALS
January 1991



Architecture, H/W and S/W

Redundancy and Partitioning Goals Should Provide Mutual Support Across Interfaces.

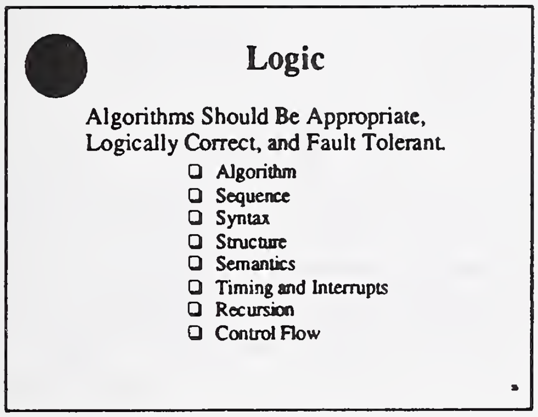
- ❑ Communications
- ❑ Concurrent Interaction
- ❑ Modularity
- ❑ Language
- ❑ Multi–Processing
- ❑ Reconfiguration
- ❑ Deadlocks
- ❑ Partitioning



Code

Higher Order Languages are Tools and Should Be Treated As Such. Language is Imperfect and Requires More Stringent Guidelines Beyond Requirements to Allow Compilation.

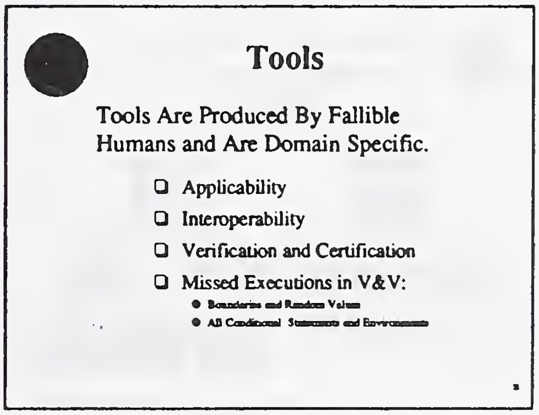
- ❑ Logic
- ❑ Tools
- ❑ Data



Logic

Algorithms Should Be Appropriate, Logically Correct, and Fault Tolerant.

- ❑ Algorithm
- ❑ Sequence
- ❑ Syntax
- ❑ Structure
- ❑ Semantics
- ❑ Timing and Interrupts
- ❑ Recursion
- ❑ Control Flow

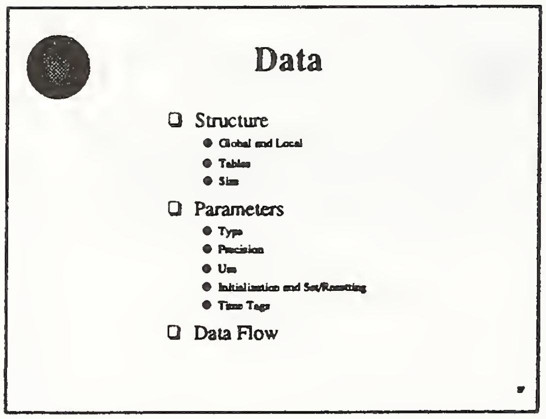


Tools

Tools Are Produced By Fallible Humans and Are Domain Specific.

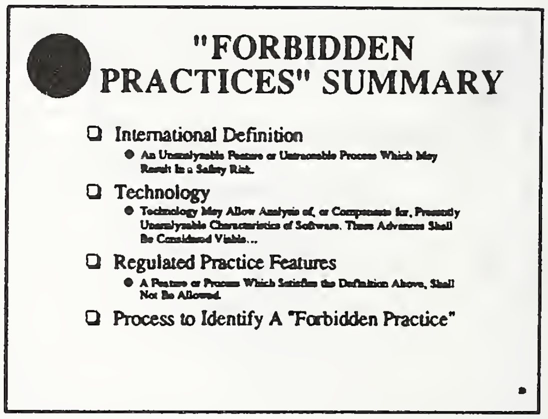
- ❑ Applicability
- ❑ Interoperability
- ❑ Verification and Certification
- ❑ Missed Executions in V&V:
  - ● Boundaries and Random Values
  - ● All Conditional Statements and Environments

NIST FIPS "FORBIDDEN PRACTICES" SESSION
GOING–IN PROPOSALS
January 1991



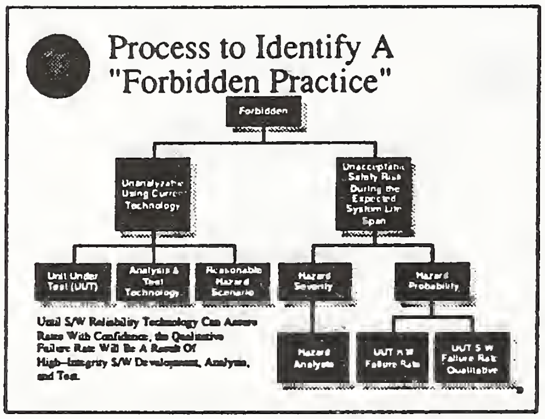
Data

- Structure
  - Global and Local
  - Tables
  - Size
- Parameters
  - Type
  - Precision
  - Use
  - Initialization and Set/Resetting
  - Time Tags
- Data Flow



"FORBIDDEN PRACTICES" SUMMARY

- International Definition
  - An Unanalyzable Feature or Untraceable Process Which May Result In a Safety Risk.
- Technology
  - Technology May Allow Analysis of, or Compensate for, Presently Unanalyzable Characteristics of Software. These Advances Shall Be Considered Viable...
- Regulated Practice Features
  - A Feature or Process Which Satisfies the Definition Above, Shall Not Be Allowed.
- Process to Identify A "Forbidden Practice"



Process to Identify A "Forbidden Practice"

Forbidden
Unanalyzable Using Current Technology
Unacceptable Safety Risk During the Expected System Life Span
Unit Under Test (UUT)
Analysis & Test Technology
Reasonable Hazard Scenario
Hazard Severity
Hazard Probability
Hazard Analysis
UUT H/W Failure Rate
UUT S/W Failure Rate Qualitative

Until S/W Reliability Technology Can Assure Rates With Confidence, the Qualitative Failure Rate Will Be A Result Of High–Integrity S/W Development, Analysis, and Test.

HAZARD ANALYSIS SESSION

Michael Brown
Naval Surface Warfare Center, Dahlgren, Virginia

Few advances in technology have impacted our lives as profoundly as the computer has: virtually every aspect is affected in one way or another. Devices from toasters and coffee makers to wide body jets use computers to control their operation. In recent years, computers have replaced control systems in which a failure can result in a disaster. In some applications, computer software makes decisions that are safety or security critical, whose errors could possibly endanger hundreds or thousands of lives. In others, the operation of the computer is essential to the successful completion of the system's mission. Theme types of systems may be classified in one or more categories including Safety Critical, Security Critical, Mission Critical, or any of several others. These terms all imply that the system must provide a high degree of assurance that it will operate in its intended environment without producing an unacceptable level of risk. We can group all of these systems together under the heading of High Integrity systems. Unfortunately, the software engineering technology available today is not able to provide the required levels of assurance with any degree of confidence.

I have a recurring vision of a disaster occurring within the U.S. Military community that will have the same impact on the Department of Defense (DOD) as the Challenger had on the National Aeronautics and Space Administration (NASA). When the possibility of the loss of a shuttle was discussed, few envisioned that it would virtually put NASA out of business for over 2 years. I refer to this disaster as the USA Challenger. As with all mishaps, there will be a Joint Adjunct General's, or JAG, investigation. However, instead of the usual attempt to place blame on an individual, the investigation will reveal that the disaster was the result of an error in the computing system. Further investigation will find that an error in the software resulted in the disaster. Unfortunately, the desire to have a "scapegoat" will surface and the investigation will turn to identifying the guilty party. Who will receive the blame? The systems engineer whose responsibility it is to develop the system and ensure the proper integration of the hardware and software? The Software Engineer who designed and developed the software? The Programmer who translated the software design requirements into code? The System Safety Engineer whose responsibility it was to identify, analyze and make recommendations to mitigate the hazards? Or will the blame fall on the safety review authority who certified the system for fleet use?

High integrity systems have been a major concern of the DOD for many years. The safety of weapon systems is critical to their effective deployment. With the increasing applications of software to weapon systems, and the increasing authority being given the software, Software Systems Safety has evolved into a recognized discipline within System Safety Engineering to provide a high degree of assurance that the software control of these systems will be safe. Similarly, a high degree of assurance must be maintained for computers that store or manipulate classified data.

NASA also has many applications in which the integrity of the software is essential to the successful execution of a system's mission. Systems such as interplanetary satellites, require that the software execute for years without any flaws that may result in the loss of the satellite or communications with that satellite.

NASA and the military are not the only government agencies that face potential disasters related to software. The Food and Drug Administration (FDA) and the Red Cross luckily found an error in the national blood supply databank that almost resulted in the distribution of thousands of pints of AIDS-tainted blood. Numerous other medical devices also contain software that could have or has had disastrous results due to failures. The Therac 25 Radiation Therapy machine is a good example. The Federal Aviation Adminstration (FAA) must rely daily on thousands of computer programs to keep our air traffic control system functioning safely and efficiently. Communications companies such as AT&T must rely heavily on computer systems to monitor and control their communication networks. Failure of such systems can result in a partial or complete lockout of the communications network, costing the company thousands or millions of dollars in lost revenues, serious damage to the company's reputation, and possibly even endanger lives or property. In other cases, the control of nuclear power plants is gradually switching to software.

The safety, security and availability issues associated with these systems are significant. Yet we find that we lack many of the tools and techniques necessary to effectively analyze and test these complex systems. Yet, even as we continue to tackle these problems daily, technological advances will soon present an entirely new set of issues. Multi-process or chips containing hundreds of microprocessors executing programs in true parallel fashion, that consist of billions of lines of code, not the millions of lines of code that we must deal with today. Execution speeds will soon be in the billions of instructions per second, with the programs stored on chips containing gigabytes of memory. Soon, practical applications of neural networks will be presenting new challenges for those trying to assure the integrity of the host systems.

The DOD has developed the 300 series tasks to MIL-STD-882B, System Safety Program Requirements, to address the integrity of safety critical applications of software [1]. Although the methodology behind these tasks is generally accepted, the tasks themselves are not. This stems partly from the level of effort required to perform the tasks and from a lack of understanding of the intent behind the tasks. There are also some issues surrounding the intrusion into another discipline: Safety engineering versus software engineering versus systems engineering. In addition, MIL-STD-082B describes what to do but does not describe how to do it.

The National Computer Security Center handbook, commonly referred to as the "Orange Book", provides evaluation criteria for trusted computer systems for security applications [2]. Classification levels are assigned according to the degree of protection provided to various levels of classified data. However, as the title implies, it is a set of evaluation criteria, not a how-to book or a description of what to do. These documents, along with several others including the British DEF-STAN-00 55 and 00 56 will be examined by this session as a possible basis to accomplish our objectives for the Hazard identification and criticality assessment session [3,4].

However, the issues that must be addressed for high integrity systems must extend beyond the issues of safety and security. It also must encompass any aspect of the system operation in which a failure could result in an unacceptable risk, whether personal, financial, or otherwise. This session will attempt to address the issue of what a Hazard is in terms of reliability, safety, availability, process security, and other high integrity system requirements. How do we define the term "hazard"? Once we have identified a hazard, how do we preclude its occurrence or reduce its associated risk? How do we determine its probability of occurrence? All of these issues will be addressed by the session.

The objectives of this session will be to:

o Identify techniques for identifying and classifying critical systems and components.

o Identify analysis methodologies to identify hazards

o Identify assessment program requirements
  - How much analysis is required
  - How much analysis is enough
  - How much additional testing is required
  - Scope the portion of the system requiring analysis
  - Identify systems that require a high degree of integrity.

Last week, I had the opportunity to attend the Inaugural Convocation of the Irvine Research Unit in Software. A portion of that Research Unit will be a Software Safety Research Center. I was asked to provide a government perspective on the Research Center and the needs of the Software Safety community in general. In discussing the government needs, I pointed out the vast differences in each agency's needs. Regulatory agencies need a methodology by which they can assess the safety or security or availability of these high integrity systems. "Developing" agencies also need analysis tools, techniques, and methodologies for analyzing and testing these systems. Certification agencies need risk assessment methodologies.

I also noted that there are several caveats that must apply to the establishment of the Software Safety Research Center. First, a full systems engineering approach must be taken to ensure that the software is analyzed and tested in a system perspective and that tools, techniques and methodologies are designed to support this approach. Next, analysis techniques must be cost effective or they will not be used. Tools must be user-friendly and transportable. Expectations of the users and testers must be realistic, Finally, there must be full industry and government participation in the research efforts. Many of these same caveats apply to the work being undertaken by this workshop.

The Hazard Identification and Criticality Assessment session will begin with introductions and brief position statements by the participants. We will attempt to define a hazard for safety, information security, process security, dependability, availability, and integrity. Next we will examine the classification of hazards and attempt to determine what is required for a criticality assessment. We will address what is required to perform a risk assessment. Finally, we will summarize our efforts for the plenary session.

APPENDIX C. Draft Templates of Techniques

The Techniques working group developed a draft template which can be used to describe techniques for assuring high integrity software. The template requires a considerable amount of information about a technique. The working group tried to describe seven techniques with the template. The group recognizes that more information is needed to fully describe most of the seven techniques and that some of the information included here may not be accurate. The descriptions are provided here for two reasons. One is to make people aware of the existence of these techniques. The second is to test how well the template will work for describing techniques. Comments on both the template and the information about the techniques may be sent to:

Dolores Wallace or Rick Kuhn
National Institute of Standards and Technology
Technology Building, B266
Gaithersburg, MD 20899

TECHNIQUE: CLEANROOM

HOW IT WORKS

***Representation Used**
- -Box structures
- -Usage profile

***Steps Performed**
- -Separate teams
- -Formal specifications
- -Incremental builds (design, implement, certify)
- -Reliability assessment

***Artifacts Produced**
- -Formal Specifications (Box abstraction)
- -Design Notation (Box refinement)
- -Certification of S/W (MTTF)

***Roles Involved**
- -Person to Task Mapping:
  - -Development team
  - -Certification team
  - -Specifications team

WHAT IT ACHIEVES WITH RESPECT TO HIGH INTEGRITY

***Positive**
- -High quality S/W
- -Certified reliability

***Negative**
- -Psychological impact
- -Dynamic requirements
- -Reliance on builds

***Other Techniques**
- -Inspections
- -Box structures
- -Statistical testing
- -Stepwise refinement

CURRENT APPLICABILITY OF A TECHNIQUE

- -Limited exposure
- -Successful results (SEL, IBM, University of Maryland)
- -Who is researching it:
  - -SEL, SET, University of Maryland, University of Tenn.
- -Related to other research/education (inspections, stepwise refinement)

***Recommendations**
- -Input from SET and H. Mills
- -Examine current success in production environments

TECHNIQUE: EHDM

HOW IT WORKS

***Representations Used**

- -Text (no graphics)
- -Syntactically enhanced "typed Higher Order Logic", Hoare sentences, Ada Code Synthesizer, hierarchical mappings
- -Built in theorem prover centered around decision procedures for linear arithmetic, propositional calculus, etc.

***Steps Performed**

- -Mechanics:
  - -Construct top-level formal spec
  - -Construct formal algorithmic-level (executable) specification
  - -Map abstract objects of formal spec onto algorithmic-level spec
  - -Construct proof that algorithmic level implements top-level formal spec
  - -Push proof through theorem prover
  - -EHDM tool synthesizes Ada Code from low-level spec
- -Tools:
  - -Type-checker
  - -Prover
  - -Proof-chain analyzer
  - -Ada synthesizer
- -Roles:
  - -Specification done by people
  - -Proofs discovered by people
  - -Theorem prover "understands" linear arithmetic and propositional calculus, so proofs must be reduced to this level but not down to the axioms of arithmetic
  - -Code synthesis (Ada) by tool from Hoare-sentences

***Artifacts Produced**

- -SPECS Formal
- -Proofs
- -Built-in tool to translate from EHDM syntax to more traditional mathematical notation (a .tex file)

***Roles Involved**

- -Person to Task Mapping:
  - -Abstract specification
  - -Detailed design (making design decisions)
  - -Mathematical spec of design and mappings
  - -Hand proofs
  - -Pushing proofs through theorem prover

- -Skills Required (Understanding of Applied Logic)
  - -Requires some mathematical "maturity"
  - -Requires specialized knowledge of tool-
    skolemization of formulas, instantiation, Lambda
    calculus, etc.

WHAT IT ACHIEVES WITH RESPECT TO HIGH INTEGRITY

***Positive**

- -Errors Identified:
  - -Very expressive specification language
  - -Design and implementation errors identified and removed
  - -Mathematical proof between abstract spec and detailed
    algorithmic spec
  - -Connection between code and spec through synthesis
  - -Correctness depends upon "validity" of the synthesis
  - -Algorithms
- -Reuse Possibilities:
  - -EHDM has direct support for "parameterized" modules
    making it suitable for verification of generic Ada modules

***Negative**

- -Fallibility:
  - -Depends correctness of theorem prover and
    synthesis algorithms
- -Bottlenecks:
  - -Proof process tedious
- -Technical barriers:
  - -Need decision procedures which encompass
    larger portions of typical formula to
    minimize need for human effort

***Other Techniques**

- -Testing
- -Simulation
- -Reliability Analysis due to physical failure

CURRENT APPLICABILITY OF A TECHNIQUE

- -Domain of Applicability:
  - -Universal but with various degrees of difficulty
- -Where is it being used:
  - -NASA, Langley, NRL, SRI International
- -Who is researching it:
  - -SRI International
- -Why are they doing this:
  - -Believe abstract specifications and refinement mapping
    is the proper framework for designing High Integrity
    Software
  - -To make money
- -Critical subsystems:
  - -Fault-tolerant clock synchronization

TECHNIQUE: FDM

HOW IT WORKS

***Representation Used**
- -Text
- -First order logic with types and equality
- -Finite state machine paradigm

***Steps Performed**
- -Synthesis and Analysis Steps
  - -Specify security invariant, transition constraints
  - -Write pre and post conditions for "transforms" (state machine transitions)
  - -Refine transforms to more detail
  - -prove transforms against im.
- -Tools Used:
  - -ITP theorem prover
  - -Inaflo flow analysis tool
  - -Tools available for verifying multi-level security properties

***Artifacts Produced**
- -Formal Specification
- -Proof transcript

***Roles Involved**
- -Person to Task Mapping:
  - -Domain expert, S/W developer, code expert, FM expert
- -Skills Required:
  - -Ability to translate requirements into mathematical logic
  - -Requires some mathematical "maturity"
  - -Requires specialized knowledge of tool- skolemization of formulas, instantation
  - -For MLS tool requires understanding of multi-level security concepts

WHAT IT ACHIEVES WITH RESPECT TO HIGH INTEGRITY

***Positive**
- -Errors Identified:
  - -Ambiguities uncovered
  - -Design and implementation errors identified and removed
  - -Mathematical proof between abstract spec and detailed algorithmic spec
  - -Connection between code and spec through synthesis.
  - -Correctness depends upon "validity" of the synthesis
  - -Algorithms

***Negative**

- -Fallibility:
  - -Depends on correctness of theorem prover and synthesis algorithms
- -Bottlenecks:
  - -Proof process tedius
- -Technical barriers:
  - -Need mechanisms to handle concurrent systems
  - -Need decision procedures which encompass larger portions of typical formula to minimize need for human effort

CURRENT APPLICABILITY OF A TECHNIQUE

- -Domain of Applicability:
  - -Security initially
  - -Could be applied to any system expressible as a finite state machine
  - -Non-real time
- -Where is it being used:
  - -NCSC/NSA, NIST, Unisys (SDC), vendors of secure systems
- -Where is it taught:
  - -National Computer Security Center offers courses
- -Who is researching it:
  - -R. Kemmerer, MCC (Aslan)
- -Why are they doing this:
  - -Formal verification considered necessary for secure systems
  - -To make money

TECHNIQUE: ESTELLE

HOW IT WORKS

***Representations Used**

- -Text:
  - -Nested Pascal style source (packed similar to Ada and Modula2)
- -Graphics:
  - -Accepted popular format for presentation of module layout and communication paths
- -Executable:
  - -Yes, Model
    Extended state transition, asysnc
    infinite queue, message parsing
    hierarchical execution model(priorities)
    atomic transitions, formal modular
    interface with interaction points

***Steps Performed**

- -Mechanics:
  - -Tools exist for: simulation, concurrency analysis prototype distributed code generation, graphical front end
  - -Translates extended state machines into c/c++/smalltalk for simulation etc..
- -Synthesis Steps:
  - -Create specification, compile, simulate(execution compilation with library support), separately specify structure for distribution of modules, run in distributed "mode" to identify "real" timing issues
- -Artifacts Produced:
  - -Structured process/module layout of system, details of communication path semantics
  - -Executable form(simulation/true distributed execution)
  - -State machine analysis(tools exist for analyzing source)

***Roles Involved**

- -Person to Task Mapping:
  - -Could involve the entire life cycle(since it can be used as high-level programming language), only supports process structure(location, interface, state model, communication)
  - -No support for data management or user interface
  - -Specifier of communication/process structure (specification is executable->specifier=programmer)
- -Skills Required:
  - -CS degree, Pascal, familiarity with finite state machines and queued message passing semantics

WHAT IT ACHIEVES WITH RESPECT TO HIGH INTEGRITY

***Positive**

-Errors Identified:
- -Simulation simplifies testing, inherits strong typing from Pascal
- -Structure syntax enforced

-Evaluation Data Produced:
- -State execution traces in simulation

-Reuse Possibilities:
- -Modular design enforced, information/structure hiding, process is a type with instances created dynamically, process interface is constant (implementation chosen at run time)

***Negative**

-Fallibility:
- -Does not model time or timing, deadlocks possible (detectable sometimes only within FSM [analysis of queued communications more difficult])

-Bottlenecks:
- -Cannot model all useful process combinations/ structures (includes own process semantics)

-Technical Barriers:
- -Very strict execution semantics

***Other Techniques**

-Data descriptions (ASN.1, ACT1), data management model

CURRENT APPLICABILITY OF A TECHNIQUE

-Domain of Application:
- -Communication protocols, distributed system structure

-Use/Research:
- -ISO protocol documents(formal description for protocol standards)
- -University of Delaware, NIST, ESPRIT/SEDOS, NCR, Phoenix Technologies

-Why:
- -Resulting documentation of protocol/system interfaces, correctness testing, automated implementation, conformance testing of protocols

-Scalability:
- -Have applied it fifteen independent state machine systems, practicality for large systems, similar to high level languages

-Prototyping:
- -Has been used

-Changes:
- -Recommendations for its evolution within standards community

-Adapt/deal with change:
- -Superior to most programming languages (more

similar to them than to Formal Methods)

TECHNIQUE: LARCH

HOW IT WORKS

***Representation Used**
- -Text
- -Non-executable

***Steps Performed**
- -Synthesis and Analysis Steps:
- -Trying to Define Data Types
  - -Identify constructors
  - -Identify behavior
  - -Identify modifier
  - -Write axioms
  - -Check consistency, completeness
  - -Attempt to prove deductions (theorems)
- -Specify Module Operations
  - -Write pre and post conditions for routines
  - -Use the theory you have already specified
  - -Construct a mapping between levels (give def)
  - -Check the mapping via commutative diagrams
- -Tools:
  - -Different tools for different languages
  - -Penelope uses Larch for ADA

***Artifacts Produced**
- -Specific documents
- -Proofs
- -Some results of analysis

***Roles Involved**
- -Domain expert (source of info, informal spec)
- -Method expert (translation to formal)
- -Code expert (translation to code)

WHAT IT ACHIEVES WITH RESPECT TO HIGH INTEGRITY

***Positive**
- -Inconsistencies and incompleteness(know the behavior of any sequence of operations)
- -Satisfaction of specifications can be demonstrated
- -Ambiguities uncovered
- -LSL allows reuse of theories

***Negative**
- -Hard to reuse
- -May be hard to write axioms (indirect)
- -Highly trained personnel required
- -Very little feedback from technology

CURRENT APPLICABILITY OF A TECHNIQUE

- -Domain of Application:

- -Small-medium size programs
  - -Non real-time, sequential
- -Where is it Used:
  - -Research
- -Where is it Taught:
  - -Universities (CMU, MIT, MD)
- -Who is Researching it:
  - -Wing, Guttag and Horning
  - -Digital
  - -Gerhart
- -Scalability:
  - -Sometimes it dies, sometimes it doesn't
  - -Some ideas scale, some are terrible
- -Adaptability:
  - -Suffers like scaling

TECHNIQUE: DESIGN IDEF0 (PETRI NET)

HOW IT WORKS

***Representation Used**
- -Text and Graphics:
  - -FSM with tokens (graphical) V IDEF0 block diagrams
  - -Automatically translatable into IDEF0 diagrams from Petri nets
  - -Automatically translatable into Petri nets from IDEF0 diagrams
  - -Data dictionary contains textual description of system
- -Executable:
  - -Usable for small to mid-size systems design and analysis

***Steps Performed**
- -Mechanics:
  - -IDEF0 -> resources, control, inputs, outputs
  - -Petri net -> places, arcs, transitions, colors
- -Synthesis and Analysis Steps:
  - -Create quotient machines (group "like" states to reduce the complexity of the directed graph)
  - -Look for deadlock, starvation of the processes, etc.
- -Tools Used:
  - -Editor (graphical or textual)

***Artifacts Produced**
- -Documents:
  - -MIL-STD 2167A reports, diagrams, error reports, (syntactic checking, dynamic testing)
- -Data:
  - -Data can be exported to other tools
- -Representations:
  - -Either IDEF0, Petri net or textual description

***Roles Involved**
- -Person to Task Mapping:
  - -Usually the entire specification of the diagram is done by one or a few analysts
- -Skills Required:
  - -Mathematics background with familiarity of automata theory and tractability/decidability theory (especially for colored Petri nets)
  - -Systems engineering background for using IDEF0 notation

WHAT IT ACHIEVES WITH RESPECT TO HIGH INTEGRITY

***Positive**
- -Errors Identified:
  - -Race conditions, incompleteness
- -Reuse Possibilities:

-Provides the analyst with a way to identify "like" portions of a diagram

***Negative**

-Fallibility:

-The larger the diagram, the longer it takes to identify all errors (computationally intensive process other then for purely syntactic checks)

-Bottlenecks:

-Hard to manage diagrams once they get large

-Technical Barriers:

-Theory underlying color Petri nets is still evolving -> no standards

***Other Techniques**

-Required:

-Simulation of the net or IDEF0 diagram

-Supported:

-Algorithm design for sequential and parallel systems; control and resource analysis

CURRENT APPLICABILITY OF A TECHNIQUE

-Use:

-By DOD (especially the USAF) for specification and design of systems

-Where is it taught:

-You must teach yourself IDEF0 notation from standards; Petri nets are usually taught in university courses on automata theory or decidability theory

-Who is researching it:

-University community, design IDEF vendor, DOD

-Scalability:

-Does not scale very well

-Why are they doing this:

-Required by the DOD

-Flexibility:

-Not very flexible

-Domain of Applicability:

-Complex systems with strict controls and resource requirements

TECHNIQUE: TRACE SPECIFICATIONS

HOW IT WORKS

***Representation Used**
- -Text and Graphics:
  - -Sequences of input [output] events
  - -Tables

***Steps Performed**
- -Specifications:
  - -Identify external events
  - -Identify legal traces
  - -Identify command forms
  - -Show how to reduce legal sequences to command forms
  - -Define output for command forms
- -Analysis:
  - -Analyze specifications for soundness and correctness
  - -Rapid prototyping (tools)
- -Implementation:
  - -Refinement of command forms into states
  - -Implement state machine
- -Verification:
  - -Prove program satisfies specifications
- -Tools:
  - -Rapid prototyping tool, table manipulation tools

***Roles Involved**
- -Person to Task Mapping:
  - -Fits well with some other methods (e.g. cleanroom)
  - -Provides front end for state based tools
- -Skills Required:
  - -Ability to think about what you want not how to do it (abstract/formal)

WHAT IT ACHIEVES WITH RESPECT TO HIGH INTEGRITY

***Positive**
- -Errors Identified:
  - -Survivor change
  - -Detects interface level errors before time/energy spent on design/implementation
- -Reuse:
  - -Modularity (decreases compiling)

***Negative**
- -Fallibility:
  - -Hard work
- -Bottlenecks:
  - -Can be long for certain applications
- -Technical Barriers:
  - -No performance analysis on initial specifications
  - -No provisions for timing consideration

CURRENT APPLICABILITY OF A TECHNIQUE

- -Use:
  - -AECB, Northern Bell, NRL, Ontario Hydro
- -Where is it taught:
  - -University of Victoria/Queens
- -Who is researching it:
  - -University of Victoria/Queens, NRL, AECB, Ontario Hydro
- -Why are they doing this:
  - -Verifiability
  - -Information hiding (promotes simple design)
  - -Testability
- -Changes:
  - -Notation
  - -Tools
  - -Timing
- -Scaling:
  - -Should be O.K. if information hiding is used
- -Understandability:
  - -People need special training
- -Domain of Application:
  - -Small S/W control systems
  - -Device drivers
  - -Communication protocols

APPENDIX D. PAPER SUBMITTED AT WORKSHOP

Cobb, Richard H., Mills, Harlan D., and Jesse D. Poore, "Comments Prepared for Participants in Workshop on Assurance of High Integrity Software"

COMMENTS PREPARED FOR PARTICIPANTS IN WORKSHOP ON ASSURANCE OF HIGH INTEGRITY SOFTWARE

by

Richard H. Cobb
Harlan D. Mills
Jesse D. Poore

Software Engineering Technology, Inc.
Vero Beach, Florida
(407) 569-3722

We believe that there is sufficient evidence to indicate that very high quality software is developed when Cleanroom Engineering practices are used. Therefore, we wish we were able to participate in the workshop on Assurance of High Integrity Software in order to help develop perspectives that included Cleanroom ideas. Since schedule conflicts prevent this we would like to make some points for consideration by each of the working groups.

For workshop participants who are not familiar with Cleanroom Engineering we have included two attachments. First is a summary of Cleanroom Engineering principles and second is a copy of a recent paper (November 1990 IEEE Software) by Cobb and Mills which discusses "Engineering Software under Statistical Quality Control".

Techniques for assuring, demonstrating high integrity

Standards should require:

The use of software construction processes and practices that have been demonstrated to have a high expectation that engineers will produce software that exhibits less than 5 failures per KLOC before any compilation or unit testing by an independent testing (certification) team.

The preparation of formal specifications which include at the minimum the following:

A precise statement of the mission (requirements) the software is to fulfill.

A definition of the stimuli and responses invented so the software can fulfill its mission.

A function which defines software responses in terms of stimuli histories.

A rigorous argument that the software as defined will fulfill its defined mission.

The expected usage profile of the software when it is being used to fulfill its assigned mission.

A plan for constructing the software in a series of accumulating increments such that each accumulation of increments is executable by user stimulus and the results are observable as user responses.

The use of statistical quality control techniques to estimate the mean time to failure for the software as developed.

The requirement that if during certification testing more than 5 errors per thousand lines of code are found in any increment that the code and design for that increment will be discarded and the increment will be redeveloped.

Our rational for setting an upper limit on the number of faults per KLOC that can be tolerated is that fixing software is the most error prone activity associated with software development. See our comments in relation to Forbidden Practices. The only engineering practices we know of that lead to producing software that has a high probability of producing software that has less than 5 defects KLOC prior to any compilation and testing is Cleanroom.

Cost - Benefit Analysis of Automation

To date no specific automation tools to support the Cleanroom Engineer have yet been constructed. The required tools have been identified. The next step is to complete prototypes to finalize the specifications.

Until tools are developed it is possible to develop software using Cleanroom Engineering practices at the rate of some 750 lines of code per staff month over the entire project cycle. This productivity is much higher than conventual practices can produce so there are significant savings even with out specific Cleanroom tools. In a few years time the organizations utilizing Cleanroom practices can look forward to productivity gains as tools come on line and their skills improve. We feel it is safe to assume productivity increases to bring total productivity to near 1500 lines of code per staff month over the entire project cycle.

Forbidden Practices

No debugging or compilation by the development team. Debugging is the most error prone activity associated with software development. It is vital that software developers stop this activity. Experience indicates that on the average that for every 5 fixes made in the software at least one of the fixes inserts a deep fault into the system that will remain in the system through all testing and only be found by some user during operational usage. The same data also indicates that the best an organization can hope for in debugging is that they will only insert 1 deep fault for each 10 debugging fixes made. The Washington Post recently reported that Lotus Development had to fix 20,000 problems alone in the new release of 1-2-3 following initial customer usage. That means that these developers inserted at least 2,000 deep latent failures into the system and more likely 4,000. That is terrible legacy from debugging to give the users of such an important product as 1-2-3 that is representative of all software development using conventional practices. It is possible that the Lotus engineers introduced no latent failures since we are discussing expectations but that is very unlikely.

There is no action software developers can take that would improve quality more than replacing private debugging by functional verification.

Hazard identification and criticality assessment

There are two issues relative to software. There is the specification of the function the software is to perform and then there is the development of the rule to implement the specification. The second of these two requirements is by the far the easier problem. Using Cleanroom Engineering there is a very high probability that the software as developed will be of contain very few latent failures and there is even a reasonable probability that for systems in the 50 to 100

KLOC range the software will be failure free in operational use. It may be desirable for safety critical systems to develop the rule with practices that say that if any significant fault is found in the software the code and design for that increment will be discarded and the increment will be redeveloped most likely by another team.

The difficult problem is preparing the specifications. This is where hazard analysis must be performed to insure the function the software is to implement takes into account all the hazards that may occur. In this way the proper design trades can be made.

Summary

We hope these comments are helpful in developing a perspective to guide the development of high integrity software.

We know the Cleanroom Engineering practices we are recommending are not widely used in software development but they are only evolutionary improvements of the practices that are now in common use. In the following diagram we characterize the practices used in three classifications of software development practices used by many observers of the industry. Traditional practices were the practices used by early developers. Modern practices are those used by many organizations today. Cleanroom practices are the ones we are advocating. The recognition of the benefits of Structured Programming were the principal driving force between the movement from traditional to structured programming. The driving force behind the recommendation to write structured programs was program verifiability. The move to structured programming was so great that verifiability seemed to get lost. The movement to Cleanroom Engineering is to continue the structured strategy but add verifiability.

The data presented in the following table is derived from a study by Dyer and Kouchakdjian of a number of software projects. This study provides some interesting data in terms of quality and productivity measures for projects using traditional, modern and Cleanroom practices.

It is not uncommon that small evolutionary changes lead to revolutionary improvements in performance. Consider the impact of the match over flint and steel as a portable fire starting device. That is the case for Cleanroom Engineering.

Cleanroom Engineering Goals and Principles

Cleanroom Engineering is a set of rigorous software engineering development and certification practices developed and perfected over the past 25 years. A rigorous engineering practice is one that is based upon sound scientific and mathematical principles.

Cleanroom Engineering goals include:

To permit organizations to produce failure-free software by replacing heuristic based development practices with rigorous software engineering practices.

To permit organizations to certify software quality by applying statistical quality control practices to the software design process.

To permit organizations to improve software productivity by avoiding design errors since error removal is a significant portion of development time.

Experience indicates these are achievable goals. Project experience permits us to project that more often than not organizations using Cleanroom Engineering should be able to produce a 50 KLOC software system so that no failure is ever experienced by a user of the software. We already have examples of failure free programs. For example, the software for the IBM Wheelwriter typewriter is a 63 KLOC system operating on 3 processors which was released in 1983. SInce then it has had billions of uses and no failure has ever been detected.

The principles that guided the development of the Cleanroom technologies are:

1. Exploit the fact that a program is a rule for a mathematical function by

    identifying and recording in the product specifications the function the software must satisfy to perform its mission,

    expanding the function top-down in small steps until following the final expansion the complete rule for the function exists in the target programming language, and

    following each expansion with a verification to show that no "algebraic" error was made as a result of the expansion.

2. Utilize sound mathematical principles to guide software development.

    The mathematics of a program is defined in the book Structured Programming: Theory and Practice by Linger, Mills and Witt, Addison-Wesley, 1979.

    The mathematics of a system of programs is defined in the book Principles of Information Systems Analysis and Design by Mills, Linger and Hevner, Academic Press, 1986.

3. Avoid the most error prone activity associated with software development by not compiling or debugging the software before independent certification.

4. Exploit the fact that usage testing is at least 20 times more effective than coverage testing when searching for failures left in the software as a result of human fallibilities.

5. Exploit the fact that software use is stochastic in software certification by

   applying statistical quality theory to plan usage tests,

   applying statistical inference theory to determine software quality, and

   applying statistical quality control theory to apprise the software design and development process.

6. Organize three engineering teams to facilitate the development of high quality software as follows:

   Specification Team to prepare, enhance and maintain the specification and construction plan.

   Development Team to design and implement software increment by increment in accordance with the specification and verify that each increment meets the specification using logical argument.

   Certification Team to certify that the accumulated increments satisfy the specification by performing independent usage testing with tests developed in accordance with the expected usage profile by applying appropriate statistical theory.

Some people react to Cleanroom Engineering when they hear a discussion of functions and function rules by saying that must be esoteric mathematics and cannot be for serious software development. That is the wrong impression since we are talking about practical engineering practices that are based on serious mathematics but which have been specialized for the real world of software development.

We believe that sufficient evidence now exists to reach a conviction that any effort devoted to improving software engineering practices that does not utilize Cleanroom Engineering practices will yield only marginal returns. Therefore, we believe that this conviction should be central to the deliberations of the Workshop on Assurance of High Integrity Software.

NIST-114A
(REV. 3-90)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NIST/SP-500/190

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
August 1991

4. TITLE AND SUBTITLE

Proceedings of the Workshop on High Integrity Software; Gaithersburg, MD; Jan. 22-23, 1991

5. AUTHOR(S)

Dolores R. Wallace, D. Richard Kuhn, John C. Cherniavsky

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)
U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED
Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)

Same as item #6

10. SUPPLEMENTARY NOTES

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

This paper provides information related to the National Institute of Standards and Technology (NIST) effort to coordinate an effort to produce a comprehensive set of standards and guidelines for the assurance of high integrity software. The effort may include adapting or adopting existing standards as appropriate. In particular, the paper presents the results of a Workshop on the Assurance of High Integrity Software held at NIST on January 22-23, 1991. Workshop participants addressed techniques, costs and benefits of assurance, controlled and encouraged practices, and hazard analysis. A preliminary set of recommendations was prepared and future directions for NIST activities in this area were proposed.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

assurance; computer security; controlled practices; cost-benefit; criticality assessment; formal methods; hazard analyses; high integrity systems; software safety; standards

13. AVAILABILITY

X	UNLIMITED
	FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).
X	ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.
X	ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.

14. NUMBER OF PRINTED PAGES
85

15. PRICE

ELECTRONIC FORM

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SYSTEMS TECHNOLOGY

Superintendent of Documents
Government Printing Office
Washington, DC 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Institute of Standards and Technology Special Publication 500-.

Name ____

Company ____

Address ____

City __________________ State ______________ Zip Code ____________

(Notification key N-503)