A Selective Survey and direction on the software of Reliability Models

Vipin Kumar
Research Scholar, S.M. Degree College, Chandausi

Abstract:
Software development, design and testing have become very intricate with the advent of modern highly distributed systems, networks, middleware and interdependent application. 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. Software reliability modeling and measurements have drawn quite a bit of attention recently in various industries due to concerns about the quality of software. In few years of 21st century, many reported system outages or machine crashes were traced back to computer software failures.

In this paper, I have many challenges in getting wide spread use of software reliability models. I am focus on software reliability models and measurements. 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. During the test phase, the failure rate of a software system is generally decreasing due to discovery and correction of software faults. With careful record-keeping procedures in place, it is possible to use statistical methods to analyze the historical record. The purpose of these analyses is twofold:(1) to predict the additional time needed to achieve a specified reliability objective; (2) to predict the expected reliability when testing is finished.

Key words: Dynamic model, Growth model, Reliability software, Static model, Telecommunication.

Introduction:

In few year of century, many reported system outages or machine crashes were traced back to computer software failures. Consequently, recent literature is replete with horror stories due to software problems. Software failure has impaired several high visibility programs in space, telecommunications and defense and health industries. The Mars Climate Orbiter crashed in 1999. The Mars Climate Orbiter Mission failure investigation Board [1] concluded that "The root cause of the loss of the spacecraft was the failed translation of English unit into metric units in a segment of ground based, navigation related mission software. Current versions of the Osprey aircraft, developed at a cost of billions of dollars, are not deployed because of software induced field failure. In the health industry [2], the Yherac-25 radiation therapy machine was hit by software errors in its sophisticated control systems and claimed several patients' lives in 1985 &1986. Even in the telecommunications industry, known for its five nines reliability, the nationwide long distance network of a major carrier suffered an embarrassing network outage on January 1990, due to software problem. In 1991, a series of local network outage occurred in a number of US cities due to software problems in central office switches [3].

Software reliability is defined as the probability of failure free software operations for a specified period of time in a specified environment [4]. The software reliability field discusses ways of quantifying it and using it for improvement and control of the software development process.. Software reliability is operationally measured by the number of field failures, or failures seen in development, along with a variety of ancillary information. The ancillary information includes the time at which the failure was found, in which part of the software it was found, the state of software at that time, the nature of the failure. ISO9000-3 [5] is the weakest amongst the recognized standards, in that it specifies measurement of field failures as the only required quality metric.

In this paper, I take a narrower view and just look at models that are used in software reliability-their efficacy and adequacy without going into details of the interplay between testing and software reliability models. Software reliability measurement includes two types of model: static and dynamic reliability estimation, used typically in the earlier and later stages of development respectively. These will be discussed in the following two sections. One of the main weaknesses of many of the models is that they do not take into account ancillary information, like churn in system during testing. Such a model is described in Growth reliability. A key use of the reliability models is in the area of when to stop testing. An economic formulation is discussed in next paragraph.

Static Models:

One purpose of reliability models is to perform reliability prediction in an early stage of software development. This activity determines future software reliability based upon available software metrics and measures. Particularly when field failure data are not available (e.g. software is in design or coding stage), the metrics obtained from the software development process and the characteristics of the resulting product can be used to estimate the reliability of the software

upon testing or delivery. I am discussing two prediction models: the phase based model by Gaffney and Davis [10] and a predictive development life cycle model from Telcordia Technologies by Dalal and Ho [11].

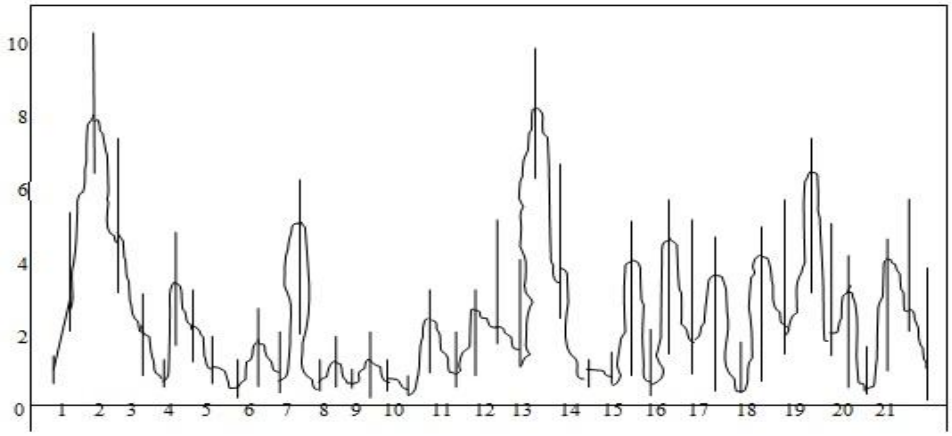
(a) Phase based Model:

Gaffney and Davis [10] proposed the phase based model, which divides the software development cycle into different phases (e.g. requirement review, design, implementation, unit test, software integration, system test, operation etc.) and assumes that code size estimates are available during the early phases follow a Raleigh density function when normalized by the lines of code. The idea is to divide the stage of development along a continuous time (i.e. t=0-1means requirements analysis and so on) and overlay the Raleigh density function with a scale parameter, known as fault discovery phase constant, is estimate by equating the area under the curve between earlier phases with observed error rates normalized by the lines of code. This method gives an estimate of the fault density for any later phase. The model also estimates the number of faults in a given phase by multiplying the fault density by the number of lines of code.

This method is clearly motivated by the corresponding model used in hardware reliability and the predictions are hardwired in the model based on one parameter. In spite of this criticism, this model is one of the first to leverage information available in earlier development life cycle phases.

(b) Predictive Development Life Cycle Model:

In this model the development life cycle is divided into the same phases as in Phase based method. However, it does not postulate a fixed relationship (i.e. Raleigh distribution) between the numbers of faults discovered during different phases. Instead, it leverages past releases of similar products to determine the relationships. The relationships are not postulated beforehand, but are determined from data using only a few releases per product. Similarity is measured by using an empirical hierarchical bays framework. The number of releases used as data is kept minimal and, typically, only the most recent one or two releases are used for prediction. The lack of data is made up for by using as many products as possible that were being developed in a software organization at around the same time. In that sense it is similar to meta analysis [12], where a lack of longitudinal data is overcome by using cross-sectional data.



22 products and their releases versus observed (+) and predicted Fault Density connected by dash lines. Solid vertical lines are 90% predictive intervals for Faulty Density

Conceptually, the two basic assumptions behind this model are as follows that one is "*defect rates from different products in the same product life cycle phase are samples from a statistical universe of products coming from that development organization*" and the second is "*different releases from a given product are samples from a statistical universe of releases for that product".*

Dynamic Models: Reliability Growth Models

Software reliability estimation determines the current software reliability by applying statistical inference techniques to failure data obtained during system test or during system operation. Since reliability tends to improve over time during the software testing and operation periods because of removal of faults, the models are also called reliability growth models. They model the underlying failure process of the software, and use the observed failure history as a guideline, in order to estimate the residual number of faults in the software and the test time required to detect them. This can be used to make

release and development decisions. Most current software reliability models fall into this category. Details of these models can be found in Lyu [9], Musa et al. [8], Singpurwalla and Wilson [13], and Gokhale et al. [14].

Classes of Models:

I am describing a general class of models. In binominal models the total number of faults is some number N; the number found by time t has a binominal distribution with mean $\mu(t) = NF(t)$, where F (t) is the probability of a particular fault being found by time t. Thus, the number of faults found in any interval of time (including the interval (t, ∞) is also binominal. F (t) could be any arbitrary cumulative distribution function. Then, a general class of reliability models is obtained by appropriate parameterization of μ (t) and N.

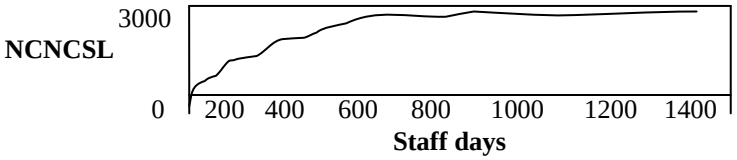
Letting N be Poisson (with some mean υ) gives the related Poisson model; now, the number of faults found in any interval is Poisson, and for disjoint intervals these numbers are independent. Denoting the derivative of F by F′, rate at time t is $F'(t)$ /[1-F(t)]. These models are Markovian but not strongly Markoviaan, except when F is exponential; minor variations of this case were studied by Jelinski and Moranda [15], Shooman [16], Schneidewind [17], Musa [18], Moranda [19], and Goel and okomoto [20]. Schick and Wolverton [21] and crow [22] made F a Weibull distribution; Yamada et al. [23] made F a Gamma distribution; and Littlewood's model [24] is equivalent to aassuming F to be Pareto. Musa and Okumoto [25] assumed the hazard rate to be an inverse linear function of time; for this "Logarithmic Poisson" model the total number of failures is infinite. The success of a model is often judged by how well it fits an estimated reliability curve μ(t) to the observed "number of faults versus time" function.

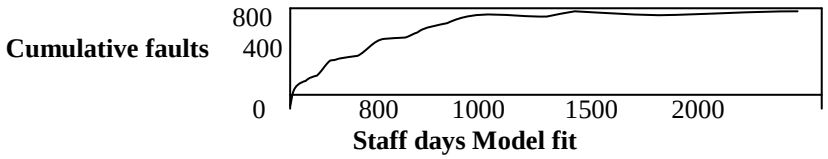
Let us examine the real example plotted in above Figure from testing a large software system at a telecommunications research company. The system had been developed over years, and new releases were created and tested by the same development and testing groups respectively. In this figure, the elapsed testing time in staff day's t is plotted against the cumulative number of faults found for one of the releases. It is not clear whether there is some "total number" of bugs to be found, or whether the number found will continue to increase indefinitely. However, from data such as that in figure, an estimation of the tail of a distribution with a reasonable degree of precision is not possible. I also fit a special case of the general reliability growth model described above corresponding to N being Poisson and F being exponential.

Reliability Growth Modeling:

We have so far discussed a number of different kinds of reliability model of varying degrees of plausibility, including phase-based models depending upon a Raleigh curve, growth models like the Goel-okumoto model, etc. The growth models take us at input either failure time on 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.

Most existing models assume that no explanatory variables are 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 KNCSL, i.e. thousands of non-commentary source lines) there are variables, other than time, that 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; at worst, a poor fitting model with unstable parameter estimates is likely. I briefly describe a general approach proposed by Dalal and McIntosh [28] for incorporating covariates along with a case study dealing with reliability modeling during product testing when code is changing.





As an example, consider a new release of a large telecommunications system with approximately 7 million NCSL and 300 KNCNCSL (i.e. thousands of lines off non-commentary new or changed source lines). 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 thee staff time spent on testing were collected. Above figure portrays the growth of the system in terms of NCNCSL and of faults against staff time. The corresponding numerical data are providing in Dalal and McIntosh [28].

Assume that the testing process is observed at time t_i , i=0, 1,. . ., h, and at any given time the amount of time it takes to find 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 assumption lead to the mass blance 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, 2, 3….h. Note that q represent the number of new faults entering the system per additional NCNCSL, and l_1 represent the number of faults in the code at the starts 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 example, 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 the given above figure (bottom). The fit is evidently very good. Of course, assessing the model on independent or new data is required for proper validation.

Now, I examine the efficacy of creating a statistical model. The estimate of q in the example 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 10000 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 10000 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 10000 NCNCSL entering system test may be too high and a plan ought to be considered to improve the incoming quality.

None of the above conclusion could have been made without using a statistical model. These conclusions are valuable for controlling and improving the process.

Conclusion:

In this paper, I am described key software reliability models for early stages, as well as for the test and operational phases and have given some examples of their uses. I have also proposed some new research directions useful to practitioners, which will lead to wider use of software reliability models.

References:

1. Mars Climate Orbiter Mishap Investigation Board Phase I Report, 1999, NASA.
2. Lee L. The day the phones stopped: how people get hurt when computers go wrong. New York: Donald I. Fine, Inc.; 1992
3. Dalal SR, Horgan JR, Kettenring JR. Reliable software and communication: software quality, reliability, and safety. IEEE J spec Areas Commun 1993; 12; 33-9.
4. Institute of Electrical and Electronics Engineers. ANSI/IEEE standard glossary of software engineering terminology, IEEE Std. 729-1991.
5. ISO 9000-3. Quality management and quality assurance standard- part 3: guidelines for the application of ISO 9001 to the development, supply and maintenance of software. Switzerland: ISO; 1991.
6. Paulk M, Curtis W, Chrises M, Weber C., Capability maturity model for software, version 1.1, CMU/SEI-93-TR-24.Carnegie Mellon University, Software engineering Institute, 1993.

7. Emam K, Jean Normand D, Melo W. Spice: the theory and practice of software process improvement and capability determination. IEEE computer Society Press; 1997.
8. Musa JD, Iannio A, Okumoto K. Software reliability measurement, prediction, application. New York: Mc Grawth-Hill; 1987.
9. Lyu MR, editor. Handbook of software reliability engineering. New York: MC Grawth- Hill; 1996.
10. Gaffney JD, Davis CF. An approach to estimating software errors and availability. SPC-TR-88-007, version 1.0,1988.
11. Dalal SR, and Ho YY. Predicting later phase faults knowing early stage data using hierarchical Bayes models. Technical Report, Telcordia Technologies, 2000.
12. Thomas D, Cook T, Cooper H, Cordray D, Hartmann H, Hedges L, Light R, Louis T, Mosteller F. Meta analysis for explanation: a casebook. New York: Russell Sage Fiundation; 1992.
13. Singpurwalla ND, Wilson SP. Software reliability modeling, Int Stat Rev 1994; 62 (3): 289-317.
14. Gokhale S, Marinos P, Trivedi K. Important milestones in software reliability modeling. In: Proceeding of software Engineering and knowledge Engineering (SEKE 96), 1996.p. 345-52.
15. Jelinski Z, Moranda PB. Software reliability research. In: Statistical computer performance evaluation. New York: Academic Press; 1972. P.465-84.
16. Shooman ML. Probabilistic models for software reliability prediction. In: Statistical computer performance evaluation. New York: Academic Press; 1972. P.485-502.
17. Schneidewind NF. Analysis of error processes in computer software. Sigplan Note 1975; 10(6): 337-46.
18. Mussa JD, A theory of software reliability and its application. IEEE Trans software Eng 1975; SE-1(3): 312-27.
19. Moranda PB. Predictions of software reliability during debugging. In: Proceeding of the Annual Reliability and Maintainability Symposium, Washington, DC, 1975.p 327-32.
20. Goal AL, Okumoto K. Time dependent error detection rate model for software and other performance measures. IEEE Trans Reliab 1979; R-28 (3): 206-11.
21. Schick GJ, Wolverton RW. Assessment of software reliability. In: Proceeding, Operation Research. Wurzburg Wien: Physica Verlag; 1973. P. 395-422.
22. Crow LH. Reliability analysis for complex repairable systems. In: Proschan F, Serfling RJ, editors. Reliability and biometry. Philadelphia: SIAM; 1974.p. 379-410.
23. Yamada S, Obha M, Osaki S. S-shaped reliability growth modeling for software error detection. IEEE Tran Reliab 1983; R-32 (5):475-8.
24. Littlewood B. Stochastic reliability growth: a model for fault removal in computer programs and hardware designs. IEEE Tran Reliab 1981; R-30 (4):313-20.
25. Musa JD, Okumoto K. A logarithmic Poisson executive time model for software reliability measurement. In: Proceeding seventh International conference on Engineering, Orlando (FL), 1984. p.230-8.
26. Miller D. Exponential order statistic models of software reliability growth. IEEE Trans software Eng 1986; SE-12(1):12-24.
27. Gokhale S, Lyu M, Trivedi K. Software reliability analysis incorporating debugging activities. In: Proceeding of International Symposium on software Reliability Engineering (ISSRE 98), 1998.P.202-11.
28. Dalal SR, Mcintosh AM. When to stop testing for large software system with changing code. IEEE Trans software Eng 1994; 20:318-23.