Software Reliability

by

S. R. Dalal, M.R. Lyu, C. L. Mallows

Bellcore, Lucent Technologies, AT&T Research

1. Introduction

The demand for complex software systems has increased more rapidly than the ability to design, implement, test, and maintain them, and the reliability of software systems has become a major concern for our modern society. With the last decade of the 20th century, computer software has already become the major source of reported outages in many systems. Consequently, recent literature is replete with horror stories of projects gone awry, generally as a result of problems traced to software.

More recently software failures have impaired several high-visibility programs in the health industry, in some cases they even killed people. Described in the book by Lee (1992), the massive Therac-25 radiation therapy machine was hit by software errors in its sophisticated control systems and claimed several patients' lives in 1985 and 1986. South West Thames Regional Health Authority (1993) reported the incidence on October 26, 1992, when the Computer Aided Dispatch system of the London Ambulance Service broke down right after its installation, paralyzing the capability of the world's largest ambulance service to handle 5000 daily requests in carrying patients in emergency situations. In the aviation industry, although the real causes for several airliner crashes in the past few years remained mysteries, experts pointed out that software control could be the chief suspect in some of these incidents due to its inappropriate response to the pilots' desperate inquires during an abnormal flight condition.

To this end, many software companies devote a major share of project development costs identified with the design, implementation, and assurance of reliable software, and they recognize a tremendous need for systematic approaches to measure and assure software reliability. IEEE (1991) defines software reliability as the probability of failure-free software operations for a specified period of time in a specified environment. It is one of the attributes of software quality, and is generally accepted as the key one since it quantifies software failures - which can make a powerful system inoperative or, as with the Therac-25, deadly. As a result, reliability is an essential ingredient in customer satisfaction. In fact, ISO 9000-3 (1991) specifies measurement of field failures as the only required quality metric: "... at a minimum, some metrics should be used which represent reported field failures and/or defects form the customer's viewpoint. ... The supplier of software products should collect and act on quantitative measures of the quality of these software products."

Software Reliability Engineering, consequently, is the field that quantifies the operational behavior of software-based systems with respect to user requirements concerning reliability. It includes:

1. Software reliability measurement, which includes estimation and prediction, with the help of software reliability models established in the literature;
2. The attributes and metrics of product design, development process, system architecture, software operational environment, and their implications on reliability; and
3. The application of this knowledge in specifying and guiding system software architecture, development, testing, acquisition, use, and maintenance.

Many of the current software reliability engineering techniques and practices are detailed in the Handbook of Software Reliability Engineering by Lyu (1996).

We focus on software reliability models and measurements in this chapter. A software reliability model specifies the general form of the dependence of the failure process on the principal factors that affect it: fault introduction, fault removal, and the operational environment. Figure 1 shows the basic ideas of software reliability modeling.

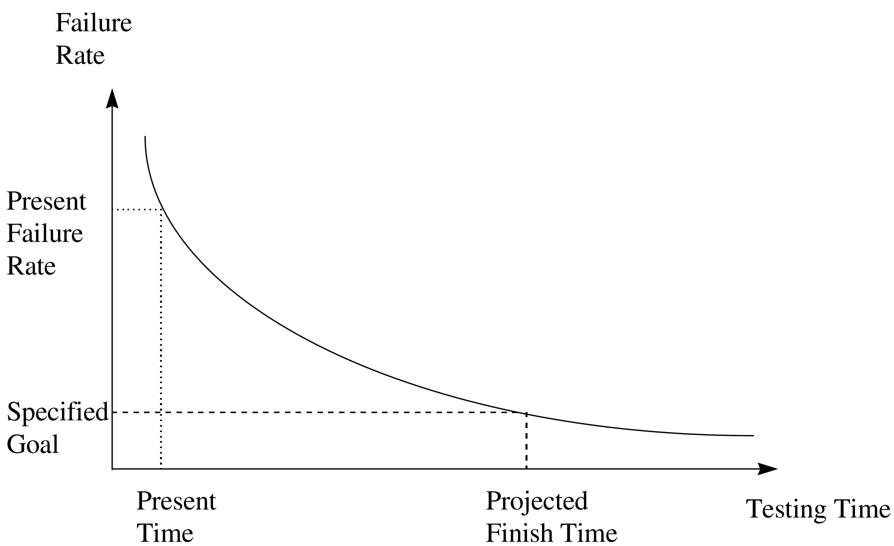


Figure 1: Basic Ideas of Software Reliability Modeling

In Figure 1, the failure rate of a software system is generally decreasing due to the discovery and removal of software failures. At any particular time (say, the point marked "present time"), it is possible to observe a history of the failure rate of the software. Software reliability modeling forecasts the curve of the failure rate by statistical evidences. The purpose of this measure is two-fold: 1) to predict the extra time needed to test the software to achieve a specified objective; 2) to predict the expected reliability of the software when the testing is finished.

Software reliability measurement includes two types of activities, static reliability prediction and dynamic reliability estimation, which will be discussed in the following two sections.

2. Static Reliability Prediction Models

The purpose of static reliability models is to perform software reliability prediction in an early stage of software development. This activity determines future software reliability based upon available software metrics and measures. Particularly when failure data are not available (e.g., software is in the design or coding stage), the metrics obtained from the software development process and the characteristics of the resulting product can be used to determine reliability of the software upon testing or delivery. We discuss two prediction models: the Phase-Based Model and the Rome Laboratory Model.

2.1 Phase-Based Model

Gaffney and Davis (1988) developed the phase-base model, which makes use of fault statistics obtained during the technical review of requirements, design, and the implementation to predict the reliability during test and operation. Basically, the model assumes that faults found in different phases of life cycle follow a Raleigh density function. This model assumes that the development effort's current staffing level is directly related to the number of faults discovered during the development phase, and the code size estimates are available during the early phases of a development effort. The faults found during the requirements analysis and software design are normalized by the code size estimates. The model is expressed as:

ΔV_t = number of discovered faults per KLOC from time t-1 to t

$= E[\exp(-B(t\text{-}1)^2 - \exp(-Bt^2)]$

where E = total lifetime fault rate expressed in faults per thousand lines of code (KLOC); t = fault discovery index, where t = 1 means requirements analysis, t = 2 means software design, t = 3 means implementation, t = 4 means unit test, t = 5 means acceptance test; and B = $1 / 2\tau_p^2$, where τ_p is the defect discovery phase constant, the peak of a continuous curve fit to the failure data. This is the point at which 39 percent of faults have been discovered.

The cumulative form of the model is $V_t = E[1 - \exp(-Bt^2)]$, where V_t is the number of faults per KLOC that have been discovered through phase t. As data become available B and E can be estimated. This quantity can also be used to estimate the number of remaining faults at stage t by multiplying $E\exp(-Bt^2)$ by the number of source line statements at that point.

2.2. Rome Laboratory Work

The Air Force's Rome Laboratory (1987) model developed predictions of fault density which could be transformed into other reliability measures such as failure rates. A number of factors related to fault density at the earlier life cycle phases were selected in this model:

1. *Application type* (e.g., real-time control systems, scientific, information management), denoted by *A*.
2. *Development environment* (characterized by development methodology and available tools), denoted by *D*. The types of development environments considered are organic, semidetached, and embedded modes.
3. *Requirements and design representation metrics,* including

SA for anomaly management

ST for traceability

SQ for incorporation of quality review results into the software

4. *Software implementation metrics,* including

SL for language type (assembly, high-order, etc.)

SS for program size

SM for modularity

SU for extent of reuse

SX for complexity

SR for incorporation of standards review results into the software

The initial fault density prediction is then:

$$\delta_0 = A \times D \times (SA \times ST \times SQ) \times (SL \times SS \times SM \times SU \times SX \times SR)$$

Once the initial fault density has been found, a prediction of the initial failure rate is made as

$$\lambda_0 = \mathrm{F} \times \mathrm{K} \times (\delta_0 \times \mathrm{KLOC}) = \mathrm{F} \times \mathrm{K} \times \mathrm{W}_0$$

The number of inherent faults is $\mathrm{W}_0 = (\delta_0 \times \mathrm{KLOC})$; F is the linear execution frequency of the program, which is the average machine instruction rate divided by the number of object instructions in the program; and K is the fault expose ratio, where $1.4 \times 10^{-7} \leq \mathrm{K} \leq 10.6 \times 10^{-7}$ based on historical projects.

3.0 Software Reliability Estimation Models

Software reliability estimation determines current software reliability by applying statistical inference techniques to failure data obtained during system test or during system operation. This is a measure regarding the achieved reliability from the past until the current point. Its main purpose is to assess the current reliability, and determine whether a reliability model is a good fit in retrospect. Since the reliability tends to improve during software testing or operation period, the measurement models are also called reliability growth models. Most current software reliability models fall into this category. Details of these models can be found in Lyu (1996), in which a number of best current software reliability tools which implement these models are also included. Other surveys are given by Musa et al (1987) and Singpurwalla and Wilson (1995).

3.1 Using reliability models.

The success of a model is often judged by how well it fits a curve to the observed "number of faults vs. time" function μ **(t).** On general grounds, this may have little to do with how useful the model is in predicting future faults in the present system, (a better fit can mean worse prediction), or future experience with another system, unless we can establish statistical relationships between measurable attributes of the systems (e.g. KLOC, various measures of complexity) and estimated parameters of the fitted models.

Different sets of assumptions can lead to equivalent models; for example the assumption that for each fault, the time-to-detection is a random variable with a Pareto distribution, these random variables being independent, is equivalent to assuming that each fault has an exponential lifetime, with these lifetimes being independent, and the rates for the different faults being distributed according to a Gamma distribution (this is Littlewood's (1981) model). A single experience cannot distinguish between a model that assumes a fixed but unknown number of faults and a model that assumes this number is random. Little is known about how well the various models can be distinguished.

3.2 Assumptions.

Most of the published models are based on similar assumptions These commonly include the following.

* The module being tested remains essentially unchaged throughout testing, except for the removal of faults as they are found.   Some models allow for the possibility that faults are not corrected perfectly. Miller (1986) assumes that if faults are not removed as they are found, then each fault causes failures according to a stationary Poisson process; these processes are independent of one another and may have different rates. By specifying the rates, many of the models mentioned below can be obtained.

* Removing a fault does not affect the chance that a different fault will be found;

* "Time" is measured in such a way that testing effort is constant. Musa (1975) reports that execution time (processor time) is the most sucessful way to measure time.

* The model is Markovian, i.e. at each time, the future evolution of the testing process depends only on the present state (the current time, the number of faults found and remaining, and the overall parameters of the model), and not on details of the past history of the testing process. In some models a stronger property holds, namely that the future depends only on the current state and the parameters, and not on the current time. We call this the "strong Markov" property.

* All faults are of equal importance (contribute equally to the failure rate).

* At the start of testing, there is some finite total number of faults, which may be fixed (known or unknown) or random; if random, its distribution may be known or of known form with unknown parameters;   alternatively, the "number of faults" is not

assumed finite, so that if testing continues indefinitely, an ever-increasing number of faults will be found.

* Between failures, the hazard rate follows a known functional form; this is often taken to be simply a constant.

3.3 Fixed-shape models.

In Binomial models, the total number of faults is some number N; the number found by time t has a Binomial distribution with mean $NF(t)$. It follows that the number of faults found in any interval of time (including the interval (t, ∞))is also Binomial. Letting N be Poisson (with some mean ν) gives the related Poisson model; now the number of faults found in any interval is Poisson, and for disjoint intervals these numbers are independent. The hazard rate at time t is $F'(t)/(1-F(t))$. These models are Markovian but not strongly Markovian, except when F is exponential; this case was stuidied by Jelinski/Moranda (1972), Shooman (1972), Schneidewind (1975), Musa (1975), Moranda (1975), and Goel/Okomoto (1979). Schick/Wagoner (1973), Wagoner (1973) and Crow (1974) made F a Weibull distribution; Yamada/Ohba/Osaki (1983) made F a Gamma distribution; and Littlewood's (1981) model is equivalent to assuming F to be Pareto. Musa/Okumoto (1984) assumed the hazard rate to be an inverse linear function of time; for this "logarithmic Poisson" model the total number of failures is infinite.

3.4 Strongly Markov models.

These can be obtained By specifying how the hazard rate of the failure process depends on the current state. Moranda (1975) assumed that between failures, the hazard rate is constant, and decreases geometrically at each failure. Littlewood/Verrall (1973) proposed a class of models in which after the i-th fault is found, the hazard rate becomes ***G_i/xi(i)*** where ***xi*** is some simple function and ***G_i*** is a random variable (independent for different i) with a Gamma distribution. Models of this class were studied by Iannino (1979) Musa (1979), and Keiller et al (1983).

4. Reliability Growth Modeling with Covariates:

We have so far discussed a number of different kinds of reliability models of varying degrees of plausibility, including phase based models depending upon a Raleigh curve, dynamic growth models like the Goel Okumoto model, etc. These models are applied at either the testing stage, or the field monitoring stage. The dynamic growth models take as input either failure time or failure count data, and fit a stochastic process model to reflect reliability growth. The differences between the models lie principally in assumptions made on the underlying stochastic process generating the data.

However, most existing models assume that there are no explanatory variables available. This assumption is assuredly simplistic, when the models are used to model a testing process, for all but small systems involving short development and life cycles. For large systems (*e.g.*, greater than 100,000 lines of code) there are variables, other than time, which are very relevant. For example, it is typically assumed that the number of faults (found and unfound) in a system under test remains stable during testing. This implies that the code remains frozen during testing. However, this is rarely the case for large systems since aggressive delivery cycles force the final phases of development to overlap with the initial stages of system test. Thus, the size of code, and consequently, the number of faults in a large system can vary widely during testing. If these changes in code size are not considered as a *covariate*, one is, at best, likely to have an increase in variability and a loss in predictive performance, and at worst, a poor fitting model with unstable parameter estimates. We briefly describe a general approach proposed by Dalal and McIntosh (1994) for incorporating covariates along with a case study they reported dealing with reliability modeling during product testing when code is changing.

Example. Consider a new release of a large telecommunications system with approximately 7,000,000 noncommentary source lines (NCSL) and 300,000 lines of noncommentary new or changed source lines (NCNCSL). For a faster delivery cycle, the source code used for system test was updated every night throughout the test period. At the end of each of 198 calendar days in the test cycle, the number of faults found, NCNCSL, and the staff time spent on testing were collected. Figure 4a and 4b portray growth of the system in terms of NCNCSL and faults, respectively, against staff time. The corresponding numerical data are provided in Dalal and McIntosh (1994)..

Assume that the testing process is observed at time $t_i, i = 0, \ldots, h$, and at any given time, the amount of time it takes to find a specific bug is exponential with rate m. At time t_i, the total number of faults remaining in the system is Poisson with mean l_{i+1}, and NCNCSL is increased by an amount C_i. This change adds a Poisson number of faults with mean proportional to C, say qC_i. These assumptions lead to the mass balance equation, namely that the expected number of faults in the system at t_i (after possible modification) is the expected number of faults in the system at t_{i-1} adjusted by the expected number found in the interval (t_{i-1}, t_i) plus the faults introduced by the changes made at t_i:

$$l_{i+1} = l_i e^{-m(t_i - t_{i-1})} + qC_i ,$$

for $i = 1, \ldots h$. Note that q represents the number of new faults entering the system per additional NCNCSL, and l_1 represents the number of faults in the code at the start of system test. Both of these parameters make it possible to differentiate between the new code added in the current release and the older code. For the data depicted in Figure 5, the estimated parameters are $q = 0.025$, $m = 0.002$, and $l_1 = 41$. The fitted and the observed data are plotted against staff time in Figure 4b. The fit is evidently very good. Of course assessing the model on independent or new data is required for proper validation.

Now we examine the efficacy of creating a statistical model. The estimate of q is highly significant, both statistically and practically, showing the need for incorporating changes in NCNCSL as a covariate. Its numerical value implies that for every additional 10,000 NCNCSL added to the system, 25 faults are being added as well. For these data, the predicted number of faults at the end of the test period is Poisson distributed with mean 145. Dividing this quantity by the total NCNCSL, gives 4.2 per 10,000 NCNCSL as an estimated field fault density. These estimates of the incoming and outgoing quality are valuable in judging the efficacy of system testing and for deciding where resources should be allocated to improve the quality. Here for example, system testing was effective in that it removed 21 of every 25 faults. However, it raises another issue: 25 faults per 10,000 NCNCSL entering system test may be too high and a plan ought to be considered to improve the incoming quality.

None of the above conclusions could have been made without using a statistical model. These conclusions are valuable for controlling and improving the process. Further, for this analysis it was essential to have a covariate other than time.

Figure 4a. NCNCSL vs. Staff Time

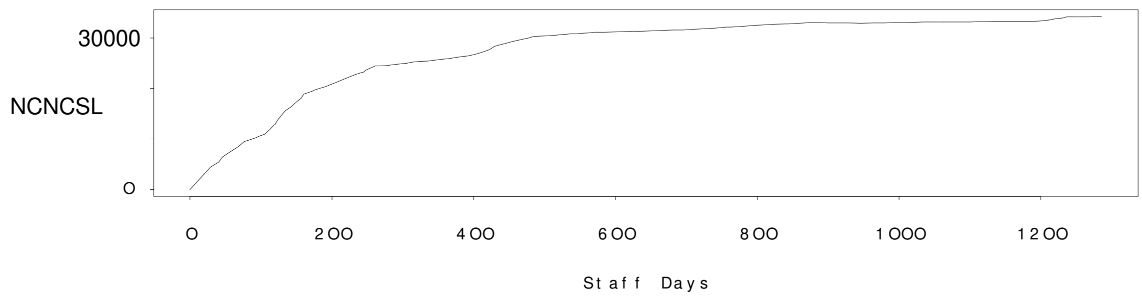
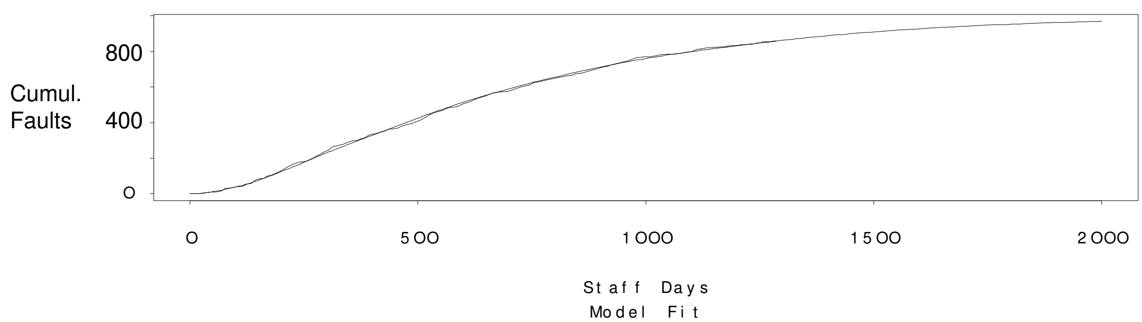


Figure 4b. Observed and Fitted Data



Figures 4a & 4b. Plots of module size (NCNCSL) versus staff time (days) for a large telecommunications software system (top). Observed and fitted cumulative faults versus staff time (bottom). The dotted line (barely visible) represents the fitted model, the solid line represents the observed data, and the dashed line is the extrapolation of the fitted model.

5. When to Stop Testing Software?

Dynamic reliability growth models can be used to make decisions related to when to stop testing. Software testing is a necessary but expensive process, consuming one third to one half the cost of a typical development project. Testing a large software system costs thousands of dollars per day. Overzealous testing can lead to a product that is over-priced and late to market, while fixing a fault in a released system is usually an order of magnitude more expensive than fixing the fault in the testing lab. Thus, the question of how much to test is an important economic question. We discuss an economic formulation of the when to stop testing issue as proposed by. Dalal and Mallows,(1988, 1990) .Other formulations have been proposed by Dalal and Mallows (1992), Singpurwalla (1991).

Like many other reliability models Dalal & Mallows' stochastic model assumes that there are N (unknown) faults in the software, and the times to find faults are observable and are i.i.d. exponential with rate m. Their economic model defines the cost of testing at time t to be $ft - c\, K(t)$. where $K(t)$ is the number of faults observed to time t and f is the cost of operating the testing lab per unit time. The constant c is the net cost of fixing a fault after rather than before release. Under somewhat more general assumptions, Dalal & Mallows (1988) found the exact optimal stopping rule by assuming that N is Poisson(l), and that l is Gamma (a , b). The structure of the exact rule, which is based on stochastic dynamic programming, is rather complex. However, for large N, which is necessarily the case for large systems, the optimal stopping rule is: stop as soon as $f\,(e$ sup $\{m$ t$\}$ -1) / (m c) $\geq K(t)$. Besides the economic guarantee, it gives a guarantee on the number of remaining faults, namely that it has a Poisson distribution with mean f / (m c) . Thus, instead of determining the ratio f / c from economic considerations, we can choose it so that there are probabilistic guarantees on the number of remaining faults. Some practitioners may find that this probabilistic guarantee on the number of remaining faults is more relevant in their application. (See Dalal and Mallows, 1992, for a more detailed discussion.) Finally, by using reasoning similar to that used in deriving equation (4.5) of Dalal and Mallows (1988), it can be shown that the current estimate of the additional time required for testing, $\Delta \boldsymbol{t}$, is given by: $(1/\boldsymbol{m})\ log\ \boldsymbol{c}\ \boldsymbol{m}\ \boldsymbol{K(t)}$ / { ($\boldsymbol{f}$ (e sup { $\boldsymbol{m}$ $\boldsymbol{t}$ } - 1)) }. For applications of this we refer to Dalal and Mallows (1990).

6. Discussions and Conclusions

Software reliability modeling and measurement have drawn a tremendous amount of attention recently in various industries concerning the quality of software. Many reliability models have been proposed, many success stories reported, several conferences and forums formed, and much project experience shared. Here we would like to offer some caution to the users regarding the usage of software reliability models.

In fitting any model to a given data set, first one must bear in mind a given model's assumptions. For example, if a model assumes a fixed number of software faults will be removed with a limited period of time, but in the observed process the number of faults are not fixed (e.g., new faults are added due to imperfect fault removal), then use another model which does not make this assumption.

A second model limitation and implementation issue concerns future predictions. If the software is being operated in a manner different from the way it is tested (e.g., new capabilities are being exercised that were not tested before), the failure history of the past will not reflect these changes, and poor predictions may result. Developing operational profiles, proposed by Musa (1996), is very important if one wants to accurately predict future reliability in the user's environment.

Another issue relates to software development environment. Most models are primarily applicable from integrated testing onward: the software is assumed to have matured to the point that extensive changes are not being routinely made. These models can't have a credible performance when the software is changing and the churn of software code is observed during testing. In this case the techniques described in this chapter should be used to handle the dynamic testing situation.

Finally software reliability models cannot make an impact if it is not tied with software testing and operational costs to determine the optimal time to stop testing. We have described such an economic model in this chapter.

References

Crow, L.H., 1974. "Reliability Analysis for Complex Repairable Systems," Reliability and Biometry, F. Proshan and R.J. Serfling *(eds.) SIAM, Philadelphia, 379-410.

Dalal, S.R. and C.L. Mallows. 1988. When should one stop software testing*? J. Amer. Statist. Assoc*. **83**:872-879.

Dalal, S.R. and C.L. Mallows. 1990. Some graphical aids for deciding when to stop testing software., *IEEE J. Special Areas in Communications* **8**:169-175. (Special issue on Software Quality & Productivity.)

Dalal, S.R. and C.L. Mallows. 1992. Buying with exact confidence. *Ann. Appl. Prob*. **2**:752-765.

Dalal, S.R. and A.M. McIntosh. 1994. When to stop testing for large software systems with changing code. *IEEE Trans. Software Engineering*, **20**:318-323.

Gaffney, J.D. and C.F. Davis, 1988. "An Approach to Estimating Software Errors and Availability," SPC-TR-88-007, version 1.0, March 1988, Proceedings of the 11th Minnowbrook Workshop on Software Reliability, July 1988.

Goel, A.L. and K. Okumoto, 1979. "Time-Dependent Error-Detection Rate Model for Software and Other Performance Measures," IEEE Transactions on Reliability, R-28(3):206-211.

Institute of Electrical and Electronics Engineers, 1991. ANSI/IEEE Standard Glossary of Software Engineering Terminology, IEEE Std. 729-1991.

ISO, 1991. "Quality Management and Quality Assurance Standards - Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software," ISO 9000-3, Switzerland.

Keiller, P.A., B. Littlewood, D.R. Miller, and A. Sofer, 1983. Comparison of software reliability predictions, Proceedings of the 13th IEEE International Symposium on Fault-Tolerant Computing (FTCS-13), Milano, Italy, 128-134.

Littlewood, B and V. Verrall, 1973. A Bayesian Reliability Model with a Stochastically Monotone Failure Rate, IEEE Transactions on Reliability, R-23(2):108-114.

Lee, L., 1992. The Day the Phones Stopped: How people Get Hurt When Computers Go Wrong, Donald I. Fine, Inc., New York.

Lyu, M.R., 1996. Handbook of Software Reliability Engineering, McGraw-Hill, New York.

Moranda, P.B. and Z. Jelinski, 1972. Final report on Software Reliability Study, McDonnell Douglas Astronautics Company, MADC Report Number 63921.

Moranda, P.B., 1975. Predictions of Software Reliability During Debugging, Proceedings of the Annual Reliability and Maintainability Symposium, Washington, D.C., 327-332.

Musa, J.D. and K. Okumoto, 1984. A Logarithmic Poisson Execution Time Model for Software Reliability Measurement, Proceedings Seventh International Conference on Software Engineering, Orlando, Florida, 230-238.

Musa, J.D., A. Iannino, and K. Okumoto, Software Reliability - Measurement, Prediction, Application, 1987. McGraw-Hill, New York.

Musa, J.D., G. Fuoco, N. Irving, D. Kropfl, and B. Juhlin, 1996. "The Operational Profile," Chapter 5 in (Lyu 1996), 167-218.

Rome Laboratory, 1987. Methodology for Software Reliability Prediction and Assessment, Technical Report RADC-TR-87-171 ; revised on Technical Report RL-TR-92-52, 1992.

Schneidewind, N.F., 1975. "Analysis of Error Processes in Computer Software," Sigplan Note, 10(6):337-346.

Singpurwalla, N.D. 1991. Determining an optimal time interval for testing and debugging software. *IEEE Trans. Software Engineering* **17**(4):pp313-319

Schick, G.J., and R.W. Wolverton, 1973. "Assessment of Software Reliability," Proceedings of the Operations Research, Physica-Verlag, Wurzburg-Wien, 395-422.

South West Thames Regional Health Authority, 1993. Report of the Inquiry into the London Ambulance Service.

Yamada, S., M. Ohba, and S. Osaki, 1983. "S-Shaped reliability Growth Modeling for Software Error Detection," IEEE Transactions on Reliability, R-32(5):475-478.