M. Xie, G. Y. Hong and C. Wohlin, "Modeling and Analysis of Software System Reliability", in Case Studies on Reliability and Maintenance, edited by W. Blischke and P. Murthy, Wiley VHC Verlag, Germany, 2003.

Modeling and Analysis of Software System Reliability

M. Xie[a], G. Y. Hong[b], and C. Wohlin[c]

[a]National University of Singapore, Singapore

[b]Massey University at Albany, Auckland, New Zealand

[c]Blekinge Institute of Technology, Ronneby, Sweden

ABSTRACT

Software is an integral part of many products, ranging from durable goods such as household appliances to large and complex commercial and industrial systems. With the increasing dependency on software in our daily lives, the size and complexity of software systems has grown dramatically. This trend is of great interest to the software industry, calling for more practical analysis and prediction techniques of the reliability of software systems. Various modelling techniques for the analysis of software reliability at different development stages have appeared in the recent literature. This chapter describes the analysis of software failure data with commonly used non-homogeneous Poisson process (NHPP) models to track the reliability growth trend during the testing phase. This chapter deals with the use of available information from earlier releases of a project for early reliability prediction. A detailed case study is presented to illustrate the approach.

1 INTRODUCTION

Software is playing an increasingly important role in our systems. With well-developed hardware reliability engineering methodologies and standards, hardware components of complex systems are becoming more and more reliable. In contrast, such methodologies are still lacking for software reliability analysis due to the complex and difficult nature of software systems. The lack of appropriate data and necessary information needed for the analysis are the common problems faced by software reliability practitioners. In addition, practitioners often find it difficult to decide how much data should be collected at the start of their analysis and how to choose an appropriate software reliability model and method from the existing ones for that specific system.

Here the data is assumed to be failure time data during testing and debugging. The most informative data is the actual time each failure occurred, but grouped data might be sufficient. It could also be useful for detailed analysis if software metrics and other information can be collected during the testing. However, we will not discuss this as their main uses are not for reliability analysis, but for quality and cost prediction.

In a well-defined software development process, data should be collected throughout the software life cycle. These data usually indicate the status and progress of planning, requirement, design, coding, testing and maintenance stages of a software project. Among all these data, failure data collected during the system testing and debugging phase are most critical for reliability analysis. During this period of time, software failures are observed and corrective actions are taken to remove the faults causing the failures. Hence there is a reliability growth phenomenon with the progress of testing and debugging. Appropriate reliability growth models can be used to analyse this reliability growth trend (Musa *et al.*, 1987, Xie, 1991 and Lyu, 1996). The results of the analysis are often used to decide when it is the best time to stop testing and to determine the overall reliability status of the whole system, including both hardware and software. A good analysis will benefit a software company not only in gaining market advantages by releasing the software at the best time, but also in saving lots of money and resources (Laprie and Kanoun, 1992).

However, the applications of software reliability growth models are often limited by the lack of failure data. This is because the traditional approach is based on the goodness of fit of a reliability growth model to the failure data set collected during testing. With a focus on the convergence and accuracy of parameter estimation, the traditional approaches commonly suffer from divergence at the initial stage of testing and usually need a large amount of data to make stable parameter estimates and predictions. In many cases, this means reliability analysis could only be started at a later stage of software testing and results of the analysis might have been too late for any changes in the decisions made.

To overcome this problem, reliability analysis should be started as early as possible and to make this possible, additional information should be used in the analysis until enough data from testing can be collected. It is noted that software systems today are often put on the market in releases. A new release is usually a modification of an earlier one and tested in a similar way. In the current practice, the data collected from the previous releases are often discarded in the reliability analysis of the current release. Although realising that this is wasteful of information, practitioners are uncertain of how to make use of these old data. In our studies, we found that although the two consecutive releases can be considerably different, the information from an earlier release could still provide us with important and useful information about the reliability of a later release from a process development point of view; at least, the information about the testing of an earlier release will help in understanding the testing process that leads to reliability growth in the software system. Thus, a simple approach on early reliability prediction with the use of information from previous releases is introduced in the chapter.

The organization of this chapter is as follows. In Section 2, we first introduce a selection of traditionally used software reliability models with an emphasis on Non-homogeneous Poisson Process (NHPP) models, as they are by far the most commonly used and easily interpreted existing models. In Section 3, a data set from the latest release is introduced and traditional modelling techniques are used for the illustration of their analysis. The convergence and accuracy problems faced by this traditional approach are explored in Section 4. In Section 5, we discuss a new approach that can be taken at a very early stage of software testing for reliability analysis of the current release, making use of the previous release information. A comparison of the two approaches is also conducted as part of the case study. A brief discussion of the conclusions of the case is given in Section 6.

2 NHPP SOFTWARE RELIABILITY GROWTH MODELS

When modelling the software failure process, different models can be used. For an extensive coverage of software reliability models, see Xie (1991) and Lyu (1996), where most existing software reliability models are grouped and reviewed. In this section, we summarise some useful models that belong to the non-homogeneous Poisson process category. Models of this type are widely used by practitioners because of their simplicity and ease of interpretation. Note that here a software failure is generally defined as the deviation from the true or expected outcome when the software is executed.

2.1. Introduction to NHPP Models

Let $N(t)$ be the cumulative number of failures occurred by time *t*. When the counting process $\{N(t); t \geq 0\}$ can be modelled by an NHPP which is a widely used counting process model for inter-event arrival (Xie, 1991), NHPP software reliability models can be used for its analysis. In general, a NHPP model is a Poisson process whose intensity function is time-dependent (Rigdon and Basu, 2000). This type of model is also commonly called the Software Reliability Growth Model (SRGM), as the reliability is improving or the number of failures per interval is decreasing during testing with the discovery and removal of defects found. The expected cumulative number of failures in [0,*t*) can be represented using a so-called mean value function $\mu(t)$. With different mean value functions, we have different SRGMs. The basic assumptions of NHPP SRGMs are:

1. Software failures occur at random, and they are independent of each other;
2. The cumulative number of failures occurred by time *t*, $N(t)$, follows a Poisson process;

Here a software failure is said to have occurred when the output from the software is different from the expected or true one. During the testing and debugging of the software, a software fault is identified after each failure, and the fault caused that failure is removed, and hence the same failure will not occur again. In this way, the failure frequency is reduced and the software can be release when it is expected to be lower than certain acceptable level.

Usually, a software reliability growth model is specified by its mean value function $\mu(t)$ with an aim to analyse the failure data. To determine the mean value function, a number of parameters have to be estimated using the failure data collected. The maximum likelihood estimation method is the most commonly used technique for the parameter estimation of software reliability models, using the failure data, such as the number of failures per interval data or exact failure time. To perform the maximum likelihood estimation of the model parameters, the likelihood function is used:

$$L(n_{1,}...n_m) = \prod_{i=1}^{m} \frac{\{\mu(t_i) - \mu(t_{i-1})\}^{n_i}}{n_i!} \exp\{-[\mu(t_i) - \mu(t_{i-1})]\} \qquad (1)$$

where n_i *(i=1,…, m)* is the observed number of failures in interval $[t_{i-1}, t_i)$ during each of *m* time intervals, with $0 \leq t_0 < t_1 < \ldots < t_m$. In general, to find the maximum likelihood estimates, we take the derivative of the log-likelihood function, equate the derivative to zero, and solve the resulting equation. Often the maximum likelihood equations are complicated and a numerical solution will be possible only using computer programs and libraries.

2.2. Some Specific NHPP Models

The first NHPP software reliability model was proposed by Goel and Okumoto (1979). It has formed the basis for the models using the observed number of faults per time unit. A number of other models were proposed after that, for example, the S-shaped model, the log-power model, and the Musa-Okumoto model, etc. The Goel-Okumoto model has the following mean value function

$$\mu(t) = a(1 - e^{-bt}), \ a>0, b>0 \qquad (2)$$

where $a = \mu(\infty)$ is the expected total number of faults in the software to be eventually detected and *b* indicates the failure occurrence rate per each software fault.

The Goel-Okumoto model is probably the most widely used software reliability model because of its simplicity and the easy interpretation of model parameters to software engineering related measurements. This model assumes that there are a finite number of faults in the software and the testing and debugging process does not introduce more faults into the software.

The failure intensity function $\lambda(t)$ is obtained by taking the derivative of the mean value function, that is:

$$\lambda(t) = \frac{d\mu(t)}{dt} = abe^{-bt} \, . \tag{3}$$

Another useful NHPP model is the S-shaped reliability growth model proposed by Yamada and Osaki (1984), which is also called the delayed S-shaped model. This model has the following mean value function

$$\mu(t) = a(1-(1+bt)e^{-bt}) \, , \, a>0, \; b>0. \tag{4}$$

The parameter *a* can also be interpreted as the expected total number of faults eventually to be detected and the parameter *b* represents a steady-state fault detection rate per fault. This is a finite failure model with the mean value function $\mu(t)$ showing the characteristic of S-shaped curve rather than the exponential growth curve of the Goel-Okumoto model. This model assumes that the software fault detection process has an initial learning curve, followed by growth when testers are more familiar with the software, and then levelling off as the residual faults become more and more difficult to detect.

Xie and Zhao (1993) proposed an NHPP model called Log-Power model. It has the mean value function

$$\mu(t) = a \ln^{b}(1+t) \, , \; a>0, b>0. \tag{5}$$

This model is a modification of the traditional Duane (1964) model. An important property is that the log-power model has a graphical interpretation. If we take the logarithmic on both sides of the mean value function, we have

$$\ln \mu(t) = \ln a + b \ln \ln(1+t)\,. \qquad (6)$$

If the cumulative number of failures is plotted versus the running time, the plot should tend to be on a straight line on a log-log scale. This can be used to validate the model and to easily estimate the model parameters. When the plotted points cannot be fitted with a straight line, the model is probably inappropriate, and if they can be fitted with a straight line, then the slope and intercept on vertical axis can be used as estimates of *b* and ln*a*, respectively.

Note that the log-power model, as well as the Duane model for repairable system, allows $\mu(t)$ to approach infinity. These models are infinite failure models with the assumption that the expected total number of failures to be detected is infinite. This is valid in the situation of imperfect debugging where new faults are introduced in the process of removing the detected faults.

Musa and Okumoto (1984) proposed an NHPP model called the logarithmic Poisson model. It also has an unbounded mean value function

$$\mu(t) = a\ln(1+bt),\ a>0,\ b>0. \qquad (7)$$

This model was developed based on the fact that faults that contribute more to the failure frequency, are found earlier, and it often provides good results in modelling software failure data. However, unlike the log-power model, there is no graphical method for parameter estimation and model validation.

The models introduced in this section are just a glimpse of the many SRGMs that have appeared in the literature. For other models and additional discussion, see Xie (1991). Understanding the software reliability models and their selection and validation is essential for successful analysis (Musa et al., 1987, Friedman and Voas, 1995 and Lyu, 1996).

2.3 Model Selection and Validation

Model selection and validation is an important issue in reliability modelling analysis. However, no single model has emerged to date that is superior to the others and can be recommended to software reliability practitioners in all situations. To successfully apply software reliability modelling techniques, we need to select the model that is the most appropriate for the data set we need to analyse. Goodness-of-fit tests can normally be performed to test the selected model. Some tests are reviewed in Gaudoin (1997). Models can also be compared in order to select a model that is best fit for the data set we want to analyse. On the other hand, as Musa et al (1987) has discussed, there are many practical aspects in model selection.

Models can go wrong in many different ways. Each model has its own assumptions for its validity. To validate a model, we can compare a model prediction with the actual observation that is available later. An appropriate model should yield prediction results within the tolerable upper and lower limits to the users. When a chosen model is proven to be invalid, a re-selection process should begin to find a more appropriate model.

Data collection is also critical for both model selection and validation. Good quality data not only reflect the true software failure detection process but also form the foundation to successful analysis.

3 CASE STUDY

In this section, a case study is presented based on a set of failure data obtained during the system test of a large telecommunication project. The case study is used to illustrate how a traditional NHPP software reliability model introduced in Section 2 is implemented in practice.

The case study is based on the following background. A telecommunication company under study has been collecting failure data for each of its software releases. The data has been collected for documentation purpose without any further analysis to make use of the data. However, they decided, as they started developing release j ($j > 1$) of the system, to try applying software reliability modelling techniques to predict the software system reliability.

Although we have an earlier version of the software that had been tested and failure data recorded for that version, when no such information is available, the standard method assuming unknown parameters can be used. Graphical and statistical methods can be employed for model validation and parameter estimation. The software failure process is then described as failure process similar to any standard repairable system. Hence, in this case study we will focus on the use of the earlier information and assume that the information is made available, which is the case in our case study. On the other hand, the use of traditional approach will be described first.

3.1. Data Set from Release *j*

Table 1 shows the failure data set collected during the system testing of the *jth* release of a large and complex real time telecommunication system. There are 28 software system components with the size of each component between 270 and 1900 lines of code. The programming language used is a company internal language targeted at their specific hardware platform. For each of the components, a problem report is filled in after the detection of a failure during the testing, and corrective actions are taken to modify the code, so similar problems will not be repeated. Each problem report corresponds to the detection of one software failure. By the 28th week of release *j*, the number of problem reports collected from testing was counted and are summarized in Table 1.

Although it is desirable to know the performance of the system as early as possible, at this point the project manager of release *j* is interested knowing how reliable this software system is, based on traditional reliability modelling prediction techniques.

3.2. Analysis of Reliability after the Release

To illustrate how traditional software reliability modelling techniques are used, first a model has to be selected. For its simplicity in illustration, we use the Goel-Okumoto model as an example in this case study. As mentioned, this model is the most widely used NHPP model and the parameters have clear physical meaning.

Second, the parameters of the select software model should be estimated to determine its mean value function $\mu(t)$. The two parameters a and b of the Goel-Okumoto model can be estimated by the maximum likelihood estimation (MLE) method using the following non-linear equations:

$$\begin{cases} \hat{a} = \dfrac{\sum_{i=1}^{k} n_i}{1-e^{-\hat{b}t_k}} \\ \sum_{i=1}^{k} \left(\dfrac{n_i}{e^{-\hat{b}t_{i-1}} - e^{-\hat{b}t_i}} - \dfrac{\sum_{i=1}^{k} n_i}{1-e^{-\hat{b}t_k}}\right)(t_i e^{-\hat{b}t_i} - t_{i-1}e^{-\hat{b}t_{i-1}}) = 0 \end{cases} \tag{8}$$

The second equation can be solved numerically using the data set in Table 1 with *k*=28. Then the estimate of *b* can be inserted into the first equation to obtain an estimate of *a*. Solving the equations, we have

$$\hat{a} = 249.2 \ \ and \ \ \hat{b} = 0.0999 \tag{9}$$

The software reliability model is determined by its mean value function, estimated by

$$\hat{\mu}(t) = 249.2(1-e^{-0.0999t}) \,. \tag{10}$$

With the mean value function, the reliability *R(x|t)* of the software system for a given time interval *(t,t+x)* can be calculated as follows:

$$\hat{R}(x \mid t) = \exp[ae^{-bt}(e^{-bt}-1)] = \exp[249.2 \times e^{-0.0999t}(e^{-0.0999t}-1)] \,. \tag{11}$$

Given *t*=0.2, we get *R*=0.74 at the end of the 28th week.

In addition to reliability, other types of analysis can be conducted. For example, we can determine when to stop testing and release the system based on the criteria of whether certain failure rate level λ_0 has been achieved. To determine the release time τ, we find the smallest *t* that satisfies the following requirement:

$$\lambda(t) = abe^{-bt} \le \lambda_0 \,. \tag{12}$$

That is, $\tau = -\left[\ln \dfrac{\lambda_0}{ab}\right] / b$.

For example, with our estimated model, $\lambda(t) = 249.2 \times 0.0999 \times e^{-0.0999t}$. Suppose that the system can be released when the failure intensity is less than 1. Then $\tau = -\left[\ln \frac{1}{ab}\right] / b = 32.2$. That is, the system can be released after five more weeks of testing.

4. PROBLEMS AND ALTERNATIVES

4.1. Some Practical Problems

In the previous section, we used a case study to show how traditional modelling techniques are used for software reliability analysis. The maximum likelihood method is commonly used to obtain the parameters of the model. However, the MLE requires a large data set for an estimate to be reasonably stable. That is only when the number of failures is large, the estimates will fluctuate slightly around the true value. A problem with the MLE method is that there might be no solution, especially at the early stages of software testing. This problem has been noticed in Knafl and Morgan (1996). A common approach is to wait until we have a large number of failures and the estimation can then be carried out. In the software industry, the availability of a large data set is often an issue. Waiting for a large data set also means reliability analysis could only be done at a later stage of the project development cycle when critical decisions have already been made. On the other hand, we may not know how long we should wait and, when estimates can be obtained, it is possible that after the next interval, there is again no solution to the likelihood equations.

To illustrate the problems encountered in model parameter estimation using the MLE in early stages of testing when only a small number of failures is collected, again we use the data set presented in Table 1. We conducted model parameter estimation from the 11th week onwards. The MLEs using the data collected at the end of each week are given in Table 2. It can be noted that the estimates are not stable, and even may not exist at the beginning. In fact, MLE does not exist before the 12th week. It is also not until the 24th week that the estimates start to stabilise. This is very late in the testing phase and change of any decision-making can be difficult and costly. The release date probably has already been decided by that time.

In fact, it is common that the MLE yields no solution or unstable solutions for model parameter estimations at the early stage of software testing. This is a practical problem that prevents the use of MLE in many cases.

Another problem revealed in the previous case study is perhaps the waste of information. We noticed that the system was developed and tested in a series of releases, which is quite common in the software industry today. However, the data collected from the previous releases are ignored.

To solve these problems, the case of analysing software in previous releases needs to be studied and a procedure for early stage software reliability predictions based on this information is needed. It can be noted here that there are other approaches that could be adopted, such as Bayesian approach. When using a Bayesian approach, prior distributions are needed for the model parameters, and posterior distribution can then be obtained via conditional probability. In the following, we will describe a simple approach that has shown to be useful in our case study.

4.2. The Case with *k* Releases

In fact, nowadays, most large software systems are developed in such a way that each version is a modification of an earlier release. A complete software product thus consists of a series of releases. Traditional software reliability growth models require a large amount of failure data, which is not usually available until the system has been tested for a long time. Although such an analysis is useful for the prediction of field reliability and for the estimation of software failure cost, many software developers would be more interested in estimating the software reliability as early as possible for their planning purpose. In our studies, we found that although two consecutive releases are not the same, some information from the previous release is useful for early reliability prediction of the current release.

Here we are concerned with a number of releases of a certain software system and the underlying trend of software reliability. For the case of k releases, we use N_j ($j=1,2, ..., k$) to denote the total number of faults and use b_j ($j=1,2, ..., k$) to represent the fault detection rate for the *jth* release in the series of k releases. If traditional reliability modelling techniques are employed, failure information of that particular release is used. During the early stage of the software fault detection process, we cannot provide any reliability predictions before enough failure data become available to build a reliability model of the release. However, after studying the characteristics of the fault detection rate, we found that the fault detection rate b_{j-1} of the *(j-1)th* release can be used for the early prediction of the *jth* release, which can be given as

$$b_j(t)= b_{j-1} (t), \quad t>0. \tag{13}$$

An unknown fault detection rate of most of the existing SRGMs always makes the parameter estimation non-linear and requires regression or other estimation techniques. On the other hand, once the fault detection rate b_j is known, determination of an SRGM becomes very straightforward. A procedure for making use of this early information for reliability prediction using Goel-Okumoto model is illustrated in the next section.

Note that equation (13) is an important assumption that should be verified. The justification of that could be subjective by studying the actual testing process as a total change of test personnel or testing strategy will certain lead to different fault detection rate over the release. The verification could also be based on the data assuming both parameters are unknown and estimated without using prior information. However, this will require a large data set. In fact, when we have a sufficiently large data set, the prior information may not be useful and the simple method of using a model and estimation approach as described in Section 3.2 will be sufficient.

4.3. A Proposed Approach

A procedure for accurate software reliability prediction by making use of the information from an early release of the software in the series of *k* releases is presented here. For illustration, we use the Goel-Okumoto model as an example. We know that the current *jth* release being developed has a previous *(j-1)th* release and we have some reliability information about that release. We can apply the following procedures (see Figure 1) for the early reliability prediction for the *jth* release (current release). A similar method was presented in Xie *et al.* (1999).

First, according to the data collected in the previous *(j-1)th* release, a suitable software reliability model $\mu_{j-1}(t)$ is selected**.** Then we apply $\mu_{j-1}(t)$ to the failure data collected from the *(j-1)th* release and estimate the model parameters.

After the *(j-1)th* release, a new release *j* is developed, normally by the same programming group. The traditional approach is to analyse this data set independently to obtain the two parameters in the Goel-Okumoto model, a_j and b_j. However, we found that the fault detection rate is quite stable for the consecutive releases, assuming that the same team is performing the testing. It can be expected that the value of the fault detection rate b_j of the *jth* release is similar to the previous *(j-1)th* release, which is $b_j = b_{j-1}$. Parameter a_j is then estimated as a simple function of the known parameter b_j.

Figure 1 indicates the flow of software reliability modelling and analysis by this sequential procedure. Our focus in this chapter is on the first four boxes. The bottom four boxes will be briefly discussed here.

There are different ways to carry out goodness-of-fit tests. The methods can be different for different models. Standard tests such as the Kolmogorov-Smirnov test, the Cramer-von Mises test and the Anderson-Darling test can be used. If a model is rejected, another reasonable model should be selected. Usually this has to be based on experience and physical interpretation. It is important to also compare with the case when no prior information is used. This is because it is possible for earlier data to be, for example, of different type or based on different failure definition, and hence the proposed approach is no longer suitable.

For illustration of the proposed approach, the same data set used in Section 3 are reanalysed in the following section, making use of additional information from the earlier releases.

5 CASE STUDY (CONTINUED)

In the case study introduced in Section 3, based on the actual failure data given in Table 1, the MLEs of the parameters using the data available after each week were calculated in Table 2. We have discussed the problems associated with this traditional reliability modelling approach in the previous section and proposed a new approach enabling reliability analysis to start as early as possible. In this section, we carry on with the case study, looking at the reliability analysis of release *j* from the early stage of testing based on the information from the previous release *j-1*.

5.1. Data Set from the Earlier Release

The system for which the failure data of release *j* was shown in Table 1, was one of the company's main products. The product went through a couple of releases and the releases under investigation are considered typical releases of this particular system. Prior to release *j*, the failure data of its previous release *j-1* were also collected for documentary purpose. The number of failures detected during the testing of release *j-1* were summarised on a weekly basis as shown in Table 3. The Goel-Okumoto model was chosen again for its reliability prediction. Based on Table 3, the MLEs of parameters a_{j-1} and b_{j-1} of release *j-1* are obtained by solving equation (8). We get

$$\hat{a}_{j-1} = 199.48 \ \ and \ \ \hat{b}_{j-1} = 0.098076 . \tag{14}$$

In the following this assumption will be used. Of course, when a large number of failures have occurred, it is more appropriate to use the complete failure data without assuming any prior knowledge of b. In fact, this was done in section 3.2 and an estimate of b is 0.0999 in equation (9) which is very close to that in equation (14). On the other hand, the justification of making the assumption initially should be based on process and development related information which is the case here; the products were developed in a similar environment and tested with the same strategy and hence the occurrence rate per fault is expected to remain the same.

5.2. Earlier Prediction for Release *j*

Using the procedures discussed in Section 4, we can actually start reliability analysis at a very early stage of release *j* making use of the information from its previous release *j-1*. First, we estimate the two parameters a_j and b_j of the Goel-Okumoto model for release *j*. Note that in this case, we assume that

$$b_j = \hat{b}_{j-1} = 0.098076 \tag{15}$$

The estimate of parameter a_j for release j is obtained by

$$\hat{a}_j = \frac{\sum_{i=1}^{k} n_i}{1 - e^{-b_j t_k}} \,. \tag{16}$$

where n_i *(i=1,…, k)* is the observed number of failures in interval $[t_{i-1}, t_i)$ as before.

Table 4 shows the re-estimation of parameter a_j of release *j* at the end of each week starting from the 11th week by using the parameter $b_j = \hat{b}_{j-1}$ from release *j-1*.

Second, the mean value function $\mu_j(t)$ of release *j* can be easily obtained using equation (2) and Table 4. Actually, with this information available from the *(j-1)th* release, the reliability prediction can be started as early as the second week of the system testing after the failure data were collected for the first week.

5.3. Interval estimation

A comparison of the early reliability prediction approach with the case of traditional approach without using the previous release information is made in this section. We only need to compare the estimates of parameter *a* using these two approaches. However, in addition to the comparison of two point estimates, we also show the 95% confidence intervals of the values of parameter a_j using the proposed early prediction approach. In fact, for practical decision making and risk analysis, interval estimation should be more commonly used as it provides the statistical significance with the values used.

For the interval estimation, we first estimate the parameters a_j and b_j of the Goel-Okumoto model using our proposed early prediction method and can construct 95% confidence intervals for the estimated parameter a_j. To obtain confidence limits, we calculate the asymptotic variance of the MLE of the parameter a_j and we get (Xie and Hong, 2001),

$$1/I(a)=\frac{1}{E[-\frac{\partial^2 \ln L}{\partial^2 a}]}. \tag{17}$$

Here, ln*L* is given by

$$\begin{aligned}\ln L &= \sum_{i=1}^{k}\ln\{\frac{[\mu(t_i)-\mu(t_{i-1})]^{n_i}}{n_i\,!}\exp[-(\mu(t_i)-\mu(t_{i-1}))]\} \\ &= \sum_{i=1}^{k}\{n_i \ln[\mu(t_i)-\mu(t_{i-1})]-[\mu(t_i)-\mu(t_{i-1})]-\ln n_i\,!\}\end{aligned}. \tag{16}$$

Applying this result to the Goel-Okumoto model, we get

$$I(a)=E[-\frac{\partial^2 \ln L}{\partial^2 a}\bigg|_{a=\hat{a}}]=\frac{\sum_{i=1}^{k} n_i}{\hat{a}^2}. \tag{17}$$

For a given confidence level α, the two-sided confidence intervals for parameter *a* is

$$a_L=\hat{a}-\frac{z_{\alpha/2}}{\sqrt{I(\hat{a})}},\quad \text{and}\quad a_L=\hat{a}+\frac{z_{\alpha/2}}{\sqrt{I(\hat{a})}}. \tag{18}$$

where $z_{\alpha/2}$ is the [100(1-α)/2]th standard normal percentile. Given α=0.05, for example, we get $z_{\alpha/2}=z_{0.025}=1.96$.

The 95% confidence intervals for the prediction of parameter a_j using parameter $b_j=b_{j-1}$ are shown in Table 5. Interval estimation provides more information than a point estimate and it can help the decision maker to consider the risk when a bound is to used for further analysis or resource allocation.

Also, from Table 5, we can see that the 95% confidence intervals are very wide and this is because of the limited amount of data and the variability of the failure process. In fact, this is a typical situation in the analysis of software failure data, especially when only a small number of failures have occurred. When the testing is continued and more failures are observed, the confidence interval will become narrower.

6 DISCUSSION

Early software reliability prediction has attracted great interests from software managers. Most large software systems today are developed in such a way that they are a modification of an earlier release. Although two releases of software are not the same, some information should be useful for the predictions. We know that the common way of estimating parameter *b* of the Goel-Okumoto model usually requires programming and only numerical solutions can be obtained. This means obtaining an analytical solution for parameter *b* is not practical for many software managers. The proposed method of estimating the model parameter by making use of the information of a previous release solves this problem nicely. Two case studies were presented and compared in this chapter.

However, there are also some limitations. The usage of early information for reliability prediction is based on the assumption that the testing efficiency is the same and the current software release can be analysed using the same type of reliability models as the prior release. When these assumptions are not satisfied, the method proposed in this chapter is not longer applicable.

References:

Blischke, W.R. and Murthy, D.N.P. (2000). *Reliability; Modeling, Prediction and Optimization.* Wiley, New York.

Duane, J.T. (1964). Learning curve approach to reliability monitoring. *IEEE Transactions on Aerospace,* **2**, 563-566.

Friedman, M.A. and Voas, J.M. (1995). *Software Assessment: Reliability, Safety, Testability*. Wiley, New York.

Gaudoin, O. (1998). CPIT goodness-of-fit tests for the power-law process. *Communications in Statistics – A: Theory and Methods*, 27, 165-180.

Goel, A.L. and Okumoto, K. (1979). Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability*, **28**, 206-211.

Knafl, G.J. and Morgan, J. (1996). Solving ML equations for 2-parameter Poisson-process models for ungrouped software-failure data. *IEEE Transactions on*

Reliability, **45**, 42-53.

Laprie, J.C. and Kanoun, K. (1992). X-ware reliability and availability modelling. *IEEE Transactions on Software Engineering*, **18**, 130-147.

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

Musa, J.D., Iannino, A. and Okumoto, K. (1987). *Software Reliability: Measurement, Prediction, Application*, McGrow-Hill, New York.

Rigdon, S.E. and Basu, A.P. (2000). *Statistical Methods for the Reliability of Repairable Systems*. Wiley, New York.

Xie, M. (1991). *Software Reliability Modelling*. World Scientific Publisher, Singapore.

Xie, M. and Hong, G.Y. (2001). Software reliability modeling, estimation and analysis. In *Handbook of Statistics 20: Advances in Reliability* (N. Balakrishnan and C.R. Rao, Eds), Elsevier, London.

Xie, M., Hong, G.Y. and Wohlin, C. (1999). Software reliability prediction incorporating information from a similar project. *Journal of Systems and Software*, **49**, 43-48.

Xie, M. and Zhao, M. (1993). On some reliability growth-models with simple graphical interpretations. *Microelectronics and Reliability*, **33**, 149-167.

Yamada, S. and Osaki, S. (1984). Nonhomogeneous error detection rate models for software reliability growth. In *Stochastic Models in Reliability Theory*, Springer-Verlag, Berlin, pp.120-143.

Yang. B. and Xie, M. (2000). A study of operational and testing reliability in software reliability analysis. *Reliability Engineering and System Safety*, **70**, 323-329.

Exercises

1. It is possible to use an S-shaped NHPP model for the data set in Table 1. Use this model and estimate the model parameters. Discuss the pros and cons of this model.

2. The Duane model is widely used for repairable system reliability analysis and software system reliability during the testing can be considered as a special case. Use the Duane model and discuss the estimation and modelling issues. In particular, explain if this model should be used.

3. Derive the MLE for the log-power model. Note that you can get an analytical solution for both parameters.

4. The Goel-Okumoto model, although commonly used, assumes that each software

fault contributes the same amount of software failure intensity. Discuss this assumption and explain how it can be modified.

5. Software reliability model selection is an important issue. Study the failure intensity function of the Goel-Okumoto model and log-power model and give some justification for using each of them in different cases.

6. Use the MLE and the graphical method for the log-power model to fit the data set in Table 1. Compare the results.

7. Discuss how the proposed method of earlier estimation in Section 4 can be used for more complicated NHPP models, such as those with three parameters.

8. What is the estimated intensity function at the time of release for release *j*? You may use the Goel-Okumoto model with the estimated parameter in (9).

9. Discuss how the intensity function can be used to determine the release time. For example, assume that there is a failure rate requirement,.that is, a requirement that the software cannot be released before the failure intensity reaches a certain level.

10. When prior information is to be used, an alternative approach is to use Bayesian analysis. Comment on the selection an of appropriate probability model, including prior distributions, in the context of the available information, and discuss the application of the Bayesian methodology in this type of study.

Table 1 Number of failures per week from a large communication system

Week	Failures	Week	Failures	Week	Failures	Week	Failures
1	3	8	32	15	7	22	3
2	3	9	8	16	0	23	4
3	38	10	8	17	2	24	1
4	19	11	11	18	3	25	2
5	12	12	14	19	2	26	1
6	13	13	7	20	5	27	0
7	26	14	7	21	2	28	1

Table 2 The ML estimates of the model parameter for Release *j*.

(Note that prior to the 12th week, the ML estimates do not exist)

Week	$\hat{a}_j$	$\hat{b}_j$	Week	$\hat{a}_j$	$\hat{b}_j$
11	NA	NA	20	NA	NA
12	211.3	0.180	21	276.8	0.0771
13	NA	NA	22	269.5	0.0819
14	NA	NA	23	NA	NA
15	NA	NA	24	261.9	0.0878
16	269.4	0.0924	25	259.6	0.0896
17	483.5	0.0335	26	256.3	0.0923
18	NA	NA	27	250.2	0.0991
19	NA	NA	28	249.2	0.0999

Table 3. Number of failures per week from the previous release (*j-1*).

Week	Failures	Week	Failures	Week	Failures	Week	Failures	Week	Failures
1	2	11	17	21	1	31	0	41	0
2	11	12	31	22	1	32	0	42	0
3	18	13	8	23	1	33	0	43	1
4	10	14	7	24	0	34	0	44	1
5	12	15	10	25	1	35	1	45	0
6	4	16	2	26	1	36	0	46	0
7	28	17	2	27	0	37	1	47	1
8	6	18	0	28	0	38	0	48	0
9	7	19	3	29	0	39	0	49	0
10	6	20	2	30	1	40	0	50	1

Table 4 Parameter a_j estimation assuming $b_j = \hat{b}_{j-1}$

Week	$a_j\ (b_j = \hat{b}_{j-1})$	Week	$a_j\ (b_j = \hat{b}_{j-1})$
11	262.12	20	264.58
12	270.32	21	268.20
13	269.23	22	264.58
14	269.20	23	261.39
15	270.01	24	258.57
16	262.70	25	256.05
17	258.86	26	253.82
18	256.97	27	251.83
19	254.48	28	250.05

Table 5 95% confidence intervals for parameter a_j assuming $b_j = b_{j-1}$

Week	$a_j\,(b_j = \hat{b}_{j-1})$	a_{jU}	a_{jL}	$\hat{a}_j$ (Traditional)
12	270.32	309.07	219.29	211.32
14	269.20	306.41	231.58	N.A.
16	262.70	298.40	231.98	269.42
18	256.97	291.94	226.99	N.A.
20	254.48	289.84	222.46	N.A.
22	264.58	298.49	230.68	266.51
24	258.57	291.69	225.44	260.22
26	253.82	286.34	221.30	257.39
28	250.05	282.09	218.01	249.22

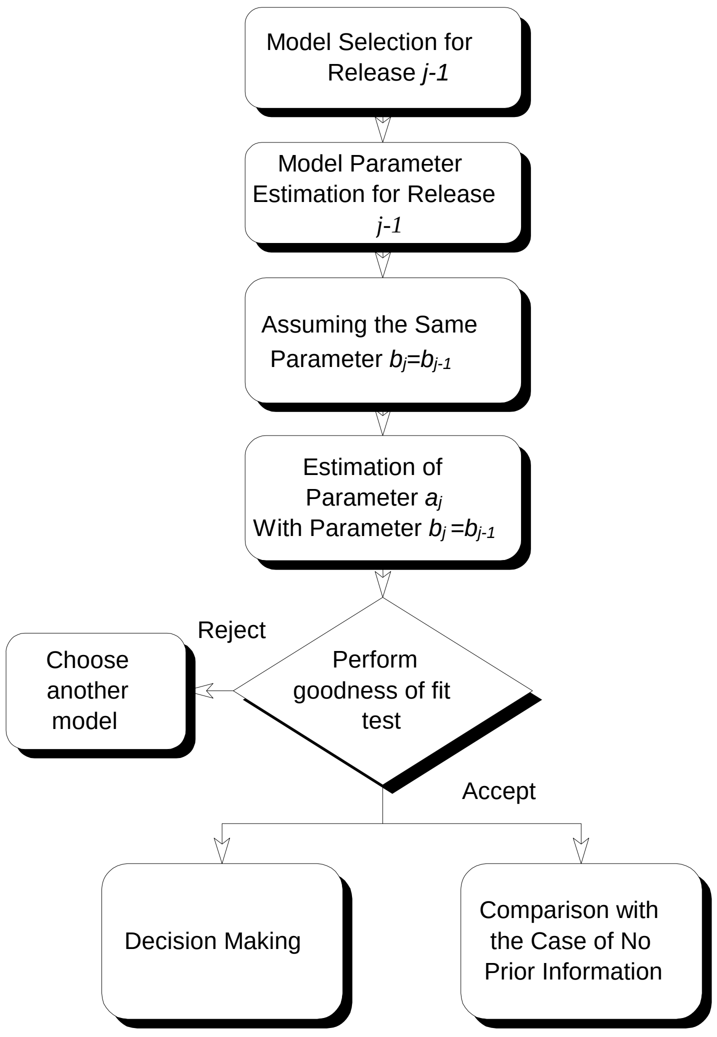
Model Selection for Release *j-1*
Model Parameter Estimation for Release *j-1*
Assuming the Same Parameter $b_j = b_{j-1}$
Estimation of Parameter a_j With Parameter $b_j = b_{j-1}$
Perform goodness of fit test
Reject
Choose another model
Accept
Decision Making
Comparison with the Case of No Prior Information

Figure 1 Procedures of using early information in reliability prediction