Software Reliability Measurement Experiences Conducted in Alcatel Portugal

Rui Lourenço, Alcatel Portugal, S.A.

Abstract

Software Reliability measurement is essential for examining the degree of quality or reliability of a developed software system. This paper describes the experiments conducted at Alcatel Portugal, concerning the use of Software Reliability models. The results are general and can be used to monitor software reliability growth in order to attain a certain quality within schedule. A method based on the analysis of the trend exhibited by the data collected is used to improve the predictions. The results show that:

- *It is difficult for the models to reproduce the observed failure data when changes in trend do not follow the models assumptions.*
- *The Laplace trend test is a major tool for guiding the partitioning of failure data according to the assumptions of models reliability growth.*
- *Prediction yields good results over a time period of a few months, showing that reliability modeling is a major tool for test/maintenance planning and follow up.*

Introduction

Software reliability models are used to monitor, evaluate and predict the quality of software systems. Quantitative measures provided by these models are a key help to the decision-making process in our organization.
Since a lot of resources are consumed by software development projects, by using software reliability models, our goal is to optimize the use of these resources, in order to achieve the best quality with lower costs and optimized schedules.
They enable us to estimate software reliability measures such as:

- The number of failures that will be found during some time period in the future
- How much time will be required to detect a certain number of failures
- What is the mean time interval between failures and what resources are needed (testing + correction) to achieve a given quality level
- To perform comparative analysis: "how does my product compare with others"

With respect to the software life cycle, the phases requiring careful reliability evaluation are:

- Test: To quantify the efficiency of a test set and detect the saturation instant, i.e., the instant when the probability of test failure detection becomes very low.
- Qualification: To demonstrate quantitatively that the software has reached a specified level of quality.
- Maintenance: To quantify the efficiency of maintenance actions. At the starting of operational life, the software might be less reliable as the operational environment changes. Maintenance actions restore the reliability to a specified level.

Data requirements needed to implement these models

To implement these software reliability models, a process needs to be set up in order to collect, verify and validate the error data needed to be used as an input.
Since we are using a defect management tool to submit, manage and track defects detected during the software development life cycle phases mentioned above, it is therefore relatively easy for us to retrieve error data from the software defects database.
This way, it is possible to collect historical and actual error data from projects, in the form of time intervals between failures and/or number of failures per time unit, as software reliability models usually request.
Normally, the following data needs to be available before we start using the models:

- The fault counts per time unit (where repeated failures are not counted)
- The elapsed time between consecutive failures
- The length of each time unit used
- The effort spent on test per time unit

Modeling approach

The basic approach here is to model past failure data to predict future behavior. This approach employs either the observed number of failures discovered per time period, or the observed times between failures of the software. The models used therefore, fall into two basic classes, depending upon the types of data the model uses:

1. Failures per time period

2. Times between failures

These classes are, however, not mutually disjoint. There are models that can handle either data type. Moreover, many of the models for one data type can still be applied even if the user has data of the other type, applying data transformations procedures.
For example, one of the models we use with more success is the S-shaped (SS) reliability growth model. For this model, the software error detection process can be described as an S-shaped growth curve to reflect the initial learning curve at the beginning, as the test team members become familiar with the software, followed by a growth and then leveling off as the residual faults become more difficult to uncover.
Like the Goel Okumoto (GO) and the Rayleigh models, that we also use very often, they can be classified as Poisson type models (the number of failures per unit of time is an independent Poisson random variable). Their performance depends basically on 2 parameters:

- One that estimates the total number of software failures to be eventually detected.
- Another that measures the efficiency with which software failures are detected.

In order to estimate the models parameters, we use the tool CASRE: Computer-Aided Software Reliability tool. This is a PC based tool that was developed in 1993 by the Jet Propulsion Laboratory for the U.S. Air Force.

Models assumptions

The modeling approach described here is primarily applicable from the testing phase onward. The software must have matured to the point that extensive changes are not being routinely made. The models can't have a credible performance if the software is changing so fast that gathering data on one day is not the same as gathering data on another day. Different approaches and models need to be considered if that is the case.
Another important issue of the modeling procedure, is that we need to know the inflection points, i.e., the points in time when the software failures stop growing and start to decrease. Reliability growth models cannot follow these trend variations, thus our approach consists of partitioning the data into stages subsequent to applying the models. Inflection points are the boundaries between these stages. A simple way to identify inflection points is by performing trend tests, such as the Laplace trend test [3].
The use of trend tests is particularly important for models such as the S-shaped, on which predictions can only be accurate as long as the observed data meet the model assumption of reliability decay prior to reliability growth. The model (S-shaped) cannot predict future reliability decay, so that when this phenomenon occurs, a new analysis is needed and the model must be applied from the time period presenting reliability decay.
However, this is not the only way of looking at the problem. Assuming that the error detection rate in software testing is proportional to the current error content and the proportionality depends on the current test effort at an arbitrary testing time, a plausible software reliability growth model based on a Non-Homogeneous Poisson Process has also been used.

How to obtain predictions of future reliability

In a predictive situation, statements have to be made regarding the future reliability of software, and we can only make use of the information available at that time. A trend test carried out on the available data helps choose the reliability growth model(s) to be applied and the subset of data to which this (or these) model(s) will be applied.
As mentioned before, the models are applied as long as the environmental conditions remain significantly unchanged (changes in the testing strategy, specification changes, no new system installation...).
In fact even in these situations, reliability decrease may be noticed. Initially, one can consider that it is due to a local random fluctuation and that reliability will increase sometime in the near future. In this case predictions are still made without partitioning data. If reliability keeps decreasing, one has to find out why and new predictions may be made by partitioning data into subsets according to the new trend displayed by the data.
If significant changes in the development or operational conditions take place, great care is needed since reliability trend changes may result, leading to erroneous predictions. New trend tests have to be carried out.
If there is insufficient evidence that a different phase in the program's reliability evolution has been reached, application of reliability growth models can be continued.
If there is an obvious reliability decrease, reliability growth model's application has to be stopped until a new reliability growth period is reached again. Then, the observed failure data has to be partitioned according to the new trend.

Number of models to be applied

With respect to the number of models to be applied, previous studies indicated that there are not "universally best" models. This suggests that we try several models and examine the quality of prediction being obtained from each of them and that even doing so, we are not able to guarantee obtaining good predictions.
During the development phases of a running project, it is not always possible to apply several models, because of lack of time, experience, and analytical and practical

tools. Usually people only apply one, two or three models to their data. Analysis of the collected data and of the environmental conditions helps us understanding the evolution of software reliability, and data partitioning into subsets helps us improve the quality of the predictions.

Models calibration and application

The models may be calibrated either after each new observed data (step-by-step) or periodically after observation of a given number of failures, say y, (y-step-ahead). Step-by-step prediction seems more interesting. However, one needs to have a good data collection process set up to implement this procedure, since data might not always be available immediately. In operational life, longer inter-failure times allow step-by-step predictions.
Since we have a database with error data from running projects in our organization (the defects are collected from the test phase onwards), we have a formal procedure to regularly retrieve, analyse and verify this data.
Then we use a periodical approach, to make predictions, which can be summarized as follows:

- Every week, we retrieve error data from the projects we are interested in evaluate software reliability.
- We analyze and validate this data and look for possible trends, in order to select the best data set that could be used for doing predictions.
- If models assumptions are met, we apply the models, validate them and analyze the results they provide.
- Then we collect feedback from people involved in the projects and, if necessary, take actions that help in improving products reliability.

Laplace trend test

The Laplace trend test [3] is used to determine the software reliability trend using data on failures relating to software:

- Time interval between failures, or
- Number of failures per time unit.

This test calculates an indicator u(n), expressed according to the data (time interval between failures or number of failures per time unit). A negative u(n) suggests an overall increase in reliability between data item 1 and data item n. A positive u(n) suggests an overall decrease in reliability between data items 1 and n. Therefore, if we notice an increase (decrease) in u(n) then we have a period of local reliability decrease (growth).
The Laplace trend test is straightforward and much faster to use than models. The reliability study can be stopped at this stage if it is believed that the information obtained has, indeed, answered the proposed questions. Of course, the obtained information is restricted to:

- Increase in reliability,
- Decrease in reliability,
- Stable reliability.

Case study

We are going to apply the previously described methodology to the software validation phase of one of the software projects currently in the maintenance phase in our company.
The project in question is a large telecom network management system, with more than 350 000 source lines of code. The volume and complexity of this software system make it difficult, if not impossible, to eliminate all software defects prior to its operational phase. Our aim was to evaluate quantitatively some operational quality factors, particularly software reliability, before the software package started its operational life.
The software validation phase for this project is a 4-step testing process: 1) integration test, 2) system test, 3) qualification test and 4) acceptance test. The first 3 steps correspond to the test phases usually defined for the software life cycle. Acceptance test consists of testing the entire software system in real environment, which approaches the normal operating environment. It uses a system configuration (hardware and software) that has reached a sufficient level of quality after completing the first 3 test phases described above.
After the validation phase has started, software errors detected by the test team were submitted into the defects database.
Failure reports that had at least one of the following characteristics were rejected:

- Failures not due to software, but data, documentation or hardware
- Reports related to an improvement request
- Results in accordance to specifications
- Reports failures already accounted for

In order to collect the test effort spent per time unit on a given project, we used data existent in another database specially created to collect manpower figures.
Our goal was to evaluate:

- The number of failures that still remained in the software, after half of the test-planned time was completed
- The time-point when 95% of all software failures existing in the software (forecasted by the models) were found
- The mean time between failures achieved by the end of system, qualification and acceptance tests
- The amount of test effort (testing + correction) still needed to achieve the target of 95% of software defects found.

When we first decided to apply these models, we were half the way through the system test phase. At that time we were interested in determine the number of defects

remaining in our application so we could re-evaluate our test strategy. The first approach consisted in considering the entire set of software failures collected up to that time, to model the software reliability.
To meet this goal, we selected a set of models that used failures per time period as an input. The S-shaped (SS) and the Brooks/Motley (BM) models were chosen, independently of the severity (critical, major and minor), of the observed failures.
Figure 1 shows that the models had difficulty in modelling the entire failure process.

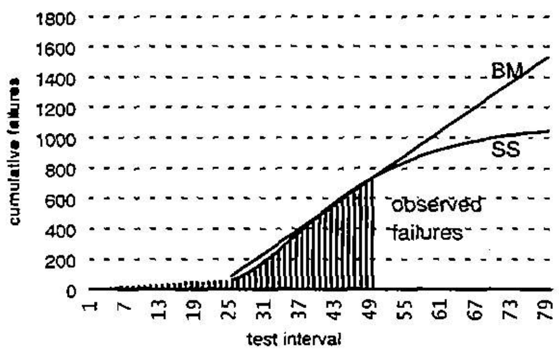

Figure 1: **BM and SS models fitting to the first data set**

Despite the fitting and adjustment problems observed, we can notice two different behaviours in the software models predictions. The SS model presents a more "optimistic" vision of the failure process than the BM. These differences are often observed, and to identify which model was trustworthy for predictions, some expert judgement was needed, since validation statistics, namely the residue and the Chi-Squared statistics, were not enough to help us deciding.
The following table summarizes the models results:

Model	Chi-Squared (degrees of freedom)	Residue	Remaining defects	Time units to achieve 90% defects	Model Rank
BM	87.46 (22)	7.84	76191	n.a.	2
SS	46.64 (21)	6.13	416	18	1

Table 1: BM and SS models results when applied to the first data set collected

Notice that the total number of failures predicted by the BM model was extraordinary big. This didn't mean that we didn't consider this model prediction. Instead of using it's asymptotic measures, we only considered the predictions for the next 20 time units, which revealed more accurate.
After these results have been analysed, the project team agreed that the system test had serious chances to be delayed, so they had to re-think their test strategy.
Later on in the project, right after the qualification test phase has started, the questions were whether if the software would be ready for shipping by the end of this phase and in case it didn't, how much effort (testing + corrections) was still required to achieve the target of 95% of all defects forecasted found. It was an important decision to be made and the conclusions could have serious implications in the project schedule.
In order to improve the accuracy of the new predictions we decided to restrict the data set to be used by applying the Laplace trend test.
As it can be seen in figure 2, the Laplace trend test graphic allowed us to observe periods of local reliability growth and decrease in the data.

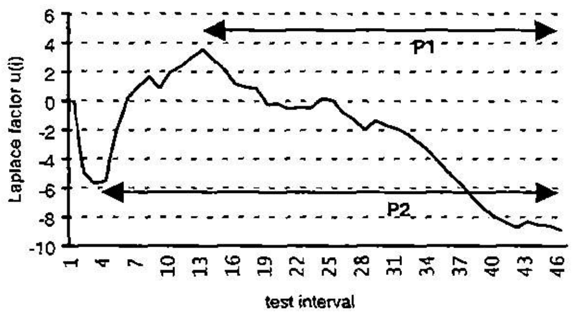

Figure 2: Laplace trend test applied to the observed failure data.

Considering the models assumptions, the periods selected for a new reliability evaluation were P2 for the SS model, since we can notice that there is a decrease in reliability followed by a reliability growth period, and P1 for the GO and BM models, since there is only reliability growth observed.
By running the models again we noticed that the deviation was significantly reduced, thus improving the reliability evaluation (see figure 3).

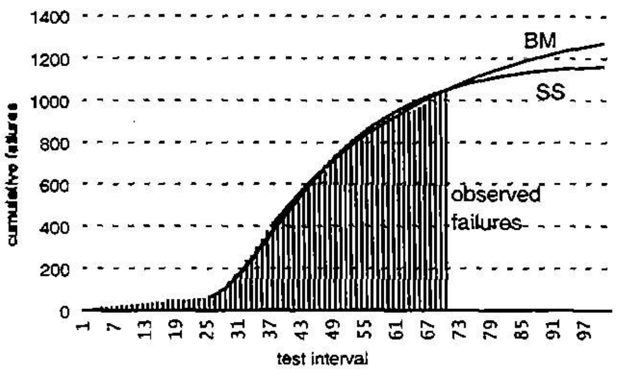

Figure 3: SS and BM models fitting to the data by using P2 and P1 data sets as inputs, respectively

The validation statistics told us that that the observed residues were now lower, which gave us more confidence in the models results.
The following table summarizes the new results observed:

Model	Chi-Squared (degrees of freedom)	Residue	Remaining defects	Time units :o achieve)0% defects	Model Rank
BM	77.6 (31)	5.23	407	42	1
SS	117 (42)	5.79	189	13	2

Table 2: SS and BM models results when applied to P2 and P1 data sets, respectively

Based on these results, plus the expert judgement provided by the project team, we considered the S-shaped model values for reference (optimistic view).
However there was still a question that needed an answer. How much test effort still had to be spent in order the software could be 95% error free? To answer to that question a different model with a different approach was needed. Since test effort is clearly correlated with the defects found during the test phases, we decided to use test effort inputs, in the S-shaped model instead of calendar time.
To include the test effort data in the model, we had to restrict the data range to the period from where we had reliable effort data figures. By doing so, it was possible for us to evaluate with the same model, the remaining failures in the software and the test effort needed to find a given amount of defects in the software system.
We decided to apply this new model (S-shaped modified - SSM) to the data set containing P2, suggested by the Laplace trend test (see figure 2), with a few adjustments in order for the test effort to reflect more accurately the defects found. Figure 4 and table 3 below summarize the results obtained by using this model.

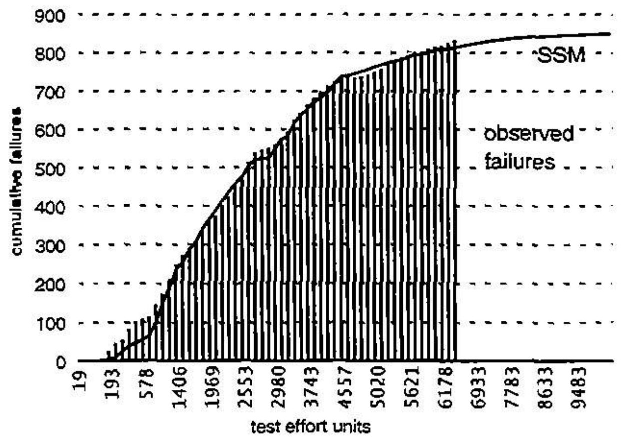

Figure 4: SSM model fitting to P2 data set adjusted

Model	Chi-Squared (degrees of freedom)	Residue	Remaining defects	Test Effort Required to ıchieve)5% defects
SSM	300.74 (39)	5.16	51	623.7

Table 3: SSM model results when applied to P2 data set adjusted

As it can be seen, the model fitting is quite accurate and reasonably adapted to the failure data observed. These results were a major help to the project team, who was able to make more accurate decisions based on the results provided by this model.
As it was mentioned before, expert judgement provided by people from the projects, plays an essential role in the process of deciding which model results to select. Unless we are pretty shore about the stability of our product, i.e., we know that we shouldn't expect too many defects in the near future, and the test environment is not suppose to change much, we can not rely significantly on these results.

Conclusions

Software reliability models are an important aid to the test/maintenance planning and reliability evaluation. However, it is well known that no particular model is better suited for predicting software behaviour for all software systems in any circumstances. Our work helps the already existing models to give better predictions since they are applied to data displaying trends in accordance to their assumptions.

With respect to the application of the proposed method to the failure data of our network management project, 2 models, namely, the S-shaped and Brooks/Motley, have been analysed according to their predictive capabilities. The results obtained show that:

- The trend test help partition the observed failure data according to the assumptions of reliability growth models; it also indicates the segment of data from which the occurrence of future failures can be predicted more accurately;
- The prediction approach proposed for the validation phases yields good results over a time period of a few months, showing that reliability modeling constitutes a major aid tool for test/maintenance planning and follow up.

References

[1] Derriennic H., and Gall G., "Use of Failure-Intensity Models in the Software Validation Phase for Telecommunications", IEEE Trans. on Reliability, Vol. 44, No. 4, December 1995, pp. 658-665.

[2] Goel A.L., and Okumoto K., "Time dependent error detection rate model for software and other performance measures", IEEE Trans. on Reliability, Vol. R-28, No. 3, August 1979, pp. 206-211.

[3] Kanoun K., Martini M.R.B., and Souza J.M., "A Method for Software Reliability Analysis and Prediction Application to the TROPICO-R Switching System", IEEE Trans. on Software Engineering, Vol. 17, No. 4, April 1991, pp. 334-344.

[4] Lyu M.R., "Handbook of Software Reliability Engineering", Published by IEEE Computer Society Press and McGraw-Hill Book Company.

[5] Yamada S., Hishitani J., and Osaki S., "Software Reliability growth with a Weibull Test Effort: A Model & Application", IEEE Trans. on Reliability, Vol. 42, No. 1, March 1993, pp. 100-106.