Bi-directional Analysis for Certification of Safety-Critical Software

Robyn R. Lutz* and Robert M. Woodhouse†
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109-8099

April 2, 1999

Abstract

For safety-critical systems, it is insufficient to certify the developer and the development process. Certification of the software product itself is also needed. SFMEA (Software Failure Modes and Effects Analysis) and SFTA (Software Fault Tree Analysis) are two engineering techniques that have been used successfully for a number of years and in a variety of safety-critical applications to verify software design compliance with robustness and fault-tolerance standards. This paper proposes the use of Bi-directional Analysis (BDA), an integrated extension of SFMEA and SFTA, as a core assessment technique by which safety-critical software can be certified. BDA can provide limited but essential assurances that the software design has been systematically examined and complies with requirements for software safety.

1 Introduction

Even as requirements for software certification proliferate, the best approach to software certification remains in dispute [42]. One suggested piece of the solution is to certify the software developer or analyst. Another approach to software certification is to certify the development process (e.g., to assure compliance with ISO 9000-3, CMM or SPICE standards). For safety-critical software, a third approach, certifying the software product itself, is an essential aspect of software certification.

All three of these certification approaches (certifying the developer, the process, and the product) share a drive towards standardization (of credentials, of development processes, of software verification methods). All three approaches involve the necessary internationalization of certification as software production increasingly ignores national boundaries. All three approaches also involve independent evaluation of compliance of the developer/process/product against some pre-existing standard or guideline. However, since the goal is to certify the quality of the software product itself, we should–to the extent possible–directly evaluate that product. It is certification of the software product, specifically of the software product identified as safety-critical, that this paper addresses.

Certification of the software product involves an assessment of the software against certain criteria. As an example, a certification criteron for non-critical vendor software may be an assured interface for ready interoperability with a customer's other applications (e.g., Novell provides such a certification program). McDermid suggests that in the safety-critical arena, future COTS vendors may produce certificates for their software components that guarantee specific behaviors or capabilities [44]. Component libraries (class libraries) will need to offer similar guarantees to users. The difficulty is that certification criteria for safety-critical software are usually significantly more complicated and ill-defined than for other software [4]. The certification often needs to verify either the absence of unsafe behavior or that certain desirable behaviors always occur.

For safety-critical systems, it is insufficient to certify the developer and the development process. Certification of the software product itself is also needed. Many standards [5, 6, 37, 24, 46] and authors [12, 14, 16, 34, 41] discussing software safety, reliability and high integrity software products and development processes describe the use of forward (inductive) and backward (deductive) design and assurance methods to help ensure that software defects are iden-

*Published in *Proceedings, ISACC'99, International Software Assurance Certification Conference*, Chantilly, VA, Feb. 28–Mar 2, 1999. First author's address is Dept. of Computer Science, Iowa State University, Ames, IA 50011-1041, rlutz@cs.iastate.edu.

†Second author is currently with XonTech Inc., Suite 600, 6151 West Century Blvd., Los Angeles, CA 90045, robert_woodhouse@xontech.com.

tified and removed from software design. SFMEA (Software Failure Modes and Effects Analysis) and SFTA (Software Fault Tree Analysis) are two methods that have been used successfully for a number of years and in a variety of safety-critical applications to verify software design compliance with robustness and fault-tolerance standards.

This paper proposes the use of Bi-directional Analysis (BDA) as a core assessment technique by which safety-critical software can be certified. BDA builds on the accepted engineering methodologies of SFMEA (inductive) [11, 19, 22] and SFTA (deductive) [16] to provide critical information about the behavior of the software design. This information plays an important role in developing and certifying safety-critical software, defined to be "software that can directly or indirectly contribute to the occurrence of a hazardous system state" [16]. BDA can provide limited but essential assurances that the software design has been systematically examined and certify that it complies with software safety requirements.

The purpose of Bi-directional Analysis (BDA) is to show that the software design is free of certain critical flaws that can contribute to hazards. BDA is a systematic technique for identifying what can go wrong with each component of a system (its failure modes), what effects each failure mode can have as it propagates through the system, and what features enable or contribute to the possibility of that failure mode in the first place. The examination of the system consequences of software failures is a key piece of the software certification of safety-critical systems.

The rest of the paper is organized as follows. Section 2 gives an overview of BDA and provides background information regarding BDA's origins in related techniques. Section 3 describes the procedure for performing a BDA and provides examples of its usage. Section 4 discusses the benefits and limitations of BDA within a standard, certifiable development process. Section 5 links BDA to existing industrial and governmental standards and certification processes for safety-critical software, and offers some concluding remarks.

2 Overview of BDA

BDA first checks the design to determine whether the effects of abnormal (e.g., out-of-range) input values and unexpected software events (e.g., unexpected termination) can contribute to unsafe system behavior. Following Leveson [16], software safety is defined to be freedom from undesired and unplanned events that result in a specified level of loss. Software safety

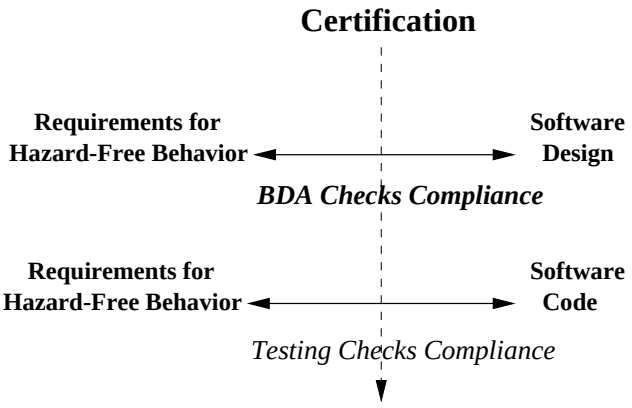

Figure 1: *Role of Bi-directional Analysis (BDA) in Certification of Safety-Critical Software*

analysis techniques determine how software can contribute to conditions that result in loss or failure. The forward direction of the BDA involves a forward analysis from abnormal inputs or events to non-compliant consequences, and has its roots in Failure Modes and Effects Analysis. Next, the BDA checks whether the non-compliant scenarios that have been identified are credible. This analysis either determines that the failure modes cannot occur given the design of this system or, if they can occur, that they are handled safely. This direction of the BDA involves a backward analysis from those abnormal scenarios with safety consequences to the collection of causes that might permit the identified scenario to happen. The second part of the BDA has its roots in Fault Tree Analysis.

For software there is no "seal of approval" that guarantees that software will behave safely. Instead, meaningful certification of safety-critical software is currently limited to a structured assessment, using well-documented techniques, that the software complies with certain specifications on its behavior [13]. For example, the ESPRIT2 project SCOPE (Software Certification Programme in Europe) pursued product certification by evaluating and assessing software compliance against requirements in a documented standard [39].

Fig. 1 shows the role of BDA (design certification) and testing (code certification) for safety-critical software. Code testing is the most important means of software certification. However, since testing is always partial and incomplete, assessment of design compliance with required behavior is also needed. In addition, design compliance can be assessed prior to testing, allowing needed changes to be made earlier in development and contributing to test scenarios.

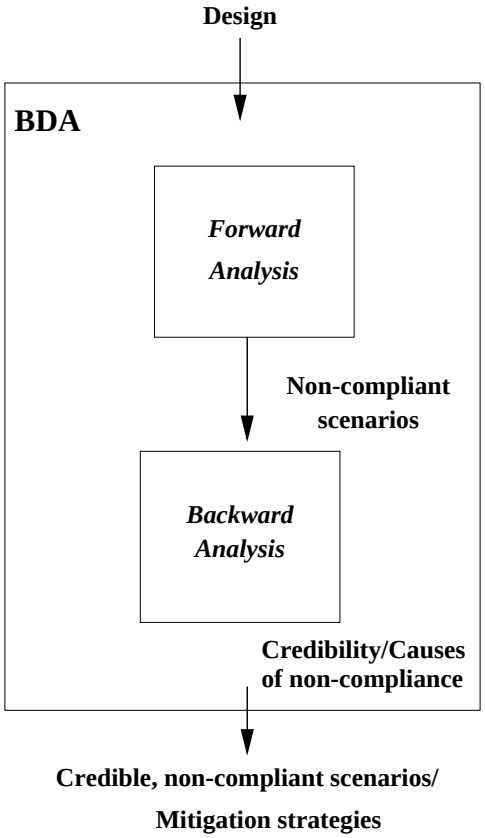

Figure 2: *Bi-directional Analysis (BDA) Procedure*

Figure 2 shows an overview of the two components of a Bi-directional Analysis, the forward analysis and the backward analysis. The forward analysis in BDA has its roots in SFMEA (Software Failure Modes and Effects Analysis), while the backward analysis in BDA has its roots in SFTA (Software Fault Tree Analysis).

Software FMEA is an extension of hardware or system FMEA (Failure Modes and Effects Analysis) and FMECA (FMEA with a criticality rating added to the effects of each failure mode). FMEA and FMECA have been widely used for certification of military and industrial applications since the mid-sixties [24]. Among the earliest standards for the FMEA methodology were NASA's in 1971 and the U.S. Military's in 1974. International standards organizations, British and German standards organizations, and the Society of Automotive Engineers have since also issued FMEA standards. (See [20] for details.) There is no comparable standard for performing Software FMEA, although its use has been well-documented since the 1970's (see, e.g., [10, 33] , and, more recently, the System Safety Society's 1993 System Safety Analysis Handbook [43]).

SFTA is likewise an extension of hardware or system FTA (Fault Tree Analysis), which has been used extensively since the 1960's [16, 36]. Fault tree analysis methods use Boolean logic to break down an undesirable event or situation into the preconditions that led to the root event. Software fault tree analysis [2, 16] adapted the FTA technique to software, using events in the code or detailed design to verify the software logic.

Use of both FMEA and FTA (SFMEA and SFTA) continues to grow, both for system safety assessment and for assurance of compliance. For example, the course that NASA offers on Software System Safety recommends both SFMEA and SFTA, spending one-and-a-half of the four days of the course on these techniques [27]. A U.S. Patent and Trademark Office Pilot Project in 1994 recommended that software FMECA become a routine procedure for the acceptance of newly installed software systems [22].

3 Description of BDA

3.1 Process

As Figure 2 shows, Bi-directional Analysis for software certification consists of a two-step process:

- The first step of the BDA is a forward analysis to identify cause/effect relationships in which anomalous data or software behavior can cause unacceptable effects. The forward analysis is similar to a SFMEA (Software Failure Modes and Effects Analysis). For example, in analysis of one system it was found that if expired data is input to the software, an inappropriate control decision results [19]. The expired data is the cause, and the consequences of the resulting control decision are the effect. The expired data could cause a wrong decision to be made in the software, jeopardizing the system's required functionality. For safety certification, each cause/effect relationship is classified according to its potential impact on safety. For those cause/effect relationships that can potentially adversely affect the system's safety, a second step is performed.

- The second step of the BDA is a backward analysis to examine the feasibility of occurrence for each anomaly that was found in the first step to produce an unacceptable effect. In addition, if the postulated hazard is found to be feasible,

the backward analysis contributes to an understanding of how to mitigate or remove the risk. The backward analysis is somewhat similar to a SFTA (Software Fault Tree Analysis). In the example above, a backward analysis showed that obsolete data could, in fact, be provided to the software (when a particular hardware component failed). The backward analysis established that this failure mode was possible and guided changes to the software to remove this vulnerability to expired data.

The BDA's integration of the forward analysis (for hazardous or non-compliant effects) and the backward analysis (for contributing causes) provides a structured way to gauge the software's robustness to anomalous circumstances. The integrated BDA combines the strength of forward search (identifying latent failure modes) with the strength of backward search (identifying coincident circumstances that allow the hypothesized failure mode to occur). The rest of this section describes the process of performing a BDA in more detail.

3.2 Forward Analysis

The first step in performing a BDA is a forward analysis of the software component requiring certification. A forward analysis begins with a description of the kinds of failures that are of concern. In a message-passing model of a distributed system, two kinds of failures are generally represented: communication failures and process failures [15]. In accordance with this model, two kinds of failures are commonly analyzed for each software component in a forward analysis. These failures are documented in Data Tables (communication failures) and Events Tables (process failures).

A typical forward analysis uses table-based worksheets to capture the relevant information. Data Tables assist in the search for communication failures, including unexpected data dependencies and software interface failures (Table 1). A Data Table evaluates the effect of the software component receiving bad or unexpected data, and the effect of the software component producing bad or unexpected output behavior on the other software that uses that data.

The identification of failure modes is the most difficult part of the forward analysis. To assist in the identification, a classification of failure modes has been developed. This classification is consistent with other current classifications of defects in software (e.g., [3, 40]). (See [19] for a more detailed description of failure modes.)

For each input and each output of the software component, each of the following four failure modes is considered:

- Missing Data (e.g., lost message, data loss due to hardware failure)
- Incorrect Data (e.g., inaccurate data, spurious data)
- Timing of Data (e.g., obsolete data, data arrives too soon for processing)
- Extra Data (e.g., data redundancy, data overflow)

The Data Table lists each possible failure mode, describes its effect for the component under consideration, and classifies the consequences of this failure. For example, in Table 1, the Failure Mode type is "Wrong timing of data" the Failure Description column is, "Sensor input data received is outdated," the Local Effect column is, "Refrigerant pump is erroneously commanded off," and the System Effect column describes the consequence for the system ("Temperature limit is exceeded").

The next column indicates the "Criticality" of the failure mode's effects. In the example, this column is "3" (of 5) indicating a threat to the subsystem but not the system. (A standard 5-part criticality measure reflects the severity of the failure's effects, ranging from "no effect" to "catastrophic effect.") The criticality column has safety implications in that high criticality ratings can be used to indicate the existence of hazards. On the other hand, non-critical effects, even if indicative of design errors, may not have safety implications. Depending on the standard against which compliance is being certified, it may be the case that only items with criticality ratings above some threshold will require further analysis.

The final column in the forward analysis table, "Recommendations," proposes corrective actions to eliminate the non-compliant scenario that has been identified. Often it makes sense to defer filling in this column until the second part of the BDA (the backward analysis to identify contributing causes of the failure mode) has been performed. In all but the simplest failure modes, it is often difficult to propose a meaningful corrective action until a better understanding of the circumstances surrounding the failure mode exist.

The second kind of table used for the forward analysis is an Events Table. An Events Table documents the effect of incorrect behavior or an incorrect event on the component and the system. The Events Table

Failure Mode	*Failure Descrip*	*Local Effect*	*Sys Effect*	*Crit*	*Recomm*
Wrong timing of data	Sensor input data received is outdated	Pump commanded off	Temp limit exceeded	3	Add test of timeliness of data

Table 1: *Excerpt from BDA Forward Analysis*

assists in the search for process failures, including the effects of software that fails to function correctly.

For each event (step in processing), each of the following four failure modes is analysed:

- Halt/Abnormal Termination (e.g., hung or deadlocked at this point)
- Omitted Event (e.g., event does not take place, but execution continues)
- Incorrect Logic (e.g., preconditions are inaccurate; event does not implement intent)
- Timing/Order (e.g., event occurs in wrong order; event occurs too early/late)

The Events Table documents the consequences of these failure modes for the events in the component under review and classifies the criticality of the effects. For example, the Failure Mode in one entry was "Timing/Order", the Failure Description was "Instrument turned on too soon", the Local Effect was "Insufficient power," the System Effect was "Undervoltage occurs," and the Criticality was rated "2" (since in this system, there was software to handle recovery from an undervoltage).

3.3 Backward Analysis

The second step in performing a BDA is the backward analysis. The backward analysis considers just those anomalies identified in the forward analysis as highly-critical (e.g., those with a criticality rating of "4" or "5"). These anomalies or deviations then become the root nodes of the backward analysis. The backward analysis examines the possibility of occurrence of each anomaly that produced a non-compliant effect.

It is important to note that although, in keeping with familiar FMEA notation, the anomalies are often called "failure modes," they may not be incorrect data or behavior, but merely unexpected data or behavior. Since it is the unexpected, as well as the strictly incorrect, behavior that is of concern in safety-critical systems, the BDA often reviews all deviations from anticipated behavior, as well as from required behavior.

The backward analysis is somewhat similar to a software Fault Tree Analysis, with a few key differences. As discussed above, the root node is not necessarily a fault. In addition, a FTA takes an event as its root node, but a BDA's root node may not be an event (e.g., it may flow from the Data Table rather than the Events Table). Another difference is that, as Leveson points out, a backward search technique is a chronological ordering of events over time, whereas each level of a fault tree only shows the same event in more detail [16]. The goals of the backward analysis and SFTA are very similar, though. As Rushby puts it, "The goal of SFTA is to show that a specific software design will not produce system safety failures or, failing that, to determine the environmental conditions that could lead it to cause such a failure [36]."

The backward step of the BDA traces the causes of the root node (e.g., "Obsolete Input Data") backward in time to the contributing circumstances, with each successive level of the fault tree expanding the previous level's nodes. The process ends when further analysis of the bottom-level nodes of the tree is deemed impossible (e.g., atomic events) or unproductive.

In many cases the anomaly described by the root node is found to be infeasible in the existing system. These anomalies are then closed issues, since they cannot contribute to a hazardous situation. In other cases, the anomaly described by the root node is found to be possible if a certain sequence or combination of contributing circumstances and events occur. In that case, the possibility of a hazard needs to be addressed.

3.4 Integration

With the information from the backward analysis, the BDA can then be completed. An entry is made in the final column ("Recommendation") of the Data Table or Events Table for this anomaly. This column identifies a change to the software that can mitigate or avoid the described risk. The column may recommend a change to the design (e.g., in one case, an explicit test of the component's postconditions to

preclude inconsistent user displays) or it may even propose a change to the requirements (e.g. a clearer policy on required handling of double-point or coincident failures) [18].

Recent work in combining forward and backward analyses of safety-critical systems supports the notion that these techniques are complementary [18, 7, 21]. Work has been done to extend the HAZOP approach to a systematic exploration of hypothetical failures, with lists of guidewords or historical failure modes contributing to the hazards analysis in the software under review [23]. Automated tools to assist with portions of these analyses are currently being tested on requirements and design models. [25].

BDA is product-oriented rather than process-oriented in that it can "exercise and stress" the design of the software product [45]. It first checks the effect on the system of corrupted input or abnormal event execution without consideration of the source of error. Once a scenario is identified that leads to non-compliant output or behavior, BDA then traces backward in time to document the contributing causes for possible re-design or test.

4 Evaluation

Bi-directional analysis (BDA) has several advantages that recommend its use as a design certification technique to developers of safety-critical software.

- *Availability.* The techniques BDA is based on are well-documented [11, 16, 19, 33, 38, 43], hence relatively easy to teach and apply. These features make the technique readily available.

- *Structure.* The structured step-by-step procedure of BDA guides implementation, and the techniques involved are familiar to engineers worldwide.

- *Maintainability.* The information developed during application of BDA analysis is broadly accessible since the format is readable, table-based, and can be web-accessible.

- *Safety Assessment.* The role of BDA in the design certification process links clearly with requirements, since it assesses the adequacy of the software design in terms of satisfying the system safety requirements. BDA also provides forward links in the development process, since it prioritizes action items (i.e., recommendations for mitigating actions), prioritizes the hazards it uncovers (via the criticality measure), and provides a critical piece of the hazard analysis for the software design. BDA can also identify test cases in order to exercise each failure mode and confirm that the system reacts safely [28].

- *Incremental/evolutionary development.* BDA analysis products fit into an incremental development process by being updatable for documentation. Initial work also suggests that for product line systems, a BDA of the product family can be partially reused in later instances of that family [18].

- *Independent certification.* BDA can be used for design level certification against documented requirements specifications and has been widely used as a means for independent verification.

- *Systems focus.* BDA is consistent with hardware certification practices (e.g., FMEA and FTA), thus encouraging a systems approach to safety.

- *Tools.* BDA may also be amenable to automation. Automated tools exist for software forward and backward analyses [1, 8, 9, 26, 30], but their capabilities are limited. More powerful tools incorporating forward and backward analyses are currently being developed, e.g., by Safeware Engineering Corporation [32].

BDA, like all analytical methods, has potential disadvantages. Like other incremental failure analysis techniques, BDA is time-consuming and much of it is tedious. BDA depends on the domain knowledge of the analyst and the accuracy of the documentation. In part due to these costs and constraints, BDA is usually performed only on the portions of the software identified as safety-critical. The forward analysis part of BDA, based as it is on SFMEA, is subject to the same criticism of only considering one fault at a time, rather than combinations of events. However, integration with the backward analysis in the BDA has been shown to help isolate combinations of events and circumstances that can lead to hazards.

The use of the forward analysis in conjunction with the backward analysis alleviates BDA's limitations in analyzing the effects of multiple or coincident failures [19]. As with FMEA, SFMEA, FTA, and SFTA, BDA is an important technique for gauging and ensuring the fault tolerance of a system. BDA can support hazards analysis, safety analysis, or reliability analysis processes.

5 BDA for Certification

Certification is a process whereby a certification authority determines if an applicant provides sufficient evidence concerning the means of production of a candidate product and the characteristics of the candidate product so that the requirements of the certifying authority are fulfilled [37]. Certification of software for airborne systems [37, 35] includes establishing a basis for certification between the certifying authority and the applicant (developers) which details the applicable regulations (requirements), as well as any special conditions, and outlines the means by which the developers expect to demonstrate compliance.

The means of compliance are typically specified in standards available from the certification authority. The means of compliance usually involve documenting all aspects of the development process (e.g., hazard analysis), providing the results of testing and other verification techniques, and often using specific analysis techniques during development. BDA, because of its role in elucidating failure modes and their causes, is a hazard analysis technique that conforms to certification standards that require system safety assessments.

The use of BDA for design certification of safety-critical systems allows defects to be discovered and removed early in the development life cycle. Some of the most hazardous and costly defects are those associated with incorrect or missing requirements. Examination of the design prior to coding allows for unforeseen hazards to be exposed, their causes to be investigated, and compliance of the design with safety requirements to be verified. The results of the BDA and its list of recommended mitigations are then used as input to the subsequent stages of the certification process, such as testing [37, 35].

BDA is an adaptation and integration of successful engineering methodologies (FMEA and FTA) to the certification of safety-critical software. BDA assesses the compliance of the software product's design with required system safety features, and identifies feasible, non-compliant scenarios for future re-work or testing. With its emphasis on system response to software anomalies, BDA supports the development and deployment of high-integrity systems.

Acknowledgments

The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Funding was provided under NASA's Code Q Software Program Center Initiative UPN #323-08. Reference herein to any specific commercial product, process, or service by tradename, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

References

[1] Bell, D., L. Cox, S. Jackson and P. Schaefer (1992), "Using Causal Reasoning for Automated Failure Modes & Effects Analysis (FMEA)", *IEEE Proc Annual Reliability and Maintainability Symposium*, pp. 343–353.

[2] Cha, S. S., N. G. Leveson, and T. J. Shimeall (1988), "Safety Verification in Murphy Using Fault Tree Analysis," in *Proc 10th International Conference on Software Engineering*, Apr, Singapore, pp. 377-386.

[3] Chillarege, R., et al. (1992), "Orthogonal Defect Classification–A Concept for In-Process Measurements," *IEEE Transactions on Software Engineering*, 18, 11, 943-956.

[4] Cruz-Neira, C. and R. Lutz (1998), "Using Immersive Virtual Environments for Certification," submitted.

[5] Electronic Industries Association (1983), "System Safety Analysis Techniques," Safety Engineering Bulletin No. 3-A, Engineering Department, Washington, D. C.

[6] Electronic Industries Association (1990) "System Safety Engineering in Software Development," Safety Engineering Bulletin No. 6-A, Washington, D. C.

[7] Fenelon, P. and J. A. McDermid (1993), "An Integrated Toolset for Software Safety Assessment," *Journal of Systems and Software*, 21: July, pp. 279-290.

[8] *FEAT (Failure Environment Analysis Tool)*, NASA Cosmic #MSC-21873.

[9] *FIRM (Failure Identification and Risk Management Tool)*, Lockheed Engineering and Sciences Co., Cosmic.

[10] Fragola, J. R. and J. F. Spahn (1973), "The Software Error Effects Analyis; A Qualitative Design Tool," in *Record, IEEE Symposium on Computer Software Reliability*, IEEE 73CH0741-9C, pp. 90-93.

[11] Hall, F. M., R. A. Paul and W. E. Snow (1983), "Hardware/Software FMECA", *Proc Annual Reliability and Maintainability Symposium*, pp. 320- 327.

[12] Herrman, D. S. (1995), "A methodology for evaluating, comparing, and selecting software safety and

reliability standards," *Proc Tenth Annual Conference on Computer Assurance*, pp. 223–232.

[13] *IEEE Standard Glossary of Software Engineering Terminology* (1990), IEEE Std 610.12-1990. New York: IEEE.

[14] Ippolito, L. M. and D. R. Wallace (1995), "A Study on Hazard Analysis in High Integrity Software Standards and Guidelines," Gaithersburg, MD: U.S. Dept. of Commerce, National Institute of Standards and Technology, NISTIR 5589, http://hissa.ncsl.nist.gov/index-pubs.html.

[15] Lamport, L. and N. Lynch (1990), "Distributed Computing Models and Methods," *Formal Models and Semantics, Vol. B, Handbook of Theoretical Computer Science*, Elsevier.

[16] Leveson, N. (1995), *Safeware, System Safety and Computers*, Addison-Wesley.

[17] Leveson, N., S. S. Cha, and T. J. Shimeall (1991), "Safety Verification of Ada Programs Using Software Fault Trees," *IEEE Software*, July, pp. 48–59.

[18] Lutz, R., G. Helmer, M. Moseman, D. Statezni, and S. Tockey (1998), "Safety Analysis of Requirements for a Product Family,"*Proc Third IEEE International Conference on Requirements Engineering*.

[19] Lutz, R. and R. Woodhouse (1997), "Requirements Analysis Using Forward and Backward Search," *Annals of Software Engineering* , 3, 459–475.

[20] Lutz, R. and R. Woodhouse, "Failure Modes and Effects Analysis," *Encyclopedia of Electrical and Electronics Engineering*, ed. J. Webster, John Wiley and Sons Publishers, to appear.

[21] Maier, T. (1995), "FMEA and FTA To Support Safe Design of Embedded Software in Safety-Critical Systems," in *CSR 12th Annual Workshop on Safety and Reliability of Software Based Systems*, Bruges, Belgium.

[22] Mazur, M. F. (1994), "Software FMECA," *Proc Fifth International Symposium on Software Reliability Engineering*, Monterey, CA, Nov. 6-9.

[23] McDermid, J. A., M. Nicholson, D. J. Pumfrey, and P. Fenelon (1995), "Experience with the application of HAZOP to computer-based systems," in *Proc of COMPASS '95*, Gaithersburg, MD, pp. 37-48.

[24] MIL-STD-882B (1984), "System safety program requirements," U.S. Department of Defense, Washington, D. C.

[25] Modugno, F., N. G. Leveson, J. D. Reese, K. Partridge, and S. D. Sandys (1997), "Integrated Safety Analysis of Requirements Specifications," *Proc Third IEEE International Symposium on Requirements Engineering*, pp. 148–159.

[26] Montgomery, T. A., D. R. Pugh, S. T. Leedham, and S.R. Twitchett (1996), "FMEA Automation for the Complete Design Process", *IEEE Proc Annual Reliability and Maintainability Symposium*, Annapolis, MD, Jan 6-10, pp. 30–36.

[27] NASA Safety Training Center, "Software System Safety Course Notes."

[28] Pfleeger, S. L. (1998), *Software Engineering, Theory and Practice*, Prentice-Hall.

[29] *Procedures for Performing a Failure Mode, Effects and Criticality Analysis* (1980), MIL-STD-1629A.

[30] Pugh, D. R. and N. Snooke (1996), "Dynamic Analysis of Qualitative Circuits for Failure Mode and Effects Analysis," *Proc Annual Reliability and Maintainability Symposium*, pp. 37–42.

[31] Raheja, J. (1991) Assurance Technologies: Principles and Practices, McGraw-Hill.

[32] Ratan, V., K. Partridge, J. Reese and N. Leveson (1996), "Safety analysis tools for requirements specifications," *Proc Eleventh Annual Conference on Computer Assurance*, pp. 149–160.

[33] Reifer, D. J. (1979), "Software Failure Modes and Effects Analysis,"*IEEE Transactions on Reliability*, R-28, 3, 247-249.

[34] Roland, H. E. and B. Moriarty, (1990), *System Safety Engineering and Management*. New York, New York: John Wiley and Sons.

[35] RTCA/DO-178B (1992), *Software Considerations in Airborne Systems and Equipment Certification*, RTCA, Inc.

[36] Rushby, J. (1993), *Formal Methods and Digital Systems Validation for Airborne Systems*, SRI-CSL-93-07.

[37] SAE (1996), *Aerospace Recommended Practice: Certification Considerations for Highly-Integrated or Complex Aircraft Systems*, ARP4754.

[38] SAE (1996), *Aerospace Recommended Practice: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*, ARP4761.

[39] SCOPE Project (1993), ESPRIT2, http://www.cordis.lu/esprit/ src/index.htm

[40] Selby, R. W. and V. R. Basili (1991), "Analyzing Error-Prone System Structure," *IEEE Transactions on Software Engineering*, 17, 2, 141-152.

[41] Stephenson, J. (1991), *System Safety 2000: A practical guide for planning, managing and conducting system safety programs*. New York, New York: Van Nostrand Reinhold.

[42] "Streamlining Software Aspects of Certification," http://shemesh.larc.nasa.gov/ssac/

[43] System Safety Society (1993), *System Safety Analysis Handbook*.

[44] Talbert, N. (1998), "The Cost of COTS: An Interview with John McDermid," Computer, 31, 6, June, 46–52.

[45] Voas, J. (1998), "A Recipe for Certifying High Assurance Software," RST Corp., http://www.rstcorp.com/paper-chrono.html.

[46] Wallace, D. R., L. M. Ippolito and D. R. Kuhn (1992), "High Integrity Software Standards and Guidelines," Gaithersburg, MD: U.S. Department of Commerce, National Institute of Standards and Technology, NIST Special Publication 500-204, September.