

## Failure Assessment

### Abstract:

*Three questions to which software developers want accurate, precise answers are "How can the software system fail?", "What bad things will happen if the software fails?", and "How many failures will the software experience?". Numerous techniques have been devised to answer these questions; three of the best known are:*

- 1) Software Fault Tree Analysis (SFTA)*
- 2) Software Failure Modes, Effects, and Criticality Analysis (SFMECA)*
- 3) Software Fault/Failure Modeling.*

*SFTA and SFMECA have been successfully used to analyze the flight software for a number of robotic planetary exploration missions, including Galileo, Cassini, and Deep Space 1. Given the increasing interest in reusing software components from mission to mission, one of us has developed techniques for reusing the corresponding portions of the SFTA and SFMECA, reducing the effort required to conduct these analyses. SFTA has also been shown to be effective in analyzing the security aspects of software systems; intrusion mechanisms and effects can easily be modeled using these techniques. The Bi-Directional Safety Analysis (BDSA) method combines a forward search (similar to SFMECA) from potential failure modes to their effects, with a backward search (similar to SFTA) from feasible hazards to the contributing causes of each hazard. BDSA offers an efficient way to identify latent failures. Recent work has extended BDSA to product-line applications such as flight-instrumentation displays and developed tool support for the reuse of the failure-analysis artifacts within a product line. BDSA has also been streamlined to support those projects having tight cost and/or schedule constraints for their failure analysis efforts. We discuss lessons learned from practice, describe available tools, and identify some future directions for the topic.*

*A substantial amount of research has been devoted to estimating the number of failures that a software system will experience during test and operations, as well as the number of faults that have been inserted into that system during its development. One of us has found that the amount of structural change to a system during its development is strongly related to the number of faults inserted into it. Using techniques requiring no additional effort on the part of the development organization, the required measurements of structural evolution can be easily obtained from a development effort's configuration management system and readily transformed into an estimate of fault content. So far, structure-fault relationships have been identified for source code; current work seeks to examine artifacts available earlier in the lifecycle to determine if similar relationships between structure and fault content can be found. In particular, relationships between requirements change requests and the number of faults inserted into the implemented system*

would provide a significant improvement in our ability to control software quality during the early development phases.

## **Introduction**

Three questions to which software developers want accurate, precise answers are "How can the software system fail?", "What bad things will happen if the software fails?", and "How many failures will the software experience?". In this paper we discuss several of the most prevalent and useful techniques that have been devised to answer these questions. For each technique, we present its purpose and background, describe the process of performing the technique, and evaluate it in a discussion section. We also discuss lessons learned from practice, describe available tools and resources to help the practitioner select and implement failure assessment techniques, and identify some future directions for the topic.

## **FMEA**

*Purpose:* Failure Modes and Effects Analysis (FMEA) is an engineering process that investigates the potential effects of postulated failures on a system and its environment. When the criticality of the effects is also considered, the technique is called a Failure Modes, Effects and Criticality Analysis (FMECA). FMEA and FMECA are widely used to discover design defects during development of a system and to troubleshoot problems during system operation.

*Background:* FMEA developed in the late 50's and 60's to provide a systematic form of failure analysis that could improve reliability. Its use has been embraced by a broad range of industries including device designers, aerospace, automotive, manufacturing and chemical processing. The earliest organizations to set standards on FMEA were NASA in 1971 and the U.S. military in 1974. See [Lutz99] for more information.

*Process:* The input to FMEA is a design description of the system or component. After determining the scope of the analysis (what components and what level of detail are appropriate), the analyst identifies the failure modes of each block. Depending on the scope of the analysis and type of system, these failure modes may include mechanical, electronic, electrical, software, environmental and/or operational aspects of the system. Often the potential failure modes are drawn from pre-existing libraries of known failure modes for each component. These guidewords (e.g., "rupture" or "short-circuit") provide a structured approach to the investigation.

After determining the failure modes, the analyst then determines the effect of each potential failure mode on the component (local effect) and on the system operating in its environment (global effect). Effects range from effects on the component itself ("leak") to effects on the environment ("contamination").

The FMEA results are recorded in a table with the left-hand columns describing the failure mode and, perhaps, the guideword used to identify it, and the right-hand columns describing the effects and, perhaps, the criticality rankings of the effects. Some FMEAs are quantitative, and may also have probabilities attached to the occurrence of the failure modes and of the effects. The rightmost column of the FMEA may contain a mitigation plan or recommendation for how to avoid/prevent/recover from the failure mode. The output of the FMEA is a set of recommended corrective actions for design improvement and the FMEA tables.

*Discussion:* FMEA is a bottom-up, static-analysis technique with the effect of failure on the individual component first being identified. FMEA is often performed hierarchically with the effects at the lower level typically serving as the failure modes at the next higher level. FMEA is also a forward-analysis technique since it proceeds forward in time from the occurrence of failures to their subsequent effects. The quality of the FMEA depends on the analyst's level of knowledge of the system being studied. Tools and automation are discussed below. A limitation of FMEA is that it usually considers only one failure at a time so may not be suitable for investigation of multiple failures or of common-cause failures, especially those involving subtle timing issues.

## **SFMEA**

*Purpose:* Software Failure Modes and Effects Analysis (SFMEA) is a software-engineering process that investigates the potential effects of postulated software failures on a system and its environment. When the criticality of the effects is also considered, the technique is called a Software Failure Modes, Effects and Criticality Analysis (SFMECA). SFMEA and SFMECA are primarily used to discover software design defects during software development.

*Background:* SFMEA is an extension of FMEA developed in the 70's to provide a systematic form of failure analysis that could improve reliability. It has been used internationally in the development of software for space systems. There are several guidebooks that describe SFMEA but no publicly available standards. See [Herm99, Lutz97, NASA04, Reif79] for more information.

*Process:* SFMEA is a structured, table-based process of discovering and documenting the ways in which a software component can fail and the consequences of those failures. The input to a SFMEA is a specification of the design or detailed requirements of the software. After determining the scope of the analysis (what software components and what level of detail are appropriate), the analyst identifies the failure modes of each block. The SFMEA process is guided by a set of standardized failure modes ("wrong timing of data", "abnormal process termination"). Note that SFMEA is most effective when the failure modes include anomalous software data and events, i.e., deviations from expected software behavior, rather than just faults.

The SFMEA process traces the propagation of anomalies from causes (failure modes) to local (subsystem or component) effects to global (system or environmental) effects. An example of a failure mode is “heater turned on too early”. An example of an effect is “battery allocation exceeded”. The left-hand columns describe the failure modes, possibly in terms of guidewords [Lutz97]; the right-hand columns describe the local and system effects. In a SFMECA a criticality rating (e.g., high, medium or low) is assigned to each failure mode based upon its likelihood of occurrence and the severity of its effects.

We recommend the construction of two types of SFMEA tables in order to consider both communication failures and process failures, a Data Table and an Events Table. The Data Table analyzes failures in software interfaces and data dependencies. This table evaluates both the effect of receiving bad or unexpected input data on software behavior and the effect of producing bad or unexpected output data on the behavior of the components that receive or use this data. The Events Table analyzes failures in the functional execution of the software program. This table describes both the local effect and global effect of performing an incorrect event (“lock door”). What constitutes an event depends on the scope of the analysis and the level of detail of the available documentation. An event is usually considered to be a single action (“perform calculation”, “sample sensor value”, “slew antenna to new position”). See [Lutz97, NASA04] for a fuller description of the process. The rightmost column of the SFMEA often describes software change requests or other corrective actions (e.g., operational flight rules) to mitigate failure modes. The output of the SFMEA is the recommendations for design improvement and the SFMEA Data and Event Tables.

*Discussion:* SFMEA is a bottom-up, forward-analysis technique. SFMEA describes the effects of the hypothesized failure mode as they propagate through the software and the system. As with FMEA, a limitation is that multiple, concurrent failures and common-cause failures may escape detection. Integration of the SFMEA with the backward-analysis technique of Software Fault Tree Analysis has shown some success in overcoming this limitation. That integration is discussed below.

## **FTA**

*Purpose:* Fault Tree Analysis is an engineering activity that investigates the potential causes of a fault or hazard. FTA is widely used to discover design defects during the development of a system and to investigate the causes of accidents or problems that occur during system operation.

*Background:* FTA was developed in the early 1960’s at Bell Telephone Laboratories to analyze the Minuteman Launch Control System]. It is probably the most widely used analysis technique in industry for reasoning about safety and reliability. It has been used for both quantitative analysis and qualitative analysis. See [Leve95, MIL80, Rah90, Stor96] for additional information.

*Process:* The input to FTA is a hazard, failure or accident, and a design description of the system. The analyst works backward from the root-node hazard, using Boolean logic to describe the combination of events or conditions that caused each node. Each result-

ing node is similarly decomposed until events determined to be basic by the analyst (i.e., not meriting further decomposition) are reached at the leaf nodes.

In each fault tree its minimum cut sets are the smallest sets of basic events that cause the root node to occur. These describe all the different ways in which the root-node hazard can happen. The analyst then focuses attention on how best to remove the vulnerabilities in the design that the FTA has identified. By removing paths to the root node, the risk of the hazard occurring is reduced. The output of the FTA is thus the fault trees it produces and the set of recommendations for corrective actions to be taken to preclude the occurrence of the hazard.

*Discussion:* FTA is a top-down technique. Fault trees are sometimes constructed with each successive level refining the understanding of the root event. Other times fault trees are constructed with backward analysis, where each successive level works backward in time. A limitation of this approach is that a fault tree must be created for every top-level hazard that is considered. Another limitation is that the root node must be a known (rather than a latent) hazard. Because basic fault trees do not capture timing information, there have been several extensions to incorporate time [Hans98].

## **SFTA**

*Purpose:* Software Fault Tree Analysis (SFTA) is a static-analysis process that investigates the potential software-related causes of a fault or hazard. SFTA is primarily used software development to discover software defects. It is most efficient when detailed requirements or design documentation exist. It has also been used for verifying software code.

*Background:* SFTA was developed in the 1980's as an application of fault trees to software systems. It has proven to be an intuitively appealing way to structure the analysis of whether an undesirable event could occur and, if so, what events would lead to the occurrence of the undesirable event.

*Process:* SFTA is a structured, tree-based process of discovering and documenting the contributing causes to the occurrence of an undesirable event, represented in the root node. The input to the SFTA is a root-node hazard or event of concern and a specification of the design or detailed requirements of the software. Alternatively, SFTA can be performed on the software code.

The SFTA process systematically traces the events or conditions that could lead to the undesirable root node, documenting them in Boolean logic. An example of a root node for an insulin delivery system is "overdose administered". An example of a high-level contributing cause is "incorrect dosage recorded".

There are two basic gates that are used in SFTA: AND gates and OR gates. Together with the NOT notation, these form the core of the SFTA representation. Different tools

and approaches have different additional gates, such as INHIBIT, XOR, and PRIORITY\_AND (to describe events that must both occur and in a specified order).

*Discussion:* SFTA is a top-down technique. Unlike FTA, a SFTA uses a backward search to find the possible causes of the problem. Note that the basic SFTA evaluates only the possibility of occurrence, not the likelihood. However, developers have often found it useful to annotate the nodes with quantitative information. One such application is in security to detect how intrusions can occur. The SFTA there is called an “attack tree” because the root node describes a security attack on the system. As with FTA, a limitation of SFTA is that the root node can only describe a known failure.

## **BDSA**

*Purpose:* BDSA integrates the forward analysis performed in SFMEA and the backward analysis performed in SFTA to provide a more complete static analysis of the design. The purpose of BDSA is to show that the software design is free of critical flaws that can contribute to hazards. It is a systematic technique for identifying what can go wrong with each software component in the system (its failure modes), what effects each failure mode can have as it propagates through the system, and what events or features enable or contribute to the possibility of that failure mode in the first place. The combination of the forward and backward search has proven effective in discovering latent safety requirements [Lutz97].

*Background:* Combining forward analysis and backward analysis grew out of the HAZOP approach used in the chemical industry to provide guideword-driven analysis of the effects and causes of deviations from the specified process. Beginning in the 1990's this approach was applied to software. Whereas we perform the forward search first (for high-risk effects) and then the backward search (for causes of these failures), some users first perform a backward search on known failures and then a forward search to identify the effects.

*Process:* BDSA initially checks the design to determine whether the effects of abnormal input values and unexpected software events can contribute to unsafe system behavior. The forward analysis is similar to a SFMECA in that it moves from failure modes to effects. Once the failure modes with unacceptable consequences have been identified, the backward analysis is performed to check whether the failure mode could occur in the given system. This analysis determines that feasibility of the failure and the design vulnerabilities that together can lead to the failure of concern. The second part of the BDSA is similar to a SFTA.

*Discussion:* The integration of the forward search entailed in an SFMEA with the backward search of a SFTA enhances the effectiveness of the investigation. In previous work we have used BDSA to find previously unidentified software failure modes, multiple, co-incident anomalies, and hidden dependencies among software processes.

More recently BDSA has been extended by one of the authors to product lines. A product line is a set of systems that share many features as well as a mission. Product lines exploit reuse potential to build new systems with lower cost and, hopefully, higher reliability. The effort is to reuse portions of the safety analysis performed when the product line is specified when new products in the product line are developed. By exploiting the similarities among the systems (and, thus, between their safety analyses) without discounting the effects of the variations among the individual systems, we can improve safety analysis of the product line while reducing the associated costs.

## **Software Reliability Engineering**

*Purpose:* Software Reliability Engineering is an engineering discipline that 1) seeks to improve the reliability of fielded software systems and 2) is used to measure, estimate, and forecast the reliability of software systems during various stages of development. It is this second aspect of software reliability engineering we discuss in the following paragraphs.

*Background:* Software reliability models that could be used to estimate and forecast the reliability of software systems during test and operations were initially developed in the early 1970s [Mor71, Shoo72]. Since that time, dozens of models have appeared in the literature – the most widely used are described in [Lyu96]. These statistical models use a software system's observed failure history (i.e., time since the last failure or number of failures observed in an interval of given length) to estimate the current reliability and forecast the reliability during future testing or operations. Detailed information on the use of these models is given in [AIAA93], [Lyu96], [Musa87], and [Musa04]. More recently, researchers have developed software defect models that relate measurable structural characteristics of software systems and/or development process to the number of defects inserted during development [Mun91], [Mun02], [Nik03]. [Nik04], [Neu92]. Most of these defect models use measurable attributes of source code as their input, although there has been some recent work in relating measurable characteristics of requirements change requests to defect content [Schn01].

*Process:* After unit testing, software reliability models can be used to better manage testing resources. To use software reliability models during test, the failure history of the system under test must be recorded – failure history can be in the form of 1) elapsed time since the last failure, or 2) number of failures in a test interval of given length (“interval data”). A software reliability model applied to this type of data will produce estimates and forecasts of reliability as well as reliability-related quantities (e.g., time to next failure, expected number of failures in the next N intervals). If a testable reliability requirement has been established, a software reliability model can be used to answer the following questions:

1. Has the software achieved the required reliability (to a specified level of confidence)?
2. How much more testing effort will be required to achieve the required reliability? This result can be readily transformed into estimates of the number of testing resources (e.g., personnel, test installations, funding) that will be required.

3. What will the impact on the system's reliability be if only a certain fraction of the required testing resources will be available?

It must be noted that it does not seem possible to determine a priori which software reliability will be most applicable to a software development effort [Abdel86, Nik95]. Fortunately, a number of statistical techniques have been developed that can help practitioners identify the most appropriate software reliability model at any point during the testing phase. Therefore, it is recommended that multiple software reliability models be applied to a set of failure data, and that model predictions be updated at regular intervals (e.g., weekly), since model preferences can change rapidly during test.

During earlier development phases, defect models that estimate the software's defect content based on its structural characteristics can be used to identify those components having a higher fault burden, and hence posing a higher reliability risk, than others. Tools for taking and analyzing the necessary measurements can be inserted into a development effort (e.g., as part of the configuration management process) without requiring additional effort on the part of the developers [Nik01]. Developers and managers can then examine the results to identify those portions of the software having the greatest defect potential; these results can be used in conjunction with other information, such as the criticality of a component, to help development organizations better decide how to allocate scarce defect identification and removal resources.

*Discussion:* Software reliability estimation and forecasting can be either black-box or white-box activities. "Traditional" software reliability models (i.e., those that operate on failure history data obtained during test) do not rely on any measurable characteristics of the software system's structure; they cannot be used to predict the effect of any design and/or development process changes on the software system's reliability. Defect models that can be applied during earlier portions of the development life cycle, on the other hand, are white-box activities, since they rely on measurable characteristics of a software system's structure and/or development process to estimate the software's defect content. If proposed changes to the software system's structure can be measured, these types of models may be used to estimate the effect of the changes on the software's defect content. Unlike traditional models, however, these models cannot be used to directly estimate or predict reliability and reliability-related quantities such as failure rates or the expected number of defects in future test intervals. However, since they can be applied earlier than traditional software reliability models, they can be used to identify defect-prone software components earlier in the development cycle, potentially reducing defect identification and removal costs during later phases.

## **Tools and Automation**

Automated and partially-automated toolsets can reduce the cost and labor-intensive of performing fault analysis. There are a number of commercial and government toolsets that assist with FMEA and FTA, such as Sapphire [Saph], Relex [Relx], and Galileo [Sul99]. We do not endorse any specific products here. Such toolsets often reduce repetitive data entry, especially during updates, provide links to libraries of component models, integrate with CAD (Computer-Aided Design) tools, calculate failure rates and

other statistics, and support report-writing. These same toolsets also support SFMEA and SFTA. However, since the quantitative analysis of software is still an open problem, the automation is primarily used for editing, configuration control, and reuse of components (e.g., of recurring sub-trees). Thus, some analysts will prefer the use of more familiar word-processing or drawing programs when performing SFMEA.

A number of software reliability modeling tools implementing traditional software models have been developed over the past 10 years. The most popular of these tools are Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) [SMERFS], and Computer-Aided Software Reliability Estimation (CASRE) [CASRE]. Both of these tools implement a number of the more popular models, allowing practitioners to apply traditional models according to the process discussion in the preceding discussion on software reliability engineering. Practitioners can use either of the two types of failure history described in the section on software reliability engineering, as each of the tools implements both types of models. In addition to displaying estimates and forecasts of software reliability and reliability-related quantities in graphical and tabular form, the tools allow user to determine model applicability using goodness of fit tests (Kolmogorov-Smirnov for times between failures data, and Chi-Square for interval data) as well as the analyses described in [Abdel86].

## **Future Directions**

Advances in failure assessment continue to be made. Three directions are of particular interest to ISHEM.

- Automated generation of FTA/SFTA and FMECA/SFMECA. Progress in model-based development is enabling automatic production of fault trees and other failure analysis artifacts from state-based models. Results to date are preliminary. However, we expect the quality and scalability of these automated techniques to improve rapidly and move into industry in the next five years.
- Product-line fault trees. As industry and NASA move toward development of product lines of similar systems, interest in product-line approaches to failure assessment has grown. Recent results extend SFTA and SFMECA to product lines [Dehl04, Dehl06]. Production of these product-line artifacts take place at the time of product-line specification or design analysis. Subsequently, as each new system in the product line is developed, the analyst, with automated tool support, prunes the product-line software fault tree to consider only those nodes relevant to that particular system. In this way, the failure-analysis products can be efficiently reused across the systems in a product line. Similarly, in the future, libraries of SFTA subtrees and SFMECA for reusable software components will be assembled and made available to the developer, much as libraries for hardware components are today.
- In software reliability engineering, more work is being done in estimating a software system's defect content earlier in the life cycle. More researchers are attempting to identify relationships between measurable attributes of specifications and the defect content of the implemented system. A good deal of work has also been done in developing software reliability models using architectural informa-

tion about a system [Gos01], [Gos01a], [Gokh98]. These models can help developers identify architectural components whose correct operation is the most critical to the overall reliability of the system; experiments reported in [Gos05] indicate that a relatively small number of components have the greatest effect on the overall reliability. Recent work also indicates that relatively non-intrusive instrumentation can be developed that will allow practitioners to assess the risk of exposure to residual faults in near-real time [Nik03a]. If this type of instrumentation can be developed, the resulting risk assessment might be used as the basis of adaptive software fault tolerance strategies.

## References

- [Abdel86] A. Abdel-Ghaly, P. Y. Chan, and B. Littlewood; "Evaluation of Competing Software Reliability Predictions," *IEEE Transactions on Software Engineering*; vol. SE-12, pp. 950-967; Sep. 1986
- [AIAA93] ANSI/AIAA, R-013-1992, "Recommended Practice for Software Reliability", American National Standards Institute/American Institute of Aeronautics and Astronautics, 370 L'Enfant Promenade, SW, Washington, DC 20024, February 23, 1993
- [CASRE] [http://www.openchannelfoundation.org/projects/CASRE\\_3.0](http://www.openchannelfoundation.org/projects/CASRE_3.0)
- [Dehl04] J. Dehlinger and R. Lutz, "Software Fault Tree Analysis for Product Lines," *Proc. 8th IEEE International Symposium on High Assurance Systems Engineering (HASE'04)*, March 24-26, 2004, Tampa, Florida, pp. 12-21.
- [Dehl06] J. Dehlinger and R. Lutz, "PLFaultCat: A Product-Line Software Fault Tree Analysis Tool", with J. Dehlinger, *Automated Software Engineering*, to appear.
- [Feng05] Q. Feng and R. Lutz, "Bi-Directional Safety Analysis of Product Lines", *Journal of Systems and Software*, to appear.
- [Gokh98] S. Gokhale, M. Lyu, and K. S. Trivedi, "Reliability Simulation of Component-Based Software Systems", *International Symposium on Software Reliability Engineering, Paderborn, Germany*, Nov. 1998
- [Gos01] K.Goseva-Popstojanova, K.S.Trivedi, "Architecture Based Approach to Reliability Assessment of Software Systems", *Performance Evaluation*, Vol.45/2-3, June 2001
- [Gos01a] K.Goseva-Popstojanova, A.P.Mathur, K.S.Trivedi, "Comparison of Architecture-Based Software Reliability Models", *Proc. 12th IEEE International Symposium on Software Reliability Engineering (ISSRE 2001)*, Hong Kong, Nov. 2001
- [Gos05] K.Goseva-Popstojanova, "Performability of Web Based Applications", *proceedings of the NASA Office of Safety and Mission Assurance Software Assurance Symposium*, Aug. 9-11, 2005.

- [Hans98] K. Hansen, A. Ravn, and V. Stavridou, "From Safety Analysis to Software Requirements", *IEEE Trans. Softw. Eng.* 24, 7 (Jul. 1998), 573-584
- [Helm02] G. Helmer, J. Wong, M. Slagell, V. Honavar, R. Lutz and L. Miller "A Software Fault Tree Approach to Requirements Analysis of an Intrusion Detection System," *Requirements Engineering Journal*, Vol. 7, Issue 4 (2002), pp. 207-220.
- [Herm99] Hermann, Debra S. *Software Safety and Reliability*. IEEE Computer Society, 1999.
- [Leve95] Leveson, Nancy, *Safeware: System Safety and Computers*. Addison-Wesley, 1995.
- [Lutz97] R. Lutz and R. Woodhouse "Requirements Analysis Using Forward and Backward Search," *Annals of Software Engineering*, Vol. 3, Sept, 1997, pp. 459-475.
- [Lutz99] R. Lutz and R. Woodhouse "Failure Modes and Effects Analysis," with R. Woodhouse, in *Encyclopedia of Electrical and Electronics Engineering*, ed. J. Webster, John Wiley and Sons Publishers, 1999, Vol. 7, pp. 253-257.
- [Lyu96] M. Lyu, ed., Handbook of Software Reliability Engineering, McGraw-Hill, 1996
- [MIL80] *Military Standard, Procedures for Performing a Failure Mode, Effects and Criticality Analysis* (1980), MIL-STD-1629A.
- [Mor71] P. Moranda, Z. Jelinski, "Software Reliability Research", McDonnell-Douglas Astronautics Co., MDAC Paper WD1808, Nov 1971.
- [Mun02] J. Munson, A. Nikora, "Toward a Quantitative Definition of Software Faults", *proceedings of the International Symposium on Software Reliability Engineering*, Nov 12-15, 2002, Annapolis, MD
- [Mun91] J. Munson, T. Khoshgoftaar, "The Use of Software Complexity Metrics in Software Reliability Modeling", *International Symposium on Software Reliability Engineering*, May 17-18, 1991
- [Musa04] J. Musa, Software Reliability Engineering: More Reliable Software Faster And Cheaper (2<sup>nd</sup> ed), Authorhouse, 2004
- [Musa87] John D. Musa., Anthony Iannino, Kazuhiro Okumoto, Software Reliability: Measurement, Prediction, Application; McGraw-Hill, 1987
- [NASA04] *NASA Software Safety Standard*, NASA-STD 8719-13B, July, 2004.
- [Neu92] A. Neufelder, Ensuring Software Reliability, Marcel Dekker, 1992
- [Nik01] A. Nikora, J. Munson, "A Practical Software Fault Measurement and Estimation Framework", *proceedings of the Industrial Presentations track of the 12<sup>th</sup> International Symposium on Software Reliability Engineering*, Hong Kong, Nov 27-30, 2001

- [Nik03] A. Nikora, J. Munson, "Developing Fault Predictors for Evolving Software Systems", *proceedings of the 9<sup>th</sup> International Software Metrics Symposium*, Sep 3-5, Sydney, Australia
- [Nik03a] A. Nikora, J. Munson, "Understanding the Nature of Software Evolution", *proceedings of the International Conference on Software Maintenance*, Sep 22-26, Amsterdam, The Netherlands
- [Nik04] A. Nikora, J. Munson, "The Effects of Fault Counting Methods on Fault Model Quality", *proceedings of the 28<sup>th</sup> Annual International Computer Software and Applications Conference (COMPSAC2004)*, Hong Kong, Sep 28-30, 2004
- [Nik95] A. Nikora, M. R. Lyu, "An Experiment in Determining Software Reliability Model Applicability", *proceedings of the Sixth International Symposium on Software Reliability Engineering, Toulouse, France*, October 24-27, 1995
- [Rah90] D. Raheja, *Assurance Technologies, Principles and Practices* McGraw-Hill, 1990.
- [Reif79] Reifer, D. J., "Software Failure Modes and Effects Analysis", *IEEE Trans on Reliability*, R-28, 3, 247-249.
- [Relx] <http://www.relexsoftware.com/>
- [Saph] <http://saphire.inel.gov/>
- [Schn01] N. Schneidewind, "Investigation of the Risk to Software Reliability and Maintainability of Requirements Changes", *proceedings of International Conference on Software Maintenance*, Nov 7-9, 2001, Florence, Italy, pp. 127-137
- [Shoo72] M. Shooman, "Probabilistic Models for Software Reliability Prediction", *Conference on Statistical Methods for the Evaluation of Computer Systems Performance*, Brown University, November 22-23, 1971. Published in Probabilistic Models for Software, Freiberger, Editor, Academic Press, New York, 1972.
- [SMERFS] <http://www.slingcode.com/smerfs>
- [Stor96] N. Storey, *Safety-Critical Computer Systems*. Addison-Wesley, 1996.
- [Sul99] K. J. Sullivan, J. B. Dugan, and D. Coppit, "Developing A High-Quality Software Tool For Fault Tree Analysis", *Proceedings of the International Symposium on Software Reliability Engineering*, pages 222-31, Boca Raton, Florida, 1999.