International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

A Comparative Study and an Overview of the Various Software Reliability Growth Models

Shivam Gautam[1], Arti Rana[2]

[1, 2]Amity Institute of Information Technology, Amity University, Noida, UP, India

Abstract: ***Here we investigate the underlying basis linking the software reliability growth models to the software debugging and testing process. It's important for respective reasons. Firstly, if the parameters have an interpretation, then they constitute a metric unit for the software test process and the software under test. Secondly, it may be possible for estimating the parameters even before testing begins. These analytical values can serve as a check for the values ciphered at the beginning of testing, when the test-data is overlooked by short term noise. They can also serve as the initial estimates when iterative computations are used. A Software Reliability Growth Model is one of the fundamental techniques used to evaluate software reliability quantitatively. It is required to have a good performance in terms of goodness-of-fit, certainty and so forth. A number of analytical models have been projected during the past three decades for assessing the reliability of the software system. In this paper, we will summarize some of the existing Software Reliability Growth Models (SRGMs), provide a critical analysis of the inherent assumptions, and assess the applicability of these models during the software development cycle.***

Keywords: Software Reliability Growth Models (SGRMs), times between failures, software reliability, fault, S-Shaped and concave models, the Goel-Okumoto model.

1. Introduction

Software reliability is a critical component of computer system availability, so it is important that customers experience a small number of software failures in their working environments. Software Reliability Growth Model can be utilized as an indication for failures that may be encountered after the software has been installed at client's side and thus as an indication of whether the software is ready to be delivered [5][7]. This method is used for system test data to predict the number of defects remaining in the software. It has been applied to portions of several releases over the past few years. This research has provided some insights into these models related to its stability, predictive ability and their utility [2]. Stability means that the model parameters should not significantly change as new data is added & Predictive ability refer to the number of defects remaining is predicted by the model should be close to the number found in field use. [1]

Software reliability is one of the key issues in modern software product development. Many efforts have been devoted to the study of measuring software reliability quantitatively in the area of software engineering. There are several existing software reliability models, especially applicable to the software testing phase in the software development process, which are of great use to estimate and predict software reliability. During the software testing phase, a software system is tested to detect software errors remaining in the system and correct them. If it is assumed that the correction of errors does not introduce any new errors, the probability that no failure occurs for a fixed time interval, i.e., the reliability, increases with the progress of software testing. A software reliability model describing such an error detection phenomenon is called a software reliability growth model (SRGM).

2. Software Reliability Growth Model (SRGM)

Software reliability can be defined as the probability of failure-free software operation for a specified period of time in a specified environment [13]. In highly complex modern, reliability of the software systems is the most important factor, since it quantifies software failures during the process of software development and software quality control. Software reliability can also be defined for software as the probability of execution without failures for some specified interval of natural units or time. A failure is a departure of system behavior in execution from user requirements and it is the result of a fault. A fault is a defect that causes or can potentially cause the failure when executed. The models applicable to the assessment of software reliability are called SRGMs. [11] [15]

Reliability is usually defined as the probability that a system will operate without failure for a specified time period under specified operating conditions. It is concerned with the time between failures or its failure rate. Defect detection is commonly a failure during a test, but tested software may also detect a flaw even though the test continues to operate in the given circumstances [6]. Defects can also be observed during design or code reviews, but we do not consider those sorts of activities in this report. Time in a test environment is a synonym for amount of testing, which can be measured in several ways. Defect detection data dwells of a time for each defect or group of defects and can be plotted as shown in Figure 1. We can derive defect detection rates from this data. [3]

Licensed Under Creative Commons Attribution CC BY

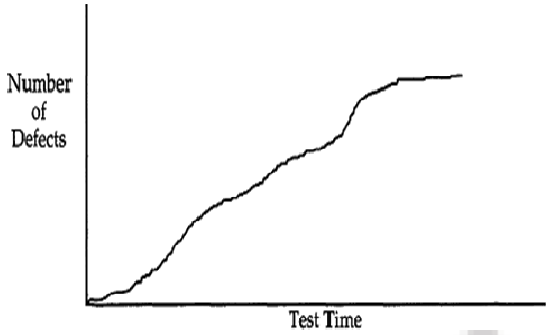

Figure 1 : Example of Defect Detection Data

A cumulative plot of number of defects vs amount of testing such as Figure 1 should show that the defect discovery rate decreases as the amount of testing increases. The theory is that each defect is fixed as it is noticed. This decreases the number of defects in the code, so the length of time between defects breakthroughs should increase [7]. When the defect discovery rate reaches an acceptably low value, the software is deemed suitable to deploy. However, it is difficult to interpolate from defect discovery rate in a test environment to failure rate during system operation, primarily because it is hard to generalize from test time to system operation time. Instead, we look at the expected quantity of remaining defects in our given code. These residuary defects provide an upper limit on the number of unique failures our customers could encounter in field use. [8]

Software reliability growth models are a statistical interpolation of defect detection data by mathematical functions. The programmer's functions are used to anticipate future failure rates or the number of residual defects in the code. Current software reliability literature is inconclusive as to which data representation, software reliability growth model and correlation statistics technique works most beneficial. The advice in the literature seems to be to try a number of the different techniques and see which works best in your environment. [14] [1]

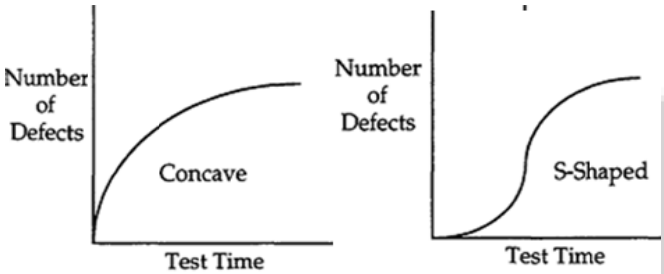

Figure 2 : Concave & S-Shaped Models

There are many different representations of software reliability models. In this paper we use the model representation shown in Figure 2. This representation shows the expected number of defects at time t and is denoted μ(t), where t can be calendar time, execution time, or number of tests executed. An example equation for μ(t) is the Goel – Okumoto model:

$\mu(t) = a(1-e^{-bt})$, where

a = expected total number of defects in the code,

b = shape factor = the rate at which the failure rate decreases, i.e., the rate at which we advance towards the total number of defects. [2] [7]

3. Software Reliability Growth Model Types

Software reliability growth models have been grouped into two classes of models Concave and S-shaped [4]. These two models are shown in Figure 2. The most important thing about both models is that they have the same behavior, i.e., the defect catching rate decreases as the number of defects noticed (and repair) increases and the total count of defects observed asymptotically approaches a finite value. The theory for their asymptotic behavior is that:

1) A finite amount of code should have a finite number of defects. Repair and new functionality may introduce new defects, which may increase the original finite count of defects. Some models explicitly account for new defect introduction during test while others assume they are negligible or handled by the statistical fit of the software reliability growth model to the data. [13]
2) It is assumed that the defect detection rate is proportional to the number of defects in the code. Each time a defect is repaired; there are lesser total defects in the code, so the defect detection rate decreases as the number of defects detected (and repaired) increases. The concave model strictly follows this pattern. In the S-shaped model [4], it assumes that the early testing is not as efficient as later testing, so it has a ramp-up period during which the defect detection rate increases. This could be a good assumption if the first tests are simply repeating tests that developers have already run or if early tests uncover defects in other products that prevents from finding defects in the product being tested. Let's take an example, an application test may uncover as defects that need to be corrected before the application can be run. Application test hours are compiled, but defect data is minimum because as defects don't count as part of the application test data. After the defects are corrected, the remainder of the application test data (after the inflection point in the S-shaped curve) looks like the concave model. [13]

The Goel-Okumoto model is a concave model and the parameter "a" would be plotted as the total number of defects in Figure 2. The Goel-Okumoto model has 2 parameters; other models can have 3 or more parameters. For most models, $\mu(t) = aF(t)$, where a is the expected total number of defects in the code and F(t) is a cumulative distribution function.

Note that $F(0)=0$, so no defects are discovered before the test starts, and $F(\infty)=1$, so $\mu(\infty)= a$ and a is the total number of defects discovered after an infinite amount of testing. [2]

4. Existing SRGMs

A. Times between Failures Models

In this class of models, the process under study is the time taken between the failures. An almost common approach is to presume that the time between, say, the $(t-1)^{st}$ and the t^{th} failures, observes a distribution whose [8] parameters depends on the number of defects remaining in the program during this interval. Estimates of the parameters are obtained

Licensed Under Creative Commons Attribution CC BY

from the observed values of times between failures and estimates for the reliability of the software, mean time to next failure, etc are then obtained from the suited model. Another approach is to treat the failure times as realizations of a stochastic process and use an appropriate time-series model to describe the underlying failure process. [6]

B. Failure Count Models

The interest of this class of models is in the number of faults or failures in specified time intervals rather than in times between failures. The failure counts are presumed to follow a known stochastic process with a time [11] dependent discrete or continuous failure rate. Parameters of the failures can be estimated from the observed values of failure counts or from failure times [3]. Approximates of software reliability, mean time to next failure, etc can again be obtained from the relevant equations.

C. Fault Seeding Models

The basic approach in this class of models is to "seed" a known number of faults in a program which is assumed to have an unknown number of indigenous faults. The program is tested and the observed numbers of seeded and indigenous faults are calculated [15]. From these, an estimate of the fault content of the program prior to seeding is obtained and used to assess software reliability and other relevant measures [6].

D. Input Domain Based Models

The basic approach taken here is to generate a set of test cases from an input distribution which is assumed to be representative of the operational usage of the program [3]. Because of the difficulty in finding this distribution, the input domain of a function is partitioned into a set of equivalence classes, each of which is commonly associated with a program path. An estimate for the reliability of the program is obtained from the failures observed during physical or symbolic execution of the test cases sampled from the input domain [1].

5. Model Assumption

Table 1 list various assumptions underlying the models defined above. Not all of the assumptions listed here are relevant to any given model but, as a totality, they provide an insight into the kind of limitations imposed by them on the use of the software reliability models. It should be pointed out that the arguments presented here are not likely to be universally applicable because the software development process is very environment dependent. What holds true in one environment may not be true in another. Because of this, even assumptions that seem reasonable, e.g., during the testing of one function or system, may not hold true in subsequent testing of the same function or system. The ultimate decision about the appropriateness of the underlying assumptions and the applicability of the models will have to be made by the user of a model. What is presented here should be helpful in determining whether the assumptions associated with a given model are representative of the user's development environment and in deciding which model is to be used by the programmer during the development phase of a software product.

Table 1: List of Key Assumptions of SRGMs

Times Between Failure Models	• Independent times are there in between failures. • Equal probability of exposure to each fault. • Embedded faults are independent from others. • Faults are removed after each of their occurrence. • New faults are included during correction, i.e perfect fault removal method.
Fault Count Models	• Testing intervals are independent from others. • Testing during intervals is fairly homogeneous. • Number of faults observed during non-overlapping intervals is independent from others.
Fault Seeding Models	• Seeded faults are arbitrarily distributed in the program. • Indigenous and seeded faults have an equa probability of being noticed.
Input Domain Based Models	• Input profile distribution is known. • Random testing is used. • Input domain can be zoned into equivalent classes.

6. Conclusions

The concept of learning has been incorporated in the fault detection rate to show the effect of learning of the testing team as the testing grows. The models have been measured, validated, and compared with other existing NHPP models by applying it on two datasets. The results show that the proposed models provides improved goodness of fit for software failure occurrence / fault removal data due to their applicability and flexibility. The concept of change point can be incorporated in the proposed model. In this paper we have taken constant probability of perfect debugging as well as constant error generation however these may vary with time. It provides us new research directions to find more realistic SRGM.

In particular, we have discussed the quantitative measures for software reliability assessment and the maximum-likelihood estimation for data analysis based on SRGM's described NHPP models. The data sets for software reliability analysis, which are observed during the testing phase have been classified into two-types: error-detection count data sets and failure-occurrence time data sets. The SRGMs discussed in this paper are useful primarily in accessing and monitoring software reliability. Such SRGMs should provide software engineers and managers with guidance related to many decision-making problems for successful software development projects. For this purpose, more useful and applicable SRGMs that incorporate information about a software and the development process will be needed.

References

[1] W. Farr, "Software Reliability Modeling Survey, in Handbook of Software Reliability Engineering", Ed. M. R. Lyu, McGraw-Hill, 1996, pp. 71-117.
[2] A.L. Goel, "Software Reliability Models: Assumptions, Limitations and Applicability", IEEE Transactions on Software Engineering; SE-11, pp. 1411-1423, 1985.

Licensed Under Creative Commons Attribution CC BY

[3] T.G. Woodcock and T.M. Khoshogoftaar, "Software Reliability Model Selection: A case study", Proceedings of The International Symposium on Software Reliability Engineering, 1991, pp.183-191.
[4] S. Yamada, M. Ohba, and S. Osaki, "S-Shaped Software Reliability Growth Models and Their Applications", IEEE Transactions on Reliability, R-33, 1984, pp. 169-175.
[5] H. Pham, "System software reliability", Springer series in Reliability Engineering, 2006.
[6] K. Pillai and VSS. Nair, "A model for software development effort and cost estimations", IEEE Transactions on software engineering, vol. 23(8), pp. 485-497, 1997.
[7] L. Goel, "A guidebook for software reliability assessment," Rep. RADCTR-83-176, Aug. 1983.
[8] S. Yamada and S. Osaki, "Software Reliability Growth Modeling: Models and Applications, IEEE Trans. On Software Engineering, vol. SE-11, no. 12, pp. 1431-1437, December1985.
[9] A.L. Goel, V.R. Basili, and P.M. Valdes, "When and how to use software reliability model," in Proc. 7th Software Eng. Workshop, NASA/GSFC, Greenbelt, MD, Nov. 1983.
[10] Y. K. Malaiya, N. Karunanithi and P. Verma, "Predictability of Software Reliability Models", IEEE Trans. Reliability, December 1992, pp. 539-546.
[11] P.K. Kapur, R.B. Garg, "A software reliability growth model for an error removal phenomenon", Software Engineering Journal, Vol. 7, pp. 291-294, 1992.
[12] Huang, Chin-Yu "Software Reliability Analysis by Considering Fault Dependency and Debugging Time Lag" IEEE Transaction on Reliability, Vol.35, No.3 pp.436-449, 2006.
[13] S. Yamada, M. Ohba, and S. Osaki, "S-shaped reliability growth modeling for software error detection," IEEE Trans. Rel., vol. R-32, pp. 475-478, Dec. 1983.
[14] H.S. Kan, Metrics and models in software quality engineering, 2nd edition, Addison-Wesley (2003).
[15] H. Pham, Software Reliability, Springer-Verlag, New York, 2000.

Licensed Under Creative Commons Attribution CC BY