SAND99-0059C

Software Reliability Cases: The Bridge Between Hardware, Software and System Safety and Reliability

Debra S. Herrmann, CSC, Crystal City
Dr. David E. Peercy, Sandia National Laboratories, Albuquerque

Keywords: software reliability, software reliability cases, software safety, software safety cases, software reliability standards, software surety.

SUMMARY & CONCLUSIONS

High integrity/high consequence systems must be safe and reliable; hence it is only logical that both software safety and software reliability cases should be developed. Risk assessments in safety cases evaluate the severity of the consequences of a hazard and the likelihood of it occurring. The likelihood is directly related to system and software reliability predictions. Software reliability cases, as promoted by SAE JA 1002 and 1003, provide a practical approach to bridge the gap between hardware reliability, software reliability, and system safety and reliability by using a common methodology and information structure. They also facilitate early insight into whether or not a project is on track for meeting stated safety and reliability goals, while facilitating an informed assessment by regulatory and/or contractual authorities.

1. INTRODUCTION

The concept of safety cases has been evolving since the mid 1980s. A safety case is a generally accepted practice for reporting data needed by contractual, regulatory, and/or independent third-party certification authorities. A generic safety case is structured to include: system safety requirements and their allocation, assumptions and/or claims based on pre-existing systems, evidence, conclusions and recommendations. The safety case addresses all components of system safety, including hardware and software. It provides a systematic process for collecting, analyzing, and interpreting data throughout the lifecycle, which can be used as evidence and for monitoring whether or not a project is on track for meeting stated system safety goals. An equivalent concept, software reliability cases, is needed to bridge the gap between hardware reliability, software reliability, system safety and reliability. Accordingly, this paper will present an approach for implementing software reliability cases, based on the new international standards SAE JA 1002, Software Reliability Program Standard[7] and SAE JA 1003, Software Reliability Implementation Guide[8], published by the Society of Automotive Engineers (SAE).

Nomenclature list:

1. **software safety**: features and procedures which ensure that a product performs predictably under normal and abnormal conditions, thereby minimizing the likelihood of an unplanned event occurring, controlling and containing its consequences, and preventing accidental injury, death, destruction of property and/or damage to the environment, whether intentional or unintentional.

2. **software safety case**: evidence collected throughout the project lifecycle which proves that software safety requirements are consistent with system safety requirements and that they have been achieved, to enable an Authorizing Body to gain assurance that the software is safe and fit for purpose, in the intended operational environment.

3. **software reliability**: (1) the probability of failure-free operation of a software program for a specified time under specified conditions; (2) a set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.

4. **software reliability case**: evidence presented throughout the project that software reliability requirements are consistent with system level requirements, are achievable, are understood by the development organization, and that ambiguities have been resolved.

5. **software surety**: attributes of and activities associated with achieving and assessing software safety, security, and reliability.

2. SOFTWARE RELIABILITY CHALLENGE

Increased use of firmware and embedded software is blurring the boundary line between software and hardware. Regardless, software is not hardware and it has unique characteristics in relation to safety and reliability. In contrast to hardware, software does not break, wear-out over time, or fall out of tolerance. Hardware reliability models are based on variability introduced in the manufacturing process and the physics of failure. Many hardware reliability models do not apply to software since software is not physical; i.e. it is not possible to perform destructive testing on software.

DISCLAIMER

Consequently, different paradigms must be used to evaluate software reliability.

There is an ongoing debate within industry, academia, and the international standards community on whether or not software reliability can in fact be quantified. Some standards promote a qualitative assessment of software reliability while others promote a quantitative assessment. J. Bieda[9] notes that there is also a debate on whether software reliability models should be time related or not. R. Rees[19] observes that the main argument against a time-related software reliability model stems from the fact that software failures are not discovered until particular program or data sequences are processed. In other words, the failure will occur the first time the defective sequence is processed; however, this may have been preceded by weeks or months of testing. Hence, the time factor is often irrelevant.

Traditional software reliability growth and estimation models are based on the number of errors found during testing and the amount of time it took to discover them. These models use various statistical techniques, many borrowed from hardware reliability assessments, to estimate the number of errors remaining in the software and to predict how much time will be required to discover them. These models provide metrics which are useful for project management, i.e. to determine when a commercial product is ready for release and how much field maintenance will be required.

Traditional software reliability models do not distinguish between functional, performance, safety, reliability, or security errors. These models do not distinguish between the severity of the consequences of the errors (negligible, marginal, critical, catastrophic) found or predicted to be remaining in the software. Nor do they take into account errors found: 1) by static analysis techniques, or 2) in phases prior to testing. New software reliability models are being developed to address these deficiencies[13].

Software safety is a component of system safety; likewise software reliability is a component of system reliability. Software safety and reliability requirements must be consistent with and correspond to system safety and reliability requirements. As a result, a reliability engineer is presented with a challenge – how to integrate and interpret dissimilar hardware and software reliability models in order to derive system reliability, especially when working in a high integrity/high consequence environment.

3. SOFTWARE SAFETY CASES

A software safety case, a component of a system safety case, is evidence collected throughout the project lifecycle which proves that software safety requirements are consistent with system safety requirements and that they have been achieved, to enable an Authorizing Body to gain assurance that the software is safe and fit for purpose, in the intended operational environment. The concept of a safety case has been well established in the U.K. since the enactment of the Health and Safety at Work Act of 1974. Safety cases are required in a number of industries, such as aerospace, chemical, nuclear, offshore, and railway. Different industries have specific requirements regarding safety cases. For example, railway safety cases (RSCs) are "a particular form of safety case which places a legal obligation on an organization to document their commitment to safety, together with how proper discharge of these commitments will be made in their respective activities... The RSC provides assurances to the Authorizing Body that the Duty Holder has management structures in place that will identify hazards, assess risks and impose suitable and sufficient control measures and that there will be compliance with Railway Group Standards."[11] B. Lawrence[15] reports that recent aerospace standards[1] also levy an explicit requirement for safety case development.

A safety case should present "a clear, defensible, comprehensive and convincing argument, ... aimed at identifying the risks inherent in operating a system, demonstrating that the operating risks are fully understood, that they have been reduced to an acceptable level and are properly managed"[22]. Shaw[22] emphasizes that, "Although one of the roles of a safety case is for submission to an industry regulator, this should not be its prime function." Safety cases should also be developed for the benefit of the system owners, to provide insight into the risks associated with all aspects of a system, so that the risks can be better managed.

There are three generic categories of safety cases: system design safety cases, system operational safety cases, and system decommissioning safety cases. The operational safety case is updated anytime modifications are made to the system or operational procedures which could affect safety. Cook[11] states that, "The safety case will be a live document. It will be updated as ... data from actual operation is collated to support, or otherwise, the statements made in the justification." From his experience with the British Railway Business Systems (BRBS), Tilloston[23] proposes a slightly different view. He recommends the development of supplier safety cases, user safety cases, and change control report safety cases. The change control report safety cases capture all new risks and all interfaces that have been changed, to

---

[1] - Aerospace Recommended Practice (ARP) 4754 (1996), 'Certification Considerations for Highly-integrated or Complex Aircraft Systems,' Society of Automotive Engineers (SAE).

- Aerospace Recommended Practice (ARP) 4761 (1996) 'Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment,' Society of Automotive Engineers (SAE).

demonstrate that the system is "no less safe after the enhancement has been applied than it was before."[23]

The generic structure for a safety case includes: 1) a discussion of the system safety requirements and their allocation, 2) assumptions or claims based on experience with pre-existing systems, 3) evidence, and 4) conclusions and recommendations. A safety argument is made from a combination of the assumptions or claims and evidence. Bishop and Bloomfield[10] have identified three types of arguments: 1) deterministic – arguments based on axioms, proofs, logic, or prior research and experience, 2) probabilistic – arguments based on failure rates, static analysis, and/or assumptions about independence, and 3) qualitative – arguments based on compliance with standards and industry best practices.

Bishop and Bloomfield[10] and Shaw[22] point out that information to support a safety argument will come from multiple sources, such as the: product and its components, design and development process, capability of development organization, performance of analogous systems, industry experience with the effectiveness of specific development methods and tools, and field experience. From observations on the CASCADE project, Rivett[20] points out that conformity to standards alone is insufficient justification for the safety argument of the system – additional evidence is needed. He also notes that a safety argument can provide an effective means for documenting the view points of different parties (developers, operators, and maintainers) in a single document.

As K.M. Wright[25] observes, "Risk identification and evaluation is performed using reliability and safety analyses in a mutually supportive way." Reliability analysis identifies function criticality, failure effects, and mission success criteria while developing reliability predictions. Safety analysis identifies hazardous conditions, the severity of their consequences, and the likelihood of them occurring. Throughout the risk identification and evaluation process, several interfaces occur among reliability, safety, and security, i.e. surety analyses; such as identification of safety-critical functions, failure modes, and accident causes and development of reliability predictions for safety critical-functions. These interfaces occur, in part, as a result of interaction between the fault tree analysis (FTA) and the failure modes effects criticality analysis (FMECA). Consequently, it is logical that a software safety case and a software reliability case should be developed in parallel. (See Figure 1.)

4. SOFTWARE RELIABILITY CASES

A software reliability case provides a justification of the approach and documents evidence that verifies the software meets reliability requirements. A software reliability case provides convincing evidence that software has been designed, tested, and accepted using adequate processes. The case demonstrates product characteristics that a specified level of failure-free operation has been achieved. It provides the necessary "bridge" to understanding software's relationship to potential hardware and system failures. Evidence can be presented in a variety of forms, but generally is represented by quantitative and/or qualitative measures. Supporting arguments and inferences also provide a level of confidence that the evidence does indeed imply the software reliability requirements have been achieved.

There are three generic types of software reliability cases: 1) pre-development software reliability case, 2) development software reliability case, and 3) in-service software reliability case. The pre-development software reliability case provides an analysis of the supplier's approach to reliability achievement. It provides evidence that:

- the individuals and organizations involved are capable of supplying software that is commensurate with the reliability requirements of the proposed system;
- the software reliability plan is appropriate for the proposed system reliability requirements;
- technical proposals and decisions upon which the tender is based are appropriate for the proposed system reliability requirements; and
- the supplier will be able to demonstrate reliability achievement during the project.

During development, the supplier should update the software reliability case with a summary and appraisal of the results of the activities that contribute to the reliability evaluation. By the time of acceptance into service, the case should contain (or reference) the complete set of evidence that the reliability requirements of the software will be met. The software reliability case should describe measurements taken of the software products and the software engineering process that provide evidence development is proceeding satisfactorily. Versions of the case should be planned for appropriate milestones in the software development lifecycle, for instance after reliability tests and trials. The current version of the case should be presented at design reviews and the outcome included in the case. Versions of the case should be managed under a configuration control mechanism to ensure compatibility with development status.

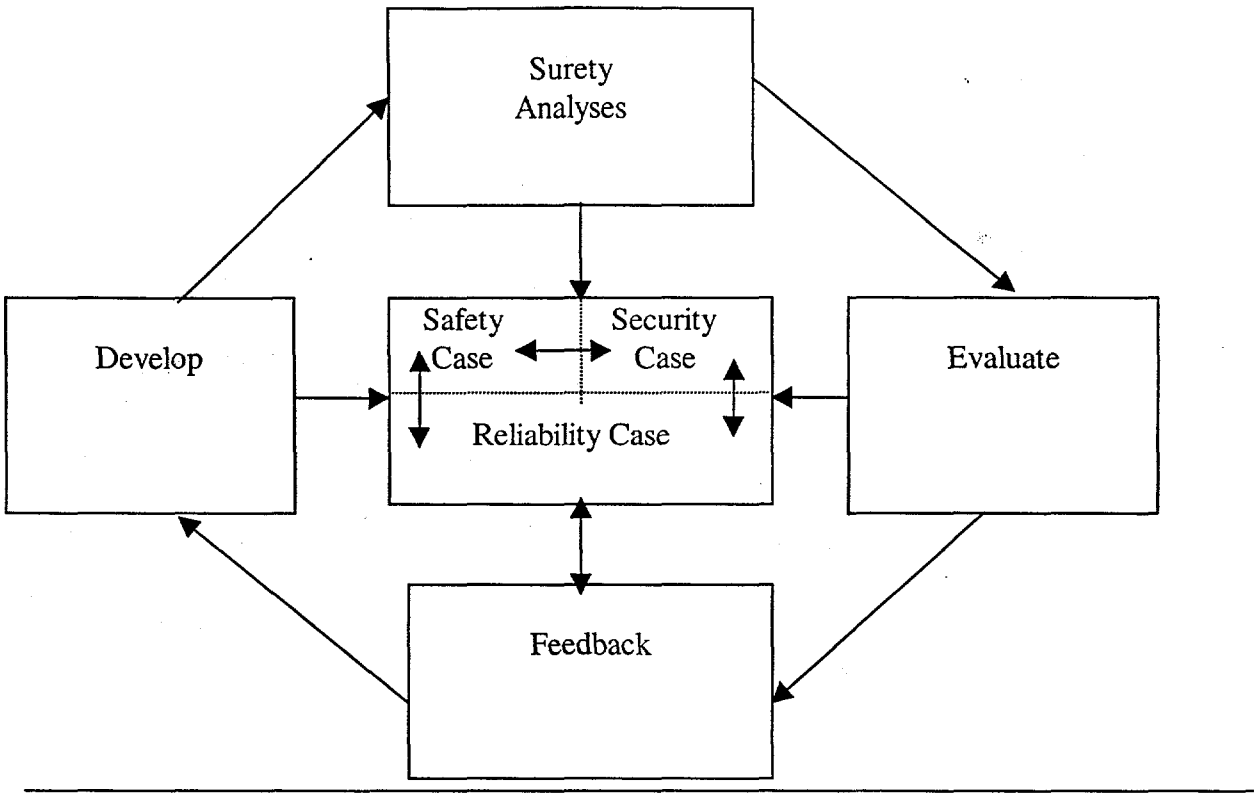

Figure 1. Joint Development of Surety Cases.

In-service software reliability management includes the collection of operational data and the maintenance of an in-service software reliability case. The case should contain a description of field experience with the software and an analysis of the impact of software failures on system reliability. The analysis should address the potential consequences of a failure, root causes, mitigation strategies, and the lessons learned for the software engineering process. In-service reliability data can be used for:

- assessing or confirming reliability achievement;
- determining reliability impact when software is used in a new environment;
- gathering experience on the performance of particular methods, techniques and processes for the benefit of future projects; and
- assessing effectiveness of change implementation during software support.

A software reliability case presents arguments and evidence that requirements can be achieved, will be achieved, and have been achieved. For maximum effect, the case should be developed and witnessed as development decisions are made. It is not intended to be a retrospective justification of the solution. The software reliability case should be a readable overview of the evidence that software meets its reliability requirements, with references to project development records and the results of analyses of software components as appropriate. The case should be a living document, proceeding through a number of stages of increasing detail during the project.

Like software safety cases, diverse types of evidence may be used within complementary approaches to demonstrate software reliability. Reliability evaluation of software with stringent reliability requirements is especially difficult. For most situations where stringent reliability requirements are to be met, the supplier should plan to provide diverse forms of evidence to cover each aspect of software reliability. The general structure of a software reliability case is very similar to a safety case. It contains:

1) claim - statement/requirement about a property of the system or component;
2) evidence - facts, assumptions, or sub-claims derived from lower-level sub-arguments;
3) argument - logical information linking the evidence to the claim using deterministic/analytical, probabilistic, or qualitative measures; and
4) inference - the mechanism that provides the transformational rules for the argument.

Direct evidence may be included in the form of:

1) defects collected during analysis, design, and coding, typically generated from inspections or other peer reviews, and evidence of corrections;

2) results from testing, including reliability growth modeling, statistical testing, data from tests and trials, and performance testing, with an assessment of its accuracy and relevance to operational use;
3) field data, including a description of any claims made on the basis of previous in-service experience, including a comparative analysis of relevant criteria;
4) results of any analytic arguments deployed to show the absence of certain faults, and the assumptions on which they are based; and
5) analyses of any third party software that is to be used and a description of the way in which it will contribute to software reliability, taking into account any changes in operational environment and use profile.

A software reliability case should also present indirect evidence in the form of an overview of the software engineering process to explain how it supports achievement of the software reliability requirements. Some items that might be included are:

1) An analysis of the methods, techniques and procedures to be used, with an assessment of their suitability for supporting the achievement of the reliability requirements. This includes a description of the generic types of software faults that will be specifically addressed and minimized; an analysis of reliability data from software developed using the proposed or similar methods, techniques and; and a justification of the choice of tools and support software, with a description of the way in which known problems (e.g. compiler faults) are to be handled and recorded.
2) 2) A description of how reliability progress is to be measured. This should include: measurements that are to be taken during the software development lifecycle of the performance of the software engineering process; the verification and validation strategy, including operational profiles, test coverage, and the means for generating tests; and acceptance criteria for these measures. During development, data should be recorded in the case and compared with the acceptance criteria. Corrective action to be taken if the acceptance criteria are not met should be described.
3) Confirmation that the minimum levels of personnel competency defined in management plans have been achieved.

5. LINKING SOFTWARE SAFETY CASES AND SOFTWARE RELIABILITY CASES

By linking software safety, security, and reliability cases redundant analyses will be reduced, complementary analyses will be increased, cost will be reduced, personnel will be utilized more effectively, and there will be broader assurance that all aspects of software that affect the surety of the system have been addressed. The development stages can use any combination of software reliability, safety, and security cases. Our recommendation is to always use a software reliability plan-case structure, as promoted by JA 1002 and JA 1003, for critical systems and to combine the cases when safety and/or security are a concern as well. It might be useful to call such a case a surety case and include appropriate subparts to address specific reliability, safety, and/or security concerns. N. Schneidewind [21] provides an interesting approach for the integration of software safety criteria, risk analysis, reliability prediction, and stopping rules for testing. Safety goals are defined in terms of reliability-related measures such as remaining failures, time to next failure, total test time, and risk criterion derived using these measures. Techniques and methods that might be used to support achievement and assessment of design and/or performance objectives for a software surety (reliability, safety, and security) case are listed in Table 1.

The required approvals will drive content of the software reliability plan and case. In particular, a graduated level of approval (with appropriate claims, arguments, inference, and evidence) will normally be required. When software safety is an issue, the legal aspects according to the law, warranties, definitions and agreements on what "goods fit for purpose" and "satisfactory quality" mean, and contractual agreements become a major part of the approval strategy. The general strategy is to identify surety activities that apply across as many of the approval levels as possible and satisfy multiple surety concerns.

JA 1002 Software Reliability Program Standard[7] was submitted for publication by SAE in June 1998 and is available from SAE. JA 1002 relies on the simple concept of a supplier-customer dialogue and partnership to define, meet, and demonstrate assurance of software product reliability requirements. This standard describes, within a plan-case framework, what performance requirements are necessary and is intended to be used by industries to address market demands for reliable software products that improve system productivity, time to market, and cost-effective implementation. JA 1002 does not describe or recommend specific software development practices. However, JA 1003 Software Reliability Implementation Guide[8] provides information about specific tasks that contribute to the overall reliability goal and also includes further details of plan and case development, like that described in this paper.

Technique/Method	Surety Activity	Case Focus
Safety Focus[4]		
Event Tree Analysis	Fault prevention, identification, removal	Design
Failure Modes, Effects, and Criticality Analysis	Fault prevention, identification, removal	Design
Fault Tree Analysis	Fault prevention, detection, removal	Design
Hazard Analysis	Fault detection, removal, tolerance	Design
Mathematical Specification Verification	Fault prevention	Design
Petri Nets	Fault prevention	Design
Sneak Circuit Analysis	Fault detection, removal	Performance
Software Change Analysis	Fault prevention, detection, removal,	Design
Specification Analysis	Fault detection, removal	Design
Testing (fault tolerant)	Fault detection, removal, tolerance	Performance
Reliability Focus[3]		
Analytical Arguments	Failure estimation, prediction, assessment	Design, performance
Checklists	Fault prevention	Design
Cleanroom	Fault prevention and tolerance	Design, performance
Defensive Programming	Fault detection, removal, tolerance	Design
Design Maturity	Fault prevention	Design
Design Reviews	Fault prevention, detection, removal	Design
Exhaustive Testing	Fault detection, removal	Performance
Fault Tolerance Verification	Fault detection, removal, tolerance	Design
Formal Inspections (In-Process Reviews)	Fault detection, removal	Design
Formal Methods	Fault prevention	Design
Human Factors in Software Design	Fault prevention	Design
Object-Oriented Methods	Fault prevention, detection, removal	Design
Performance Testing	Fault detection, removal	Performance
Prototyping	All	Design, performance
Reliability Estimation	Failure estimation	Performance
Reliability Growth Modeling	Failure estimation, prediction, assessment	Performance
Reliability Prediction	Failure prediction	Performance
Static Analysis	Fault detection, removal	Design
Statistical Testing	Failure estimation, prediction, assessment	Performance
Structured Design	Fault prevention	Design
Testing	Failure estimation, prediction, assessment Fault detection, removal	Performance
Traceability Analysis	Fault detection, removal	Design
Use of Field Data	Failure estimation, prediction, assessment	Performance
Reliability Focus[1]		
Recommended Practice for Software Reliability	Failure estimation, prediction, assessment	Performance
Reliability Estimation Models - Exponential NHPP - Non-Exponential NHPP - Bayesian	Failure estimation, prediction, assessment	Performance
Reliability Focus[2]		
Software Development Process Models	Fault prevention	Design
Software Property Models	Fault prevention	Design
Stochastic Reliability Models	Failure estimation, prediction, assessment	Performance

Table 1 - Methods and Techniques Supporting Software Surety Cases

6. REFERENCES

[1] ANSI/AIAA R-013-1992, AIAA Recommended Practice for Software Reliability, February 1993

[2] British Standards Institute Standard 5760, Reliability of Systems, Equipment and Components, Part 8: Guide to Assessment of Reliability of Systems Containing Software,
Draft, July 7, 1997.

[3] Defence Standard 00-42 (PART 2)/Issue 1, Reliability And Maintainability Assurance Guides, Part 2: Software, United Kingdom Ministry of Defence, 1997.

[4] IEEE Std 1228-1994, IEEE Standard for Software Safety Plans.

[5] JA 1000, Reliability Program Standard, Society of Automotive Engineers, 1998.

[6] JA 1001, Software Reliability - An Overview, Society of Automotive Engineers, (draft) 1998.

[7] JA 1002 Software Reliability Program Standard, Society of Automotive Engineers, 1998.

[8] JA 1003 Software Reliability Implementation Guide, Society of Automotive Engineers (SAE), (draft) 1998.

[9] Bieda, J. Software Reliability: A Practitioner's Model, Reliability Review, June 1996, vol. 16, no. 2, p.18-28.

[10] Bishop, P.G. and Bloomfield, P.E. The SHIP Safety Case Approach, Adelard, London, UK, 4 September 1995.

[11] Cook, R. "Management of Dependability: A Railway Perspective," Safety-critical Systems: The Convergence of High Tech and Human Factors, Springer-Verlag, 1996, p. 61-70.

[12] Edwards, C. "Railway Safety Cases," Safety and Software reliability Based Systems, Springer-Verlag, 1995, p. 317-322.

[13] Herrmann, D. Sample Implementation of the Littlewood Holistic Model for Assessing Software Quality, Safety and Reliability, Proceedings of the International Symposium for Product Quality and Integrity (RAMS 98), IEEE, January 19-22, 1998, Anaheim, CA.

[14] Lakey, Peter B., and Neufelder, Ann Marie, "System and Software Reliability Assurance Notebook," Rome Laboratory, September, 1996.

[15] Lawrence, B. "Safety Cases for Integrated Systems," Safety Systems, vol. 7, no. 2, January 1998, p. 8-10.

[16] Musa, John D. "Operational Profiles in Software Reliability Engineering", IEEE Software, March 1992.

[17] Purton, T. Contracting for Reliability – The UK/MOD View as a Buyer, British Crown, 1991.

[18] Reliability Analysis Center (RAC), Blueprints for Product Reliability, May 15, 1996.

[19] Rees, R.A. Detectability of Software Failure, Reliability Review, December 1994, vol. 14, no. 4, p.10-30.

[20] Rivett, R. "Is There a Role for Third Party Software Assessment in the Automotive Industry?" Safer Systems, Springer-Verlag, 1997, p. 160-184

[21] Schneidewind, N., "Reliability Modeling for Safety-Critical Software," IEEE Transactions on Reliability, Vol. 46, No 1, March 1997, p. 88-98.

[22] Shaw, R. "Safety Cases – How Did We Get Here?" Safety and Software Reliability Based Systems, Springer-Verlag, 1995, p. 43-95.

[23] Tilloston, J. "In the foothills of Systems Safety," Safety Systems, vol. 7, no. 1, September 1997, p. 11-13.

[24] Wilson, S.P., Kelly, T.P., and McDermid, J. "Safety Case Development: Current Practice, Future Prospects," Safety and Software reliability Based Systems, Springer-Verlag, 1995, p. 135-156.

[25] Wright, K.M. "Safety for European Space Agency Programs," Directions in Safety-critical Systems, Springer-Verlag, 1993, p. 17-35.

7. BIOGRAPHIES

Debra S. Herrmann, Principal Software Engineer
CSC
6030 California Circle #401
Rockville, MD 20852-4842
USA
703.413.7337 (v)
dherrma2@csc.com (e)

Ms. Herrmann has over twenty years professional experience in industry, academia, and government. She specializes in conducting software safety and reliability research and assessments across multiple industrial sectors and has published many papers in this area. Ms. Herrmann has been the technical lead for a variety of defense contracts involving the specification, design, development, test, and evaluation of secure, high reliability software and data communications applications; including performing software safety assessments of nuclear missile control software. She has performed software safety and reliability assessments of many new high risk medical devices as part of the FDA approval process. She taught a 14-week course for the FDA Staff College, "Safety and Reliability for Medical Device Software" and is the primary author of the new FDA software guidance document which describes the software risk management procedures to be followed by the biomedical industry. Ms. Herrmann has been in active in the international standards community for many years, as a representative to IEC, IEEE, and SAE committees developing software safety and reliability standards. She is a member of the ACM, Centre for Software Reliability, IEEE Computer Society, IEEE SESC Balloting pool, IEEE Standards Association, and the ASQ Software and Reliability Divisions She holds a M.S. in Computer Science from the University of North Texas and is a ASQ Certified Software Quality Engineer.

Dr. David E. Peercy, Chair G-11SW
Sandia National Laboratories
P.O. Box 5800, MS-0638
Albuquerque, NM 87185-0638
USA
505.844.7965 (v)
depeerc@sandia.gov (e)

Dr. Peercy has 25 years of experience, 26 publications, and numerous conference presentations in software development, supportability/maintenance, reliability, security and process methodology research. Dr. Peercy is currently chair of the Society of Automotive Engineers G-11 Reliability, Maintainability, Supportability, Logistics (RMSL) Software Committee that is developing standards and guidelines on software reliability and software supportability. At Sandia National Laboratories, Dr. Peercy conducts quality engineering activities on nuclear weapon related software systems and performs research and practical application of software process improvement, software reliability, software supportability, and software measurement across laboratory software programs. Dr. Peercy is co-author of an internal 1996 Sandia research report, SAND96-2924, "A Review of Research and Methods for Producing High-Consequence Software." The research documented State of the Practice and State of the Art methods and techniques in software reliability and safety that would be useful for Sandia use. Dr. Peercy also provides technical leadership for development of a lab-wide Software Management Program. Dr. Peercy obtained his Ph.D. in Mathematics from New Mexico State University in 1971 and is an active member of ACM, IEEE Computer Society, IEEE SESC Balloting pool, SOLE, and ASQ. Dr. Peercy is an ASQ Certified Software Quality Engineer.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.