Protection against Software Failures: Low Failure Rate and Fault Tolerance

Herbert Hecht
SoHaR Incorporated
Culver City CA 90230
herb@sohar.com

Abstract – In some environments the mitigation of high severity software failures is conditioned on the probability of failure but in others protective measures are required regardless of the reliability assessment. The paper investigates how this difference results in dissimilar approaches to the verification of the programs. The regulatory basis for these differences is examined. It is shown that new verification tools simplify the verification, particularly where evidence of program reliability is not accepted.

1. Introduction

In evaluating the ability of a computer program to service critical applications the user (frequently represented by a regulatory agency or professional organization) may require demonstration of its reliability or demand protection against severe failure modes by monitoring or fault tolerance. Reliability demonstration is based on the premise that failure probability (or failure frequency) is a fixed quality of a computer program. If we know this probability we can assess the risk associated with failure, and therefore we refer to this also as the *risk* approach. In other areas it is held that probability of failure at very low levels is at present impossible to assess, and that demonstration of the integrity of the software controlled system requires protective measures such as fault tolerance. This is referred to as the *integrity* methodology. The two approaches are not inherently mutually exclusive but the shift in emphasis can require very different approaches to software verification. It will be shown that reliability demonstration is primarily used where failures, most with limited consequences are routinely observed (e. g., communication networks and some automotive applications) while areas where failures are rarely observed but can cause multiple casualties emphasize fault tolerance (e. g., aircraft or nuclear safety systems).

2. Requirements for Software Verification

Where failure of a system is likely to have dangerous or severe consequences the user will want assurance that the developer or vendor has taken adequate precautions to either prevent the occurrence of such failures or to mitigate their effects. As will be seen below, the government or trade organizations have issued regulations that define levels of risk and in some cases prescribe mitigating measures. In hardware systems compliance with these regulations is routinely achieved through testing, and the procedures used in this testing have themselves become subject to regulation or standardization by professional organizations such as ASTM International.

Because it is manifestly impossible to test every possible state or state sequence of a large computer controlled system it has long been realized that most of the effort needs to be directed at segments that are most likely to cause severe system failures. The documents referred to below examine risk at the system level and most of them were originally targeted at hardware but can either be interpreted to software components or have been specifically modified to include software risks. Software verification is thus focused to demonstrate compliance with these governing regulations or documents. It will be convenient to discuss these regulations under the headings of *risk* and *integrity* although the two share some features.

3. The Risk Approach

The risk approach is exemplified by the Risk Assessment requirements (par. 4.5) of MIL-STD-882D[1] that define risk as a combination of hazard severity and hazard probability. Suggested risk assessment values is shown in Table 1 which is taken from Appendix par. 4.4 of the standard. Low numbers indicate high risk.

Table 1 Risk Assessment Values (RAV)

Probability	Severity Category			
	I	II	III	IV
Frequent	1	3	7	13
Probable	2	5	9	16
Occasional	4	6	11	18
Remote	8	10	14	19
Improbable	12	15	17	20

One use of the risk assessment values is to determine the level of authority for acceptance of a risk. The standard suggests the following levels (Table A-IV)[3].

For RAV 1-5: Component Acquisition Executive
For RAV 6 – 9: Program Executive Officer
For RAV 10 – 17: Program Manager
For RAV 18 – 20: As Directed

The designated authority will most probably want to know what can be done to reduce or eliminate the risk. When proposals for these improvements are presented either resources for reliability improvement will be allocated or, very seldom, the authority will accept the risk. By this interpretation the risk assessment values are a guide to the amount of resources that can be expended for risk reduction. We will now examine some details of the application of Table 1.

An abbreviated explanation of possible consequences associated with the severity categories, taken from Table A-I of the standard is

I – *Catastrophic* –death or loss exceeding $1 million
II – *Critical* –permanent disability or loss exceeding $200,000
III – *Marginal* – lost work injury or loss exceeding $10,000
IV – *Negligible* – injury without lost work or loss exceeding $2,000.

An abbreviated explanation of mishap probabilities, taken from Table A-II of the standard, is presented in the following. The numbers listed represent the probability that a mishap will be observed *during the life of an item.* For software this poses the need for interpretation as to the "life" of a program. Is it the time to the next revision or is it the time to a substantial rewrite or complete replacement? A reasonable assumption is that the life of a program ends when a new certification is required.

Frequent – Greater than 0.1
Probable – Between 0.01 and 0.1
Occasional – Between 0.001 and 0.01
Remote – Between 10^{-6} and 0.001
Improbable - Less than 10^{-6}

Suppose a program has a failure rate of 10^{-6} per hour, is used 2,000 hours per year, and has an expected life (to the next certification) of 10 years. The mishap probability computed by these guidelines is 0.02 which puts it into the "probable" classification. Going back to Table 1 we see that this places it in the highest risk category if it can cause severity I or II consequences.

A very similar approach is taken in the Criticality Analysis (Task 102) of the currently inactive MIL-STD-1629A[2], except that the term criticality" is used instead of "risk". Figure 1 shows the concept of criticality assignment taken in that standard. A related methodology is also followed in a recent SAE standard[3] that is targeted at the automotive industry, It considers lack of testability or observability as an additional risk factor. Failures in the automotive environment can have high severity consequences, including fatalities. But they do not have the potential for multiple fatalities associated with aircraft accidents or those involving nuclear installations.

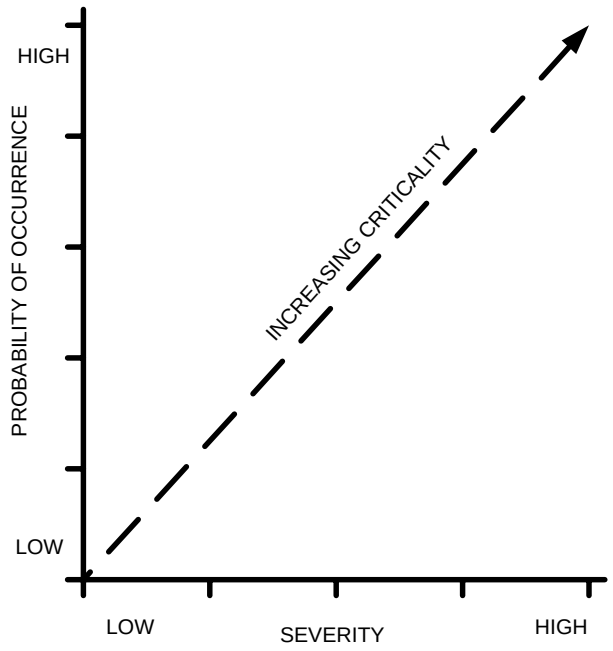

Figure 1 Criticality Assignment

The measurement and estimation of software reliability were extensively studied by a group at Bell Laboratories and associated academic personnel[4 5 6]. These researchers were primarily concerned with software running in the computer based switching systems, programs that underwent a lengthy period of testing during which faults were continuously removed as they were being found. As faults were being removed, the interval to detection of another fault increased, and this gave rise to fault depletion or reliability growth models such as the one shown in Figure 2.

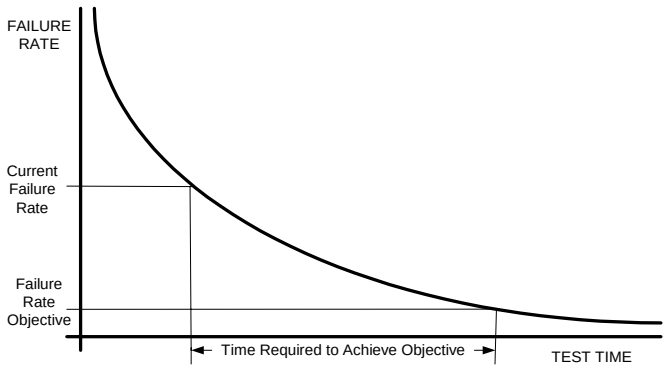

Figure 2 Generic Fault Depletion Model[7]

A number of alternatives for the shape of the curve and for the measurements required to estimate the parameters are provided in the AIAA Recommended Practice from which the figure was adapted. Software for implementing several of these is available[8]. However, some inherent limitations of this approach are evident from an examination of Figure 2.

The first of these is that as the failure rate objective approaches zero the objective line becomes coincident with the asymptote of the curve. The location of the intercept that defines the end of the required test period is therefore very difficult to determine. It is obvious from the curve that a close approach to zero failure rate will necessitate a large extension of the test time and also uncertainty of the test time that will be required to achieve the objective. Keep in mind the hypothetical 10^{-6} failures per hour attribute that was mentioned earlier.

Further, knowledge of the "current failure rate" will always be subject to error. It can be based on the reciprocal of the interval between the last preceding failure and the last one, using a single sample and thus subject to large statistical uncertainty. Or it can be derived from an ensemble of failures that were observed over a given time interval, possibly accounting for the change in expectation by modeling the shape of the curve. While increasing the sample size, this makes the failure rate estimation subject to errors in the knowledge of the curve parameters.

A very significant difficulty arises from the inherent assumption in most of the models that the fault population is homogenous so that the time to encountering a fault is inversely proportional to (or nearly so) the number of remaining faults. There is considerable evidence that this assumption does not hold at very low failure rate levels, the ones that are of most interest for critical applications [9 10]. Models have been proposed to account for the diversity in exposure time to find faults, but the test time required to validate the models and estimate parameters is very much greater than those for the homogenous models.

In summary, estimation of the failure probability that is required for numerical risk assessment requires large sample sizes (long test times). It works best when the failure rate objective can be set high which implies that the failure effects must be less than catastrophic or critical (Table1). As we have indicated, the quantitative risk approach had its origin in the communication industry where outages caused by software failures can cause significant loss of revenue but where there are no

safety issues. Communication and consumer services industries meet these requirements, and these fields usually also have large databases that permit model refinement and facilitate estimation of model parameters.

4. Integrity Approach

The integrity approach emphasizes prevention of conditions that can cause high severity consequences. The prevention can take the form of redundancy or back-ups, monitoring of precursor conditions and subsequent limiting or disengagement of the offending software controlled function, or warnings to the operators to take corrective action. A typical integrity approach is represented by the requirements for protection systems for nuclear power plants that make no mention of the probability of failure:

Redundancy and independence designed into the protection system shall be sufficient to assure that (1) no single failure results in loss of the protection function and (2) removal from service of any component or channel does not result in loss of the minimum redundancy unless the acceptable reliability of the protection system can be otherwise demonstrated"[11].

Similar requirements are specified for safety systems in the process industry [12]. Other critical systems may state requirements under the acronym FOFOFS, meaning fail-operative under a first and second failure and then fail safe. Of particular interest is the development of integrity based requirements for software functions in aircraft systems. These originate with FAR* 25.1309 which states in part:

(a) The equipment, systems, and installations whose functioning is required by this subchapter, must be designed to ensure that they perform their intended functions under any foreseeable operating condition.

(b) The airplane systems and associated components, considered separately and in relation to other systems, must be designed so that--

(1) The occurrence of any failure condition which would prevent the continued safe flight and landing of the airplane is extremely improbable, and

(2) The occurrence of any other failure conditions which would reduce the capability of the airplane or the ability of the crew to cope with adverse operating conditions is improbable.

These tersely stated requirements were are expanded in an advisory circular [13], AC 25.1309-1A, that includes a definition of failure conditions as catastrophic, major, minor and negligible, essentially similar (except for terminology) with those shown in Table 1. It also contains a figure that is conceptually similar to Figure 1 above. Further, it provides quantitative guidance for the qualitative probability of occurrence terms in the FAR (all numbers are per flight hour):

Probable failure conditions are those having a probability greater than 10^{-5}
Improbable failure conditions are those having a probability between 10^{-9} and 10^{-5}
Extremely improbable failure conditions are those having a probability of less than 10^{-9}

This may appear to signify a tendency toward the risk based approach except that the targets for the *improbable* and *extremely improbable* categories are set so high that direct verification is impracticable for software (for hardware appropriate safety margins can be used to demonstrate compliance). Moreover, a very significant feature of the advisory circular is that it addresses software failures in Section 7.i which starts:

In general, the means of compliance described in this AC are not directly applicable to software assessments because it is not possible to assess the number and kinds of software errors, if any, that remain after the completion of system design, development and test.

The section then refers to further guidance for demonstrating compliance of software controlled functions with the intent of FAR 25.1309. Currently the most important one of these guidance documents is RTCA DO-178B[14] which states (par. 12.3.4)

In the preparation of this document, methods for estimating post-verification probabilities of software errors were examined. The goal was to develop

* Federal Aviation Regulation

numerical requirements for such probabilities for software in computer-based airborne systems or equipment. The conclusion reached, however, was that currently available methods do not provide results in which confidence can be placed to the level required for this purpose. Hence this document does not provide guidance for software error rates.

Thus verification of software for aircraft systems is largely tied to the integrity approach. In essence this means identifying

1. the most critical effect on aircraft performance and safety that could be caused by anomalous execution of a software segment
2. means for detecting such anomalous behavior before it causes irrecoverable effects at the system level
3. recovery mechanism(s) for restoring the system or aircraft to a condition that permits safe continuation of the flight and landing

The acceptance of the software it is thus dependent on identification of the worst failure modes and on demonstration of the adequacy of detection and recovery provisions. An established mechanism for systematic review of detection and recovery provisions oriented toward specific failure modes is the Failure Modes and Effects Analysis (FMEA). In this connection it is significant that the FMEA as identified in Task 101 of MIL-STD-1629A[2] does not encompass failure rates or failure probabilities. A summary of the required fields is shown in Table 3.

Table 3. Required Fields in MIL-STD-1629A Task 101.

	FMEA Field
1	Identification No.
2	Item
3	Function
4	Failure Mode
5	Mission Phase
6	Failure Effect – Local
7	Failure Effect – Next Higher
8	Failure Effect – System
9	Failure Detection Method
10	Compensating Provisions
11	Severity Class
12	Remarks

Failure effects are addressed at three levels: local, next higher, and system. In most cases the local effects will be a failure at a processor (failure to execute, wrong output, etc) that can be detected by timers and assertions at the input of the following stage. The next higher level is typically an LRU where effects might be failure of outputs, out-of-sequence outputs, and wrong numerics. Again, these can be detected at the receiving end by suitable assertions. At the system level the effects may be failure to respond to pilot commands, entering an anomalous plant state in a nuclear installation, or a trajectory deviation in a launch vehicle. These are detectable by sensors that are independent of the malfunctioning system. In general local level detection is much more effective than detection at subsequent levels because it prevents propagation of faulty data and provides the best diagnostic for future prevention of the problem. But all levels of detection that can be brought to bear on a failure mode should be listed.

Compensating provisions or recovery mechanisms can also exist at several levels. Usually they take the form of retry, switching to a back-up function, or a work-around that avoids use of data from a suspect source. Again, initiating recovery at a low level is usually preferred because it keeps most of the system operating in a normal manner. However, there are situations in which a processor failure detected at the local level is compensated for by switching to a stand-by at the LRU level because it was not feasible to provide switching or redundancy at the local environment.

A properly completed FMEA is a major step towards acceptance of a software product in under the integrity based approach. and there have been several suggestions for adapting the originally hardware oriented FMEA format for software [15 16]. The approach has been either to assign failure modes to software functional blocks or to consider specific effects due to faulty processing of one input variable at a time[18].

The main disadvantage of the functional approach is that function boundaries are not formally defined. In previous work we have shown that exception handling is a very frequent cause of failures[19], and not every analyst will consider an exception handling routine as a function. Thus the visibility over exception handling tends to get lost in functional partitioning. Similarly, because the need for exception handling frequently arises from conditions on more than one variable, the other approach has obvious limitations as well. The

advent of UML has permitted computer-aided generation of software FMEAs, making the process not only faster but also much more consistent [20]. Currently this methodology is being transferred to MATLAB-Simulink based software development. These developments are expected to permit more consistent and wide spread us of the integrity approach.

5. Conclusions

We have seen that there is a distinction in certification requirements between environments where the worst failure effects are short of devastating and those where they are not. The former can accept the risk-based approach in which software reliability is demonstrated by test or other means to have such low failure probability as to be acceptable for intended use. This group includes some automotive applications that are safety critical.

The other environments comprise those where the consequences of failure are so severe (multiple fatalities) that the level of reliability that can reasonably be demonstrated is not sufficient, and the emphasis is on detection and recovery from failures. A systematic review of detection and recovery provisions for each critical failure mode is facilitated when a software FMEA has been prepared. Tools for efficient and consistent generation of software FMEA are currently being developed.

References

[1] Department of Defense, "Military Standard, System Safety Program Requirements" January 1993 AMSC No. F6861

[2] Department of Defense, "Military Standard, Procedures for Performing Failure Mode, Effects and Criticality Analysis", MIL-STD-1629A, November 1980, AMSC No. N3074

[3] Society of Automotive Engineers, SAE J Standard 1739, " Potential Failure Mode and Effects Analysis in Design (Design FMEA) and Potential Failure Mode and Effects Analysis in Manufacturing and Assembly Processes (Process FMEA) and Effects Analysis for Machinery (Machinery FMEA)," August 2002.

[4] Musa, John D., "A Theory of Software Reliability and its Applications", *IEEE Transactions on Software Engineering*, SE-1 No. 3 September 1975

[5] Shooman, Martin L ., "Structural Models for Reliability Prediction", *Proc. Second National Conference on Software Reliability*, San Francisco, October 1976.

[6] Musa, John D., A. Iannino and K. Okumoto, *Software Reliability*, McGraw Hill, 1987

[7] *Recommended Practice, Software Reliability*, ANSI/AIAA R-013-1992

[8] Farr, W. H., and O.Smith, *"Statistical Modeling and Estimation Functions for Software (SMERFS) – Users Guide"* NSWCDD TR84-371, Revision 3, September 1993

[9] Lttlewood, B. and J. L. Verrall, "A Bayesian Reliability Model with a Stochastically Monotone Failure Rate", *IEEE Transactions on Reliability*, June 1974

[10] Fenton, N. and B. Littlewood, "Limits to Evaluation of Software Dependability", *Software Reliability and Metrics*, Elsevier Applied Science, 1990

[11] Code of Federal Regulations, 10CFR50, App. A, Criterion 21.

[12] The Instrumentation, Systems and Automation Society, "Safety Instrumented Systems for the Process Industry, Part 3: Guidance for the Determination of the Required Safety Integrity Levels –Informative, 02 Sept 2004, ISBN 15561792001

[13] Federal Aviation Administration, Advisory Circular AC 25.1309-1A "System Design and Analysis", 6/21/88. This circular has now been adopted by international bodies.

[14] RTCA DO-178B, "Software Considerations in Airborne Systems and Equipment Certification", December 1992

[15] Fragola, J. R. and Spahn, J. F., "The Software Error Effects Analysis - A Qualitative Design Tool, 1973 IEEE Symposium on Computer Software Reliability, New York, NY

[16] Reifer, Donald J., "Software Failure Mode and Effects Analysis", *Transactions on Reliability*, vol.28, no.3, August 79

[17] Bowles, J. B. and Chi Wan, " Software Failure Modes and Effects Analysis for a Small Embedded Control System", *Proc. of the 2001 Reliability and Maintainability Symposium*, Philadelphia PA, January 2001, pp. 1 – 6.

[18] Goddard, P. L. "Software FMEA Techniques", *Proc. of the 2000 Reliability and Maintainability Symposium*, Los Angeles CA, January 2000, pp. 118 – 122.

[19] Hecht, Herbert and Patrick Crane, "Rare Conditions and their Effect on Software Failures", *Proceedings of the 1994 Reliability and Maintainability Symposium*, pp. 334 - 337, January 1994

[20] Hecht, Herbert, Xuegao An and Myron Hecht, "Computer-Aided Software FMEA for Unified Modeling Language Based Software", Reliability and Maintainability Symposium (RAMS), Los Angeles, January 2004

[7] *Recommended Practice, Software Reliability*, ANSI/AIAA R-013-1992