

Software Reliability Assessment— Myth and Reality

by Myron Hecht, Dr. Herbert Hecht and Dr. Dong Tang

The importance of software as a contributor (if not the actual cause) of catastrophic events has been well documented (Leveson, 95). Moreover, as software is integrated into safety critical systems, the same quantitative reliability requirements which have been previously allocated to hardware are now being allocated to both hardware and software. For example, both U.S. Federal Aviation Regulations and International Joint Aviation Regulations impose maximum acceptable probabilities for failures of systems in passenger transport aircraft. Part 10 of the U.S. Code of Federal Regulations also establishes maximum acceptable probabilities for radioactive releases from nuclear power plants. When these standards were written, analog control systems were the dominant technology, and there was an accepted

methodology for reliability prediction. Now digital (i.e., software-based) systems are replacing analog controls, but the old standards remain in force. The need for updating the standards and methodology extends to unregulated fields (e.g., computer-based automobile electronics), where there is economic motivation to being able to quantify the expected failure behavior.

The greatest need is for methodologies that can demonstrate that quantitative requirements are being met. More detailed quantitative characterizations are also needed to identify system bottlenecks and provide insight for decision making. An overview of the principal methodologies is presented in Table 1, and individual descriptions of each methodology follow.

Technique	Life Cycle Phase	Typical Measure	Advantages	Limitations	Predictive Power
Fault density	All (1)	Faults/KSLOC	Reference data available	Must assume encounter rate	Low
Reliability growth	Test	Failures/execution hour	Some reference data available, objective measurement	Requires observation of multiple failures	Medium
Structured dependability	Test & operation	Failures/execution hour for each segment	Models software structure, objective meas.	Few reference data, requires observations	Medium/high
Rare events	Operation	Failures/operating year	Applicable to very high integrity systems	No reference data, requires observations	Potentially high

(1) Prior to the coding phase, a measure of deficiencies per estimated KSLOC can be employed.

Table 1. Comparison of Reliability Assessment Techniques.

Fault Density Model

The fundamental assumption behind fault density-based prediction models is that as the number of software coding defects (faults) increases, reliability decreases. The U.S. Air Force Rome Laboratory sponsored research into developing predictions of fault density (i.e., number of coding defects per thousand lines of source code) which they could then transform into reliability measures, such as failure rates (Friedman, 92). The predictions of fault density are based on the characteristics of the application, development environment, extent of reuse and other factors. This study and other sources contain data on expected fault density which currently ranges from 1 to 5 faults per thousand source lines of code (KSLOC). The translation of fault density to failure rate requires assumptions about the probability of encountering a fault during execution. This probability can vary widely, depending on the location and nature of the fault. The empirical data on this probability that are currently available do not support very accurate predictions of the failure rate.

Software Reliability Growth Models

Software reliability growth models use measured trends of failure rates (or change in intervals between failures) and extrapolate them to future operation. In most cases, they evaluate the reduction in failure frequency during successive developmental test intervals to estimate the software reliability at the conclusion of the test (and sometimes into operational deployment).

Reliability growth models have been an active area of research since the early 1970s (Farr, 93). Examples are the Schneidewind model, the generalized exponential model, the Musa/Okumoto Logarithmic Poisson model, and the Littlewood/Verrall model (ANSI, 92).

Figure 1 shows an example of such a model. The software is executed over a certain time interval, represented as T_n , until a failure occurs. The time between failures defines a hazard rate. It is expected (but not required in this particular model) that overall, the hazard rate will decrease over time, but that

there are discontinuities as each failure occurs. However, as the program runs for more time, there is increasing confidence in the reliability of the program. Applications of these models have all been demonstrated using real data from software with typical failure rates of 10^{-1} to 10^{-3} per hour (Abdel-Ghaly, 86, Musa, 87).

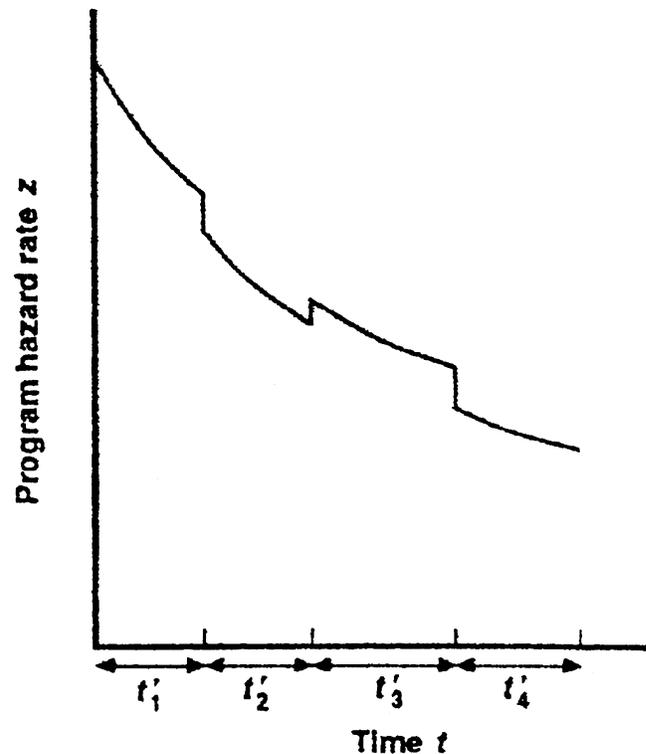


Figure 1. Reliability growth model.

Because of the very low failure rate required for life-critical software, reliability growth models and traditional testing techniques are not suitable (Butler, 93). For example, it would take 108 to 1010 hours (thousands of years) of testing to demonstrate a failure rate of 10^{-7} to 10^{-9} per hour, assuming one copy of software would be tested and one failure would be observed (Butler, 93). Even if 10 copies of the software are tested concurrently, it would still take hundreds of years. The study also cited comments of other experts in the field on this issue, including the following:

Clearly, the reliability growth techniques are useless in the face of such ultra-high requirements. It is easy to see that, even in the unlikely event that the system

had achieved such a reliability, we could not assure ourselves of that achievement in an acceptable time. (Littlewood, 93)

Another limitation of reliability growth models is their lack of ability to model software structure. Reliability growth models treat the software as a black box and form a single expression for its reliability. Critical software systems include fault tolerance mechanisms, such as error detection and handling, redundancy management and back-up tasks. As such, the reliability of the whole software system cannot be simply quantified by the number of failures observed at the component level. For example, a transient task failure may be covered by the fault tolerance provisions and may not affect critical functions. This scenario has been verified by several studies (Lee, 93; Tang, 95) which showed that 80 to 95 percent of software failures in real time fault-tolerant systems are recoverable by redundant processes. In such a case, reliability growth models do not provide meaningful answers, and structured dependability models must be used.

Structured Dependability Models

An alternative approach uses structured measurements, similar to the established hardware practice. In this technique, application software tasks, operating system kernels or executives, and hardware components are regarded as equivalent elements in a system. The operating times, failure rates, correlated failure probability, recovery times and recovery probabilities of any of these elements can be measured in reliability tests.

Reliability and availability are then estimated by models of the system structure, using measurement-based parameters for each component (Tang, 95). Statistical estimation of reliability and availability parameters and reliability modeling based on these parameters has been a research topic in computer engineering for 15 years (Iyer, 93). These analyses are based on operational logs and failure data.

Dependability models have been used to evaluate operational software based on failure data collected from commercial computer operating systems for

about ten years (Hsueh, 87; Tang, 92; Lee, 93). The methodology has been extended to evaluate availability for air traffic control software systems in the late testing phase (Tang, 95) and most recently to the early operational phase at multiple sites.

In our experience, three model structures have been found useful in measurement-based dependability evaluation: the reliability block diagram, the k-out-of-n model, and the Markov chain. Both reliability diagrams and k-out-of-n models are combinatorial models and typically assume failure independence among modeled components. Markov chains are stochastic models that can incorporate interactions among components and failure dependence in the model.

However, the current practice of measurement-based evaluation for individual software systems (with the number of installations <100) is still limited to failure rates of 10^{-2} to 10^{-5} per hour and an availability of three to five 9's (0.999 to 0.99999). For example, the newly developed FAA Voice Switching and Control System (VSCS) is being installed in 21 major U.S. air traffic control centers. The system availability (dominated by software) was evaluated to have five 9's as of March 31, 1996. If no major failure occurs in the future, it would take 15 years of normal operation of the 21 systems to demonstrate an availability of the required seven 9's at the 80% confidence level.

In the process of collecting and analyzing such data, additional studies can be undertaken for more detailed examinations of underlying causes. For example, analyses of workload and failure data collected from IBM mainframes (Butner, 80) and DEC minicomputers (Castillo, 81) revealed that the average system failure rate is strongly correlated with the average workload on the system. Recent studies of data from DEC (Tang, 92) and Tandem (Lee, 93) systems showed that correlated failures across processors are significant in multicomputers, and their impact on dependability is significant.

The underlying assumption in these measurement-based approaches is that the fundamental failure mechanisms are triggered stochastically, i.e., are

non-deterministic (“Heisenbugs”). However, there is a class of failures in which the software runs to completion but produces an unacceptable output. For example, an electronic speed control on a turbine may in fact not shut down the device in an over-speed condition even though there was no crash, hang, stop or delay failure. This deterministic failure condition may be traced to a logic fault in the code or an incorrect set of parameters (e.g., the RPM threshold for that particular turbine under the specified set of pressures and temperatures). However, the root cause of the failure may in fact lie much deeper, i.e., defects in the system requirements or software requirements.

The techniques and methodologies for estimating the probabilities for these deterministic incorrect response failures are very immature. It is tempting to “wish them away” by positing that an adequate V&V (verification and validation) or integration testing program should uncover them. However, resources are finite, and it is rarely feasible to provide sufficient time or money to perform the level of testing needed to uncover all such failures, even in systems designed for high dependability. From a practical perspective, when estimating software failure rates, one should look not only at failures that cause losses or delays of system services (e.g., crash, hang, stop) but also incorrect response failures. If there are incorrect responses at the final stages of testing or integration, or in initial operation, then reliability predictions made exclusively on the basis of stochastic failures may not be valid.

Obtaining adequate data from which to assess reliability and availability is critical to any measurement-based methodology. This obvious principle can be difficult to implement in practice for dependability assessments because of the constraints of an expensive testing program or impending project deadlines. Adequate data means monitoring and recording events of interest such as failures and recoveries of components, as well as performance parameters of the target system while it is operating under representative workloads. It also means collecting data on failure modes so that an assessment of the importance of deterministic failures can be made. The events and parameters to be collected should be rep-

resentative of the system operation and meaningful for the assessment of the system. Measurements should be made continuously for a sufficient period to yield statistically significant data. Operating logs should include information about the location, time and type of the error, the system state at the time of failure or abnormal operation, and error recovery (e.g., retry) information where applicable.

Assessment by Rare Events Technique

As previously discussed, none of the techniques described above can furnish a credible direct assessment for failure rates lower than 10^{-6} per hour. Under favorable circumstances, the structured dependability approach may support the conclusion that such requirements are met by two or more independent versions running under a highly reliable selection or voting scheme, and this is indeed the way adopted by many exacting applications. It is an expensive solution, because in addition to the multiple software implementations it requires the development and very extensive testing of selection mechanisms. Further, multi-version software tends to degrade the computational performance (because of the need to wait for the slowest version to complete execution and related issues), and the independence of the versions cannot be taken for granted (because they implement a common set of requirements). Therefore, there is ample motivation to investigate other assessment techniques.

The basic premise of the rare events approach is that well-tested software does not fail under routine input conditions, which means that failures must be triggered by unusual input data or computer states. This assumption is validated by a number of investigations that are summarized elsewhere (Hecht, 94). Late-phase testing will usually subject the program to test cases that emphasize these rare conditions, and this permits assessment of the failure probability by the likelihood of encountering the rare conditions that triggered the failure rather than by test time.

As an example, consider a program that failed twice during the last 1,000 hours of test. The first failure occurred on restart after a simulated power interrup-

tion, while at the same time one of the input signals faulted to zero (sensor fault). The second failure occurred when one out of three inputs faulted to high and another one to low. Is the failure rate of this program 2×10^{-3} per hour as computed from the test time?

Most observers would disagree with such an assessment and will find it more reasonable to take into account the occurrence rate of the triggering events in the environment in which this program will operate. Assume that power interruptions normally occur only once a year, and sensor failures to zero are expected to occur only once every two years. The combined probability of the joint event (assuming the individual triggers to be independent), is therefore well over 10^{-7} per hour. The second test case that triggered a failure (one sensor high and one low), has an even lower probability. After the software has been modified so that it will not fail again due to these triggers, its failure probability will be much lower than that computed from the test time.

A quantitative assessment will consider the total number of test cases that had been used and the probability of the natural occurrence of the simulated conditions. To illustrate the basics of the quantitative assessment, assume that during the 1,000 hours of test there were 10,000 test cases that simulated conditions that are expected to arise more frequently than once per 10,000,000 hours and 1,000 test cases simulating conditions that are expected to occur less frequently. Since the only failures observed were due to the second category, and since there was a ten-fold greater opportunity for failures under the first category, it can be reasoned that the failure rate in the natural environment is expected to be not more than 10^{-7} per hour. The mathematical formulation of this approach is based on the probability of drawing black and white balls from an urn (Hecht, 96).

Conclusions

Reliability assessments based on fault density and reliability growth models support planning and comparative evaluations but are usually not sufficiently validated to be a credible basis for stating that a soft-

ware product has attained a required reliability, particularly when the required reliability is high. Structured dependability models can furnish estimates that are more precise and that also identify the elements where reliability improvement will provide the greatest benefit. They are well suited for designing and maintaining highly dependable computer systems intended for flight control, ground transportation, air traffic control and nuclear power plant safety functions.

Except under unusually favorable circumstances, none of these methods can currently assess whether a software product meets requirements for failure rates of less than 10^{-6} per hour. The rare events approach, described in the preceding section, has the potential for being useful for applications that demand the highest dependability, but it is the least validated of the methodologies discussed here. Because of the constantly increasing use of software-based systems in critical applications, further research into software reliability assessment is urgently needed.

References

- A. A. Abdel-Ghaly, P. Y. Chan and B. Littlewood, "Evaluation of Competing Software Reliability Predictions," *IEEE Transactions on Software Engineering*, vol SE-12 no. 9, September 1986, pp. 950-967.
- "American National Standard, Recommended Practice for Software Reliability," American National Standards Institute, ANSI/AIAA R-013-1992.
- R. W. Butler and G. B. Finelli, "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software," *IEEE Transactions on Software Engineering*, vol SE19 no. 1, pp. 3 - 12, January 1993.
- S.E. Butner and R.K. Iyer, "A Statistical Study of Reliability and System Load at SLAC," Proc. 10th Int. Symp. Fault-Tolerant Computing, pp. 207-209, Oct. 1980.

-
- X. Castillo and D.P. Siewiorek, "Workload, Performance, and Reliability of Digital Computer Systems," *Proc. 11th Int. Symp. Fault-Tolerant Computing*, pp. 84-89, July 1981.
- W. H. Farr and O. Smith, "*Statistical Modeling and Estimation Functions for Software (SMERFS) - User's Guide*," NSWCDD TR84-371, Revision 3, September 1993.
- Michael Friedman, "Methodology for Software Reliability Prediction and Assessment" Report RL-TR-92-52, Rome Laboratory 1992 (2 volumes).
- Herbert Hecht and Patrick Crane, "Rare Conditions and their Effect on Software Failures," *Proceedings of the 1994 Reliability and Maintainability Symposium*, pp. 334 - 337, January 1994.
- Herbert Hecht and Myron Hecht, "Quality Assurance and Testing for Safety Systems," *Proc. CADT-ED*, Beijing, July 1996.
- M. C. Hsueh and R. K. Iyer, "A Measurement-Based Model of Software Reliability in a Production Environment," *Proceedings of the 11th Annual Computer Software and Applications Conference*, pp. 354-360, October 1987.
- R.K. Iyer and D. Tang, "Experimental Analysis of Computer System Dependability," Technical Report CRHC-93-15, Center for Reliable and High-Performance Computing, University of Illinois at Urbana-Champaign, July 1993.
- I. Lee, D. Tang, R.K. Iyer, and M.C. Hsueh, "Measurement-Based Evaluation of Operating System Fault Tolerance," *IEEE Transactions on Reliability*, pp. 238-249, June 1993.
- Nancy G. Leveson, *Safeware*, Addison Wesley, Reading, Mass., 1995.
- D. Tang and R.K. Iyer, "Analysis and Modeling of Correlated Failures in Multicomputer Systems," *IEEE Trans. Computers* Vol. 41, No. 5, pp. 567-577, May 1992.
- D. Tang and M. Hecht, "Evaluation of Software Dependability Based on Stability Test Data" *Proc. 11th Int. Symp. Fault-Tolerant Computing*, Pasadena, California, June 1995.