I²SDS
The Institute for Integrating Statistics in Decision Sciences

Technical Report TR-2010-17
December 16, 2010

Software Reliability

Refik Soyer
Department of Decision Sciences
The George Washington University

Software Reliability

Refik Soyer
Department of Decision Sciences
The George Washington University

Abstract

In this article we present an overview the state of the art in software reliability. We present some of the traditional software reliability models as well as recent advances in modeling. In so doing, we discuss use of hidden Markov models, as well as nonparametric models including mixtures of Dirichlet processes. Furthermore, we review decision problems in software reliability such as testing strategies and optimal stopping rules. We discuss computational issues associated with use of the models, their statistical analyses and development of optimal strategies.

1 Introduction and Overview

Since the publication of the software reliability model of Jelinski and Moranda (1972), software reliability has gained much attention in the operations research (OR), statistics and computer science communities. Availability of software reliability data sets by Musa (1979) tremendously helped involving researchers from OR and statistics in software reliability modeling and analysis.

In this overview, we will focus on two issues that are of interest to statisticians, applied probabilists and operations researchers in general. These are: (i) how to model and how to describe the changes in the performance of the software as a result of the modifications made to it during testing; (ii) how to decide when to terminate the testing/modification process and to release the software. The former involves stochastic modeling and statistical analysis and is referred to as software reliability modeling whereas the latter involves decision analysis and optimization and is referred to as optimal stopping problem; see Barlow and Singpurwalla (1985). As will be discussed in the sequel, the two issues are related and they both require use of computational methods. There are other issues and problems that are relevant to the above, but they will not be a part of our coverage. Books by Musa, Iannino and Okumoto (1987) and Singpurwalla and Wilson (1999) are good references on these for the interested reader.

Our coverage will start with some overview of software reliability where we introduce concepts and definitions and describe the software failure mechanism. Once this is done we next focus on software reliability

modeling where we present different classes models such as parametric and nonparametric models. Our review will include traditional models such as Jelinski and Moranda (1972) model as well as more recent advances in modeling such as hidden Markov models, mixtures of Dirichlet processes and nonparametric point process models. One of these recent models will be highlighted in our presentation. In so doing, computational issues in statistical inference and stochastic modeling will be discussed.

The last part of our review will address the optimal stopping (or optimal release) problem where we discuss both the earlier approaches as well as more recent advances which are based on decision-theoretic approaches and Bayesian viewpoint. Computational difficulties that arise in the solution of the decision problems will be pointed out. Details of one decision model will also be highlighted in our coverage in this part.

2 Concepts and Definitions

Software reliability is a measure of the quality of a piece of software and is defined as the "probability of failure-free operation of a computer program in a specified environment for a specified period of time"; see for example Musa and Okumoto (1984). The specified environment is usually referred to as the operational profile [Musa (1993)] and the specified time is called the the mission time.

Although they have similar definitions, the causes of software failure are different from those of hardware. As noted by Singpurwalla and Soyer (1996), whereas hardware reliability tends to decrease with mission time, software reliability can, in principle, be 100% reliable for any mission time. Failure of software is due to errors referred to as bugs which are caused by human error in the logic of a software code. This is unlike hardware failures which are typically caused by material defects, and/or wear, and aging. Singpurwalla and Soyer (1996) point out that, although every software failure is caused by a bug, not every bug in the program will lead to a software failure. Thus, as suggested by Jelinski and Moranda (1972) it is important to distinguish between software bugs and software failures. Such a distinction has motivated models for software failures rather than software bugs in the literature.

As noted by Singpurwalla and Soyer (1996) in modeling software failures "it is the uncertainty about the presence, the location and the encounter with a bug that induces randomness." The most commonly used random variable used for modeling software reliability pertains to the times between software failures. It is motivated by the notion that the arrival times, to the software, of the different input types are random. Such inputs which traverse through their designated paths in the software will produce desired outputs; but those which do not, due to bugs in the software will produce erroneous outputs. For assessing software reliability, one observes T_1, T_2, ..., T_i, where T_i is the time between the $(i-1)^{st}$ and the i^{th}software failure. With this conceptualization, even though the failure of software is not generated stochastically, the detection of errors is stochastic, and the end result is that there is an underlying random process that governs the failure

characteristics of software.

Most of the well-known models for assessing software reliability are centered around the interfailure times T_1, T_2, ..., or the point processes that they generate; see Singpurwalla and Wilson (1994). In addition to being used in reliability monitoring, these models can also be used in determining optimal release time for software during the development phase; see for example, Singpurwalla (1991), Daid and Wilson (2001) and Morali and Soyer (2003). As noted by Singpurwalla and Wilson (1994), the software reliability models can be classified into two categories: those that model the times between successive software failures (Type I models) and those that model the number of software failures up to a given time (Type II models). In all these models, time is typically taken to be CPU time.

In the Type I category modeling of T_i's is often accomplished via a specification of the failure rate of the T_i's. More specifically, the failure rate $\lambda_{T_i}(t)$ is specified for all i and this yields different probability models for T_i's. These failure rates may be thought of as the rate at which errors are detected in the software. During the development phase, after each failure, the software is debugged with the hope of discovering and correcting the bugs. Thus, it is expected that the successive failure rates decrease over time and this process is referred to as reliability growth; see Sinpurwalla and Soyer (1985) and Erkanli, Mazzuchi and Soyer (1998). The models that descibe this notion are known as perfect debugging models. Majority of the earlier models in the literature were based on this assumption. However, it is possible that an attempt to debug the software may result in introduction of new bugs and therefore may deteriorate the reliability of the software. Models reflecting this scenario are known as imperfect debugging models; see Ruggeri and Soyer (2008). The Type II models are point processes generated by the interfailure times and they descibe the number of failures during a specified time period. More specifically, $N(t)$, number of software failures during $[0, t)$, is described by a Poisson process with mean value (or cumulative intensity) function $\Lambda(t)$.

3 Parametric Software Reliability Models

We give an overview of the more traditional Type I and II software reliability models next. It is important to note that these are parametric models. Nonparametric software reliability models will be discussed in a later section.

3.1 Modeling Times between Successive Failures

Although the first software reliability model was introduced by Hudson (1967) [see Musa, Iannino and Okumoto (1987)], the model proposed by Jelinski and Moranda (1972) was the first widely known and used software model in the literature. The Jelinski and Moranda (JM) model is a bug counting Type I model. The model assumes that the software initially contains an unknown number of bugs, say N, and after each

failure one bug is discovered and corrected. The failure rate during the i^{th} stage of testing, that is, failure rate of T_i is assumed to be constant and proportional to $(N - i + 1)$, number of bugs remaining in the program. Thus, the JM model is a perfect debugging model where time between failures are conditionally independent exponentially distributed random variables, that is,

$$f_{T_i}(t|N, \phi) = \phi(N - i + 1)e^{-\phi(N-i+1)t} \tag{1}$$

where parameter ϕ denotes the common contribution of each bug to the failure rate. We denote the exponential model (1) as $(T_i|N, \phi) \sim Exp(\phi(N - i + 1))$. An alternative shock model interpretation of the JM model is given by Langberg and Singpurwalla (1985). As shown by Forman and Singpurwalla (1977), statistical analysis of the JM model via maximum likelihood methods may provide misleading results. Bayesian analyses of the JM model were considered by Meinhold and Singpurwalla (1983a) and Kuo and Tang (1995). An extension of the JM model was considered in Goel and Okumoto (1978) who modified the model as $(T_i|N, \phi, q) \sim Exp[\phi(N - q(i - 1))]$ where q denotes the probability of fixing a bug. Thus, the model is capable of reflecting the imperfect debugging scenario where the case of $q = 1$ represents the JM model. Other extensions of the JM model include the nonconstant failure rate model of Schick and Wolverton (1978) where T_i's were assumed to have a Rayleigh distribution.

Littlewood and Verrall (1973) directly modeled times between software failures by allowing the failure rate of the software to vary from one stage to another. In so doing, they imposed a stochastically decreasing ordering on the failure rates of successive stages, citing the fact that it is always the programmer's intent to improve the software when correcting an error. More specifically, they assumed that given the failure rate λ_i at stage i, T_i is an exponentially distributed random variable, that is,

$$f_{T_i}(t) = \lambda_i e^{-\lambda_i t}, \tag{2}$$

where λ_i's are stochastically decreasing. This was achieved by assuming that λ_i's are conditionally independent gamma random variables with shape parameter α and scale $\Psi(i)$, where Ψ is a monotonically increasing function of i. In other words, we have

$$p(\lambda_i|\ \alpha, \Psi(i)) \propto \lambda_i^{\alpha-1}\, e^{-\Psi(i)\lambda_i} \tag{3}$$

implying that $E[\lambda_i|\alpha, \Psi(i)] = \alpha/\Psi(i)$ is decreasing in i. We will denote the gamma model (3) as $(\lambda_i|\ \alpha, \Psi(i)) \sim Gam(\ \alpha, \Psi(i))$. It follows from the above that in the Littlewood-Verrall (LV) model, the reliability function for T_i is given by the Pareto distribution

$$R_{T_i}(t|\alpha, \Psi(i)) = \left[\frac{\Psi(i)}{\Psi(i)+t}\right]^{\alpha}. \tag{4}$$

Littlewood and Verrall (1973) considered the case where $\Psi(i)$ is unknown and use of goodness-of-fit tests to determine the form of $\Psi(i)$. The authors considered specific cases for $\Psi(i)$ such as $\Psi(i) = \beta_0 + \beta_1 i$ and developed estimates of unknown parameters of $\Psi(i)$ and α.

Mazzuchi and Soyer (1988) introduced a hierarchical Bayes extension of the LV model by pointing out that the LV model leads us to an empirical Bayes setup in the sense of Morris (1983). They presented a fully Bayes analysis of the LV model in the sense of the Deely and Lindley (1981) by using a Bayes empirical Bayes (EB) setup. The hierarchical Bayes setup of Mazzuchi and Soyer (1988) assumes exchangeability of the failure rates λ_i's over the stages. The exchangeability of λ_i's was motivated via the argument that it was often "not possible to assign a pattern to the λ_i's." Thus, the authors considered the λ_i's as being generated from some distribution, more specifically, a gamma density

$$p(\lambda_i|\ \alpha, \beta) \propto \lambda_i^{\alpha-1}\, e^{-\beta\lambda_i} \tag{5}$$

with unknown parameters shape α and scale β, that is, $(\lambda_i|\ \alpha, \beta) \sim Gam(\alpha, \beta)$. As noted by Mazzuchi and Soyer (1988), the assumption that λ_i's over all stages constitute a random sample from the gamma distribution in (5) yields an EB setup. The hierarchical Bayes or the Bayes empirical Bayes setup is obtained by assuming a distribution, say $p(\alpha, \beta)$, for describing uncertainty about α and β. Mazzuchi and Soyer (1988) also considered a hierarchical Bayes (HB) setup of the LV model by assuming a specific form $\Psi(i) = \beta_0 + \beta_1 i$ for the scale parameter of the gamma density in (3) and specifying a joint prior distribution for the unknown parameters α, β_0 and β_1. Bayesian analysis of the HB exchangeable model and HB version of the LV model were developed by using posterior approximation methods proposed by Lindley (1980) and Tierney and Kadane (1986). Extensions of the hierarchical model was considered by Kuo and Yang (1995) who assumed a k^{th} order polynomial for $\Psi(i)$ and used Gibbs sampling [see Gelfand and Smith (1990)] for Bayesian computations.

Another class of Type I models includes Markovian models where the Markov dependence was assumed either for the T_i's as in Singpurwalla and Soyer (1985, 1992) or on the λ_i's as in Chen and Singpurwalla (1994) and Basu and Ebrahimi (2003). The former approach proposed by Singpurwalla and Soyer (1985) involves a multiplicative dynamic autoregressive process for T_i's which were assumed to be lognormally distributed. The model implies that $logT_i$'s are normally distributed given a dynamic autoregressive coefficient θ_i and previous failure time, that is,

$$logT_i|T_{i-1}, \theta_i \sim N(\theta_i logT_{i-1}, \sigma_1^2). \tag{6}$$

The model in (6) is capable of reflecting both reliability improvement and reliability deterioration from one testing stage to another. For example, assuming that T_i's are scaled so that $T_i \geq 1$ for all stages, values of $\theta_i > 1$ implies a growth in reliability from stage $(i-1)$ to i. However, a value of $\theta_i < 1$ implies a deterioration from $(i-1)$ to i. Two alternative dependence structures for θ_i's were considered in Singpurwalla and Soyer (1985). In the first model, it was assumed that θ_i's constitute an exchangeable sequence. This was achieved by assuming that given an unknown mean λ, θ_i's are conditionally independent random quantities denoted as $(\theta_i|\lambda) \sim N(\lambda, \sigma_2^2)$ where λ itself was coming from a normal distribution with known parameters μ_λ and σ_λ^2, that is, $\lambda \sim N(\mu_\lambda, \sigma_\lambda^2)$. Alternatively, θ_i's can be assumed to follow a Markovian structure as

$$\theta_i|\theta_{i-1} \sim N(\theta_{i-1}, \sigma_2^2) \tag{7}$$

implying a steady model in the sense of West and Harrison (1997). Bayesian analysis of the both models can be developed using a Kalman filter setup [see Meinhold and Singpurwalla (1983b)] by using (6) as an observation equation and using either (7) or its exchangeable version as the state equation. Details of the Bayesian analysis of the model and an adaptive Kalman filter extension of it can be found in Singpurwalla and Soyer (1992).

A non-Gaussian Kalman filter model was proposed by Chen and Singpurwalla (1994) where the observation model was specified as $T_i|\alpha, \lambda_i \sim Gam(\alpha, \lambda_i)$ with a dynamic scale parameter. The scale parameter λ_i has a Markovian evolution given by

$$\lambda_i = \frac{\lambda_{i-1}}{\gamma}\epsilon_i \tag{8}$$

where $\epsilon_i|D_{i-1}$ has a beta distribution, $D_{i-1} = (D_{i-2}, T_{i-1}),$ and $0 < \gamma < 1$ is a discount parameter. The model is a member of the conjugate class of distributions considered by Bather (1965). The above implies a random walk type of evolution, that is,

$$E[\lambda_i|\lambda_{i-1}, D_{i-1}] = \lambda_{i-1} \tag{9}$$

and provides a stochastic ordering of λ_i's. More specifically, the evolution equation (8) implies that $\lambda_i < (\lambda_{i-1}/\gamma)$. Therefore, the model can capture both reliability growth and decay scenarios. Other prior properties of the model and posterior updating results conditional on γ are given in Miller and Smith (1986). A special case of the above where $\alpha = 1$ was considered by Morali and Soyer (2003) who developed optimal software testing strategies.

An alternative Markovian structure was considered in Basu and Ebrahimi (2003) who proposed a martingale process [see also Koch and Spreij (1983)] for the failure rates λ_i. Given λ_i's, T_i's, time between software failures, were assumed to be conditionally independent exponential random variables as in (2). In order to capture both reliability growth and decay, the authors specified the prior model for $(\lambda_1, \lambda_2, \ldots)$ as

a martingale by assuming

$$\lambda_1|\alpha,\beta \sim Gam(\alpha,\beta)$$

$$\lambda_i|\alpha,\lambda_{i-1} \sim Gam(\alpha,\frac{\alpha}{\lambda_{i-1}}),\ i>0. \tag{10}$$

The prior conditional mean is

$$E[\lambda_i|\lambda_{i-1}] = \lambda_{i-1} \tag{11}$$

which gives the martingale property. The Bayesian analysis of the model was developed using Markov chain Monte Carlo (MCMC) methods such as the Gibbs and Metropolis sampling; see Chib and Greenberg (1995).

Another class of models for time between software failures is the hidden Markov models (HMM); see Soyer (2011) for a general review of HMMs in reliability analysis. The HMMs have gained much attention recently in the software reliability literature. An earlier use of these models is due to Durand and Gaudoin (2005) who considered exponentially distributed T_i's whose failure rates is governed by a discrete time Markov chain (MC). Since the states of the MC is latent the underlying model is referred to as a HMM. Durand and Gaudoin (2005) analyzed the HMM using a non-Bayesian approach based on EM algorithm to obtain maximum likelihood estimates.

More recently, Pievatolo, Ruggeri and Soyer (2010) introduced a Bayesian HMM motivated by potential introduction of new bugs to the software during the software debugging process. As noted by the authors, the model takes into account the possibility of not knowing if a new bug has been added at each stage and allows for inference about introduction of new bugs at different stages of testing. More specifically, Pievatolo, Ruggeri and Soyer (2010) assume that during the testing stages, the failure rate of the software is governed by a latent process Y. If Y_i denotes the state of the latent process at stage i, then given the state at time i, T_i is assumed to follow a exponential model given by

$$T_i|Y_i \sim Exp[\lambda(Y_i)]. \tag{12}$$

In the above, the failure rate of the software depends on the latent random variable. The authors point out that the states of the latent process reflect the effectiveness of the changes made to the software at each stage of testing. It is assumed that the latent process Y is a discrete time MC with some transition matrix $\boldsymbol{P}$ on a discrete state space E. Given the latent process T_i's are conditionally independent. A Bayesian analysis of the model was developed using a Gibbs sampler similar to the one proposed by Robert, Celeux and Diebolt (1993) for a given dimension of the state space E of the hidden chain. A new approach using marginal likelihoods was proposed to infer the unknown dimension of the hidden chain. The computation of

the marginal likelihoods was based on the method of Chib (1995). The attractive feature of Chib's method is its allowing for use of posterior samples generated by the Gibbs sampler in computing the marginal. Once marginal likelihoods are obtained for a given dimension, they can also be used to obtain Bayes factors; see Kass and Raftery (1995).

3.2 Modeling Number of Software Failures

As previously discussed, the Type II models describe the number of software failures during a specified time period. This class of models are basically point processes generated by the interfailure times T_i's. Typically, the point process models used for this purpose are Poisson processes and its extensions.

One of the earlier point process models is that of Goel and Okumoto (1979) which is known as the time-dependent error detection model. Most other Type II models are extensions of the Goel and Okumoto model. As pointed out by Singpurwalla and Wilson (1994), the model assumes that expected number of software failures to time t is given by the mean value function $\Lambda(t)$. The function $\Lambda(t)$, which is non-decreasing and bounded above, is given by

$$\Lambda(t) = a(1 - e^{-bt}) \tag{13}$$

In (13), a denotes the total number of faults (that is, bugs) in the software and b is considered to be the fault detection rate. The mean value function $\Lambda(t)$ defines the nonhomogeneous Poisson process (NHPP), $N(t)$ describing the number of software failures during $[0, t)$. Thus, we can write

$$Pr[N(t) = n] = \frac{[\Lambda(t)]^n}{n!} e^{-\Lambda(t)} \tag{14}$$

where the intensity function of the process is

$$\lambda(t) = \frac{d}{dt}\Lambda(t) = abe^{-bt}. \tag{15}$$

We note that, unlike the JM model, in the above the total number of errors is a random variable with mean a. Also, unlike the JM model, the above model implies that the times between successive software failures are dependent.

Langberg and Singpurwalla (1985) pointed out the relationship between the JM and the Goel-Okumoto model using a Bayesian perspective. More specifically, if ϕ and N are unkown in the failure rate of the JM model, that is, in $\phi(N - i + 1)$ then the JM model can be obtained by assuming degenerate priors on ϕ and N. Similarly, the Goel-Okumoto model is obtained by assuming a degenerate prior on ϕ and a Poisson prior on N with mean θ.

Statistical analysis of the model via maximum likelihood methods was discussed in Goel and Okumoto

(1979). Bayesian analysis of the model using MCMC methods was discussed in Kuo and Yang (1996). As pointed out by Singpurwalla and Wilson (1984), other NHPP models have been proposed based on different assumptions on the fault detection rate of the Goel-Okumoto model. Yamada (1991) gives an overview of such models.

Another NHPP model is the logarithmic Poisson execution time model of Musa and Okumoto (1984) which is based on the relationship between the intensity function and the mean value (cumulative intensity) function of a NHPP. The authors assume that the intensity function is given by

$$\lambda(t) = \lambda_0 e^{-\theta\Lambda(t)} \tag{16}$$

where $\lambda_0 > 0$ and $\theta > 0$. Note that $\lambda(t) = \frac{d}{dt}\Lambda(t)$ and using $\Lambda(0) = 0$, the differential equation (16) yields

$$\lambda(t) = \frac{\lambda_0}{\lambda_0\theta t + 1} \text{ and } \Lambda(t) = \frac{ln(\lambda_0\theta t + 1)}{\theta}. \tag{17}$$

A classical statistical analysis of the model is given in Musa and Okumoto (1984). Campodónico and Singpurwalla (1994) present a Bayesian analysis of the model using expert opinion and software failure data. A more general approach for incorporating expert opinion in the analysis of NHPPs was considered in Campodónico and Singpurwalla (1995).

A unification of the NHPP models used in software reliability is given in Kuo and Yang (1996). This was achieved by modeling software failure times using the general order statistics (GOS) and record value statistics (RVS) models. The authors showed that in the GOS models, the mean value function of the NHPP $N(t)$ can be written as

$$\Lambda(t) = \theta F(t) \tag{18}$$

where $F(t)$ is the cumulative distribution function for the failure times and θ is the mean of N, unknown number of bugs in the software. Under GOS model N is assumed to be a Poisson random variable. For example, if $F(t)$ is exponential, that is, $F(t) = 1 - e^{-bt}$ then (18) yields the Goel-Okumoto model. This was referred to as the exponential order statistics model by Miller (1986). Other GOS models such as Pareto, Weibull and extreme value statistics models can be obtained by specifying $F(t)$ accordingly in (18). Kuo and Yang (1996) discussed Bayesian inference for these NHPP models using Gibbs sampler sometimes together with data augmentation [see Tanner and Wong (1987)] and with Metropolis algorithms.

Alternatively, most of the Type I and Type II models above can be considered as special cases of self-exciting point processes which provide another unification framework for the software reliability models; see Chen and Singpurwalla (1997).

As pointed out by Singpurwalla and Wilson (1989), incorporation of covariate information into software

reliability modeling has not been very common. However, such information can be helpful in reliability assessments. More recently, Ray, Liu and Ravishanker (2006) considered use of software metrics in estimating intensity functions of NHPP models and developed Bayesian analysis of this class of models using MCMC methods.

Hidden Markov type models have also been considered in the context of point processes in software reliability. Ozekici and Soyer (2003) introduced the idea of operational process motivated by the concept of operational profile of Musa (1993, 1996). As pointed out by Ozekici and Soyer (2003), an operational profile simply consists of the set of all operations that a software system is designed to perform and their probabilities of occurrence. The authors introduce E representing the set of all operations and Y_t as operation performed by the software at time t. The continuous time process $Y = \{Y_t; t \geq 0\}$ is defined as operational process with state space E as the operational space. If the process Y is ergodic with some limiting distribution π, then the pair (E, π) defines the operational profile of the software. The operational process Y was assumed to be a Markov process in Ozekici and Soyer (2003).

The authors defined N_t as the number of faults remaining in the software at time t and the process $N = \{N_t; t \geq 0\}$ describing the evolution of number of faults over time. Defining the counting process $M = \{M_t; t \geq 0\}$ as the number of failures observed by time t and assuming perfect debugging we have $M_t = N_0 - N_t$ where N_0 is the initial number of faults in the software. The authors point out that the counting process M is modulated by the bivariate process $Z = (Y, N)$ where the intensity function of the M process is given by

$$\widehat{\lambda}(t) = N_t \lambda(Y_t). \tag{19}$$

The above is a generalization of Markov modulated Poisson processes (MMPPs) where the modulating process is the Y process implying $\widehat{\lambda}(t) = \lambda(Y_t)$ in (19). In other words, the MMPP model is the special case of the model where there is no debugging, that is, $N_t = N_0$ for all $t \geq 0$. A survey of MMPPs can be found in Fischer and Meier-Hellstern (1992). The MMPP is also a special case of the doubly stochastic Poisson process of Kingman (1964) where the intensity of the process is given by $\widehat{\lambda}(t) = \lambda(Y_t)$. A Markovian analysis of the model was presented in Ozekici and Soyer (2003) using the bivariate Markov process $Z = (Y, N)$ and a Bayesian analysis of the model was developed. Ozekici and Soyer (2006) discuss probabilistic and statistical issues related to these processes in a rather general setting and consider their extensions to include semiMarkov modulated Poisson processes.

More recently an MMPP model was considered by Landon, Özekici, Soyer (2009) to describe software failures and a Bayesian analysis of the model was developed using the exact Gibbs sampler given in Fearnhead and Sherlock (2006). Inference about the dimension of the Markov process was made by adopting the marginal likelihood method of Chib (1995). A discrete time latent Markov process was considered in

Ravishanker, Liu and Ray (2008) to describe the evolution of the parameters of NHPP.

4 Nonparametric Software Reliability Models

Our review of software reliability models in the previous sections focused on parametric models. It is often desirable to relax some of the parametric assumptions to achieve robustness and therefore to consider semiparametric or nonparametric models. Nonparametric models have found limited use in the software reliability literature.

Earlier attempts in nonparametric software reliability modeling include Miller and Sofer (1986) who used monotone regression estimates of failure rates and Miller and Sofer (1991) developed long-range reliability predictions based on the monotonic regression model. Barghout, Abdel-Ghaly and Littlewood (1998) present a nonparametric treatment of general order statistics models such as that of Miller (1986). Their proposed approach is based on kernel density estimation methods.

One of the first nonparametric Bayesian approaches to software reliability modeling was introduced in El-Aroui and Soler (1996) where the authors assumed exponentially distributed conditionally independent times between failures. In other words, given the failure rates λ_i's , they assume $T_i|\lambda_i \sim Exp(\lambda_i)$ for $i \geq 1$ where T_i's are independent. It was assumed that the failure rates $\{\lambda_i\}$ follow a Markovian evolution specified by prior distributions $p(\lambda_i|\lambda_{i-1})$ for all $i \geq 1$. The authors referred to their setup as the Bayes exponential Markov assupmtion. The form of the priors are specified using a maximum entropy (ME) based method; see Ebrahimi, Soofi and Soyer (2010) for the ME method. Bayesian analysis of the corresponding model was developed using MCMC methods.

Wilson and Samaniego (2007) introduced a nonparametric Bayesian approach for analysis of the order statistic models of discussed in the above. In so doing, the authors modeled $F(t)$, the cumulative distribution function for the failure times nonparametrically and assumed a uniform prior for N, initial number of faults in the software. A Dirichlet process (DP) prior [see Ferguson (1973)] was used for F and MCMC methods were implemented for posterior analysis.

Nonparametric versions of NHPP models were considered in Kuo and Ghosh (2001) who developed Bayesian analysis of such models. Their approach involved use of gamma process [Singpurwalla (1997)] and beta process [Hjort (1990)] priors for cumulative intensity functions and extended gamma process [Dykstra and Laud (1981)] priors for intensity functions of the NHPPs. Posterior analysis of the models were developed using MCMC methods.

More recently, Soyer (2010) proposed a nonparametric Bayesian model for times between software failures. The model assumes exponential times between failures, but relaxes the parametric assumptions used for the priors in Type I models. Thus, it can be considered as a semiparametric model.

4.1 A Semiparametric Model for Times between Successive Failures

A generalization of the LV model and the HB model of Mazzuchi and Soyer (1988), can be obtained by relaxing the parametric assumptions on the prior distributions.

A common view in software reliability modeling is that, unlike hardware, software does not exhibit any aging [Morali and Soyer (2003)] and this lack of aging motivates the exponential model (2) for the times between successive failures. Thus, we assume that $T_i|\lambda_i \sim Exp(\lambda_i)$ and T_i's are conditionally independent. The LV and HB models, that we have previously discussed, are both based on this assumption but they differ in prior models describing the evolution of λ_i's over testing stages. Also, both models have parametric prior distributions for λ_i's.

A fully parametric prior model may not be adequate in capturing the changes in the failure rates. As pointed out by Gelfand (1999), in cases where parametric assumptions are too restrictive, a semi-parametric model can be used by a non-parametric specification of the some components of the model. In view of the above, Soyer (2010), presented a semi-parametric model by treating the prior distribution for λ_i's nonparametrically and in so doing used a Dirichlet process (DP) prior [see Ferguson (1973)] to model the unknown distributions.

Consider the HB model setup of Mazzuchi and Soyer (1988) and treat the distribution of λ_i, say G, as unknown. Uncertainty about G can be explained by a DP prior. We denote this as

$$G|G_0, M \sim DP(G_0, M), \tag{20}$$

where G_0 is the baseline prior representing the best guess about G and M is the strentgh of belief in the best guess. In Soyer (2010) the baseline prior G_0 is assumed to be a gamma density with parameters α and β, that is, $G_0 = Gam(\alpha,\, \beta)$. If α and β are assumed to be unknown, then we can specify a joint prior, say, $p(\alpha,\, \beta)$. Thus, the semi-parametric setup of Soyer (2010) can be summarized as follows:

$$\begin{cases} [T_i|\lambda_i] \sim Exp(\lambda_i) \\ [\lambda_i|G] \sim G \\ [G|G_0, M] \sim DP(G_0, M) \\ [G_0|\alpha,\, \beta] = Gam(\alpha,\, \beta) \\ [\alpha,\, \beta] \ \sim p(\alpha,\, \beta) \end{cases} \tag{21}$$

In the above setup, the mixing distribution G is assumed to be unknown and the prior distribution assumed for the mixing distribution is a Dirichlet process. Therefore, the above class of models are termed as "Dirichlet process mixed models"; see Mukhopadhyay and Gelfand (1997).

We note that if the form of G is specified in (21), then we are back to the the hierarchical Bayes setup

of Mazzuchi and Soyer (1988). Thus, the hierarchical model is obtained as $M \to \infty$ in (21). It is possible to put a prior on M and to learn about it from data. For example, following Escobar and West (1995), one can specify a gamma prior for M. As discussed in Escobar and West (1995), by learning about M one can learn about the distinct number of λ_i's. More specifically, Escobar and West defined K to be the number of unique values of λ_i's, also referred to as the number of cliques by MacEachern (1998). Using results of Antoniak (1974), it can be shown that, conditioned on K, M is independent of all other parameters. This result can be used in developing a Gibbs sampler for analysis of the model.

Bayesian analysis of the semi-parametric model (21) requires additional steps in the Gibbs sampler when we draw samples from the λ_i's. An algorithm has been proposed by Escobar and West (1995) that enables us to draw samples from the full conditional distribution of λ_i's without sampling from the distribution of G. If the belief parameter M is treated as unknown and M has a gamma prior, then it can be shown that full conditional posterior distribution of M can be obtained as a mixture of two gamma densities using a data augmentation step at each Gibbs iteration. This allows one to obtain the distribution of K. The details are given in Escobar and West (1995).

Soyer (2010) used the semiparametric model (21) by assuming a gamma prior on M and implemented the Escobar and West algorithm to analyze the System 40 data of Musa (1979). The data which has been previously analyzed by Singpurwalla and Soyer (1992) consists of 101 succesive times between software failures. A plot of the data is shown in Figure 1 where times between failures tend to increase over time suggesting an overall improvement in reliability.

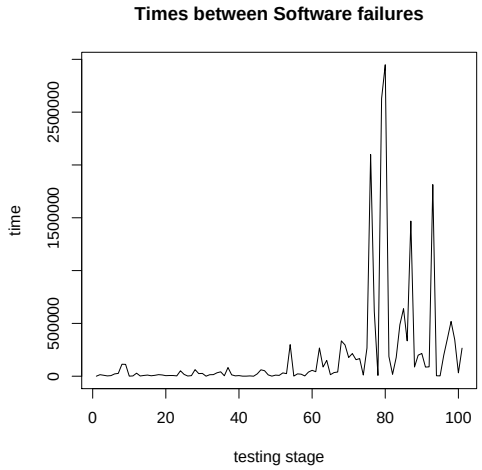

Figure 1: Musa's System 40 Data

Figure 2 presents posterior density plots for distributions of failure rates $\lambda_{15}, \lambda_{45}, \lambda_{85}$ and λ_{101}. We note from Figure 2 that the posterior distributions of the failure rates has mixture components as implied by

(21). Furthermore, we see from the figure that the failure rates have different supports at each stage and the values of failure rates in general seem to be decreasing by time, that is, by testing stages. The same conclusion was reached by Singpurwalla and Soyer (1992) who used dynamic Bayesian time series models to analyze the same data. A similar conclusion can be reached from Figure 3 which shows the box plots of posterior distributions of λ_i's over different stages of testing. Again we note that the λ_i's differ from one stage to another and on average λ_i's tend to go down over time suggesting potential reliability growth.

The use of the Dirichlet process prior for G implies discreteness and therefore there is a positive probability that some of the λ_i's will take the same values. This formation of groups of λ_i's allows grouping of similar testing stages. This similarity may be due to the similar design changes made to the software at those stages and/or due to the similar faults encountered during the debugging process. Posterior inference about number of heterogeneous groups can be made using the posterior distribution of K given in Figure 4. The figure suggests that 9-10 different groups of failure rates seem to be the most likely one.

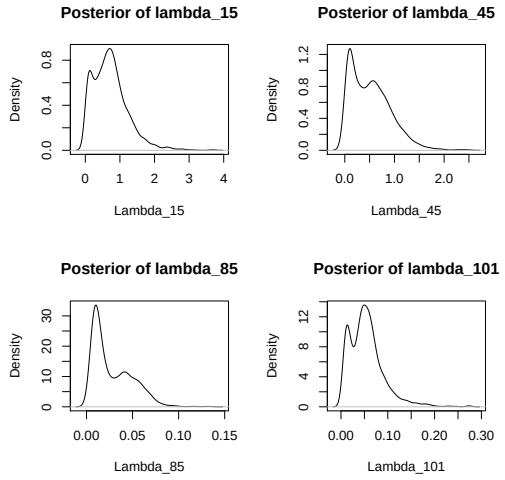

Figure 2: Posterior distributions of $\lambda_{15}, \lambda_{45}, \lambda_{85}$ and λ_{101}.

5 Decision Models in Software Reliability

As noted by Singpurwalla and Wilson (1999), in addition to inference and prediction, software reliability models can also be used for decision making. An important component of software reliability modeling is development of optimal testing strategies. As Singpurwalla and Wilson (1999) point out, such strategies involve determining "how long should the software be tested" as well as deciding "which test cases are to be used." Our discussion in this section will focus on the former, that is, on deciding when to stop testing the software and release it. In so doing, we will review the literature and discuss the decision theoretic

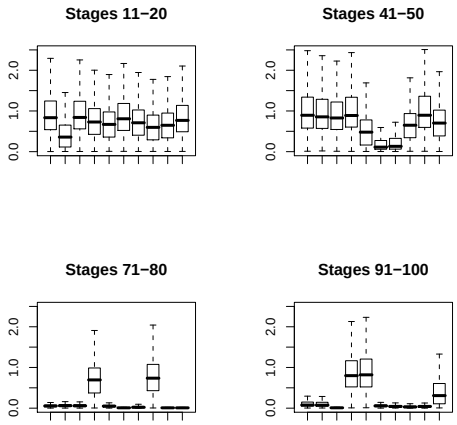

Figure 3: Posterior distributions of λ_i's over different stages testing

formulation of this class of problems.

Earlier works on determining when to stop the testing process and release the software include Forman and Singpurwalla (1977, 1979), Okumoto and Goel (1980), Yamada, Narihisa and Osaki (1984) and Ross (1985). None of these earlier approaches are based on a formal decision-theoretic approach. The decision theoretic approach involves trade-offs such as cost of testing versus cost of releasing an unreliable piece of software.

Dalal and Mallows (1986) was among the first who considered a decision theoretic approach to the problem. Singpurwalla (1989, 1991) also tackled the problem using a Bayesian decision theoretic setup. The work of Dalal and Mallows provides an exact but a complicated solution and as a result, an asymptotic solution is given by the authors. The work of Singpurwalla addresses a two-stage problem where the solution is complicated due to computational difficulties involved in preposterior analysis. More recent work by Ozekici and Catkan (1993) and McDaid and Wilson (2001) also propose decision theoretic solutions. Ozekici and Catkan (1993) present a very general setup and provide characterizations of the optimal release policy. Their approach does not include any revision of uncertainty in the Bayesian sense. McDaid and Wilson (2001) considered the case of single stage testing using nonhomogeneous Poisson process type models and developed a Bayesian solution. The authors also considered a sequential testing problem but the solution was not analytically tractable. Boland and Singh (2003) looked at the optimal release time problem using the geometric Poisson model and developed solutions for the case of known parameters as well as using a Bayesian framework.

Morali and Soyer (2003) addressed the optimal stopping problem during the software development phase. They formulated the problem as a sequential decision problem and used a Bayesian approach. Ozekici and

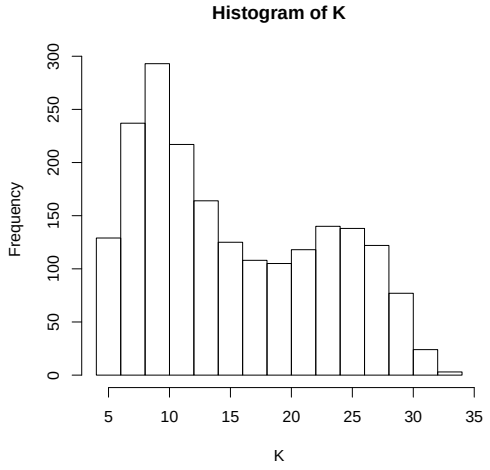

Figure 4: Posterior distribution of K

Soyer (2001) developed optimal testing strategies for software with an operational profile as discussed in Musa (1993, 1996). The authors assumed that the software was tested sequentially for given durations of time in all of the operations. In so doing, they considered a testing scenario where each test case was designed to test a given operation in the profile and obtained optimal testing time for each operational profile.

The Bayesian decision theoretic approach to the optimal release time problem requires specification of three components:

(i) a utility (loss) function that reflects the consequences of selecting a specific release (or stopping) time;

(ii) a probability model;

(iii) a prior distribution reflecting software engineer's a priori beliefs about all unknown quantities.

These three components are common to all the decision theoretic approaches discussed in the above. In what follows, we highlight the setup by Morali and Soyer (2003) who considered the problem of how to decide when to terminate the testing/modification process and to release the software.

5.1 A Sequential Decision Model

Consider the testing protocol where testing is done sequentially until failure (or up to a prespecified testing time) and at the end of each test stage corrections and modifications are made to the software with the hope of increasing its reliability. Let $T_i, i = 1, 2, \ldots$, denote the life-length of the software during the i-th stage of testing after the $(i-1)$-st modification made to it. Assume that T_i follows the exponential model in (2) with failure rate λ_i which changes from one stage to another as a result of the modifications made to it after each stage.

At the end of each stage, following modifications made to the software, a decision must be made whether to terminate the debugging process. Thus after completion of i stages of testing, the decision of whether or not to stop testing will be based on $T^{(i)} = (T_i, T^{(i-1)})$, where $T^{(0)}$ represents information prior to any testing. Morali and Soyer (2003) assumed that the evolution of λ_i's was described by the Markovian model (8). The authors considered the loss function associated with stopping and releasing the software after the i-th testing stage as

$$\mathcal{L}_i(T^{(i)}, \lambda_{i+1}) = \sum_{j=1}^{i} \mathcal{L}_{\mathrm{T}}(T_j) \; + \; \mathcal{L}_{\mathrm{S}}(\lambda_{i+1}), \tag{22}$$

where $\mathcal{L}_{\mathrm{T}}(\cdot)$ represents the loss due to testing for one stage, and $\mathcal{L}_{\mathrm{S}}(\cdot)$ relates to the loss associated with stopping and releasing the software. Note that $\mathcal{L}_0$ (the loss associated releasing the software before any testing) is a function of λ_1. We note that in (22) $\mathcal{L}_{\mathrm{T}}(\cdot)$ captures the cost of testing which is a function of T_j and $\mathcal{L}_{\mathrm{S}}(\cdot)$, the loss associated with stopping and releasing the software which is a function of the failure rate λ_{i+1}. This component reflects costs associated with bugs discovered and fixed after testing, that is, it deals with failures during the usage phase. Since λ_{i+1} will be proportional to the number of bugs present in software after release, $\mathcal{L}_{\mathrm{S}}(\cdot)$ will be assumed to be an increasing function of λ_{i+1}.

Morali and Soyer (2003) represented the stopping problem as a sequential decsion problem as a m-stage decision tree given in Figure 5. The solution of the decision problem involves using dynamic programming. As noted by the authors, solution of the tree proceeds in the usual way by taking expectation at random nodes and minimizing the expected loss at the decision nodes. At decision node i, the additional expected loss associated with the STOP and the TEST decisions are given by the terms $\mathrm{E}[\mathcal{L}_{\mathrm{S}}(\lambda_{i+1})\big|T^{(i)}]$ and $\mathrm{E}[\mathcal{L}_{\mathrm{T}}(T_{i+1})\big|T^{(\mathrm{i})}] + \mathrm{L}^*_{i+1}$, respectively where

$$\mathrm{L}^*_i = \; \mathrm{MIN}\Big\{\mathrm{E}[\mathcal{L}_{\mathrm{S}}(\lambda_{i+1}) \mid T^{(i)}] \; , \mathrm{E}[\mathcal{L}_{\mathrm{T}}(T_{i+1}) \mid T^{(i)}] + \mathrm{L}^*_{i+1} \; \Big\} \tag{23}$$

for $i = 0, 1, \ldots$, and the optimal decision at decision node i then is the one associated with L^*_i.

In Figure 5, m, the maximum number of testing stages, can be considered infinite with $\mathrm{L}^*_{m+1} = \infty$. We note that even for the case of finite m the calculation of L^*_i in (23) is not trivial as it involves implicit computation of expectations and minimizations at each stage. Morali and Soyer (2003) investigated the possibility of developing one-stage ahead optimal stopping rules by using results from van Dorp, Mazzuchi and Soyer (1997) and illustrated implementation of their approach using simulated as well as actual software failure data.

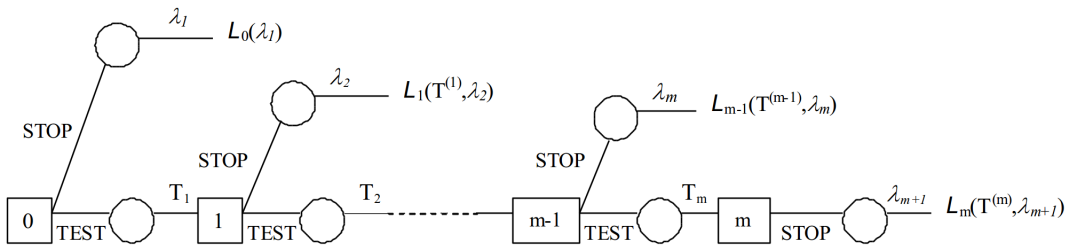

Figure 5: The m-stage decision tree for the optimal release problem

6 Concluding Remarks

In this article we have presented an overview of software reliability. Our overview has focused on two issues, namely, software reliability modeling and software testing strategies including optimal stopping decisions. We have presented basic concepts and definitions in software reliability and discussed unique aspects of software failure mechanism. Our coverage of software reliability modeling has included both traditional models as well as more recent developments such as hidden Markov models and nonparametric modeling strategies. We have discussed basic issues in development of software testing strategies and have reviewed earlier approaches most of which were not based on decision theoretic approaches. More recent Bayesian decision theoretic approaches have been emphasized in our overview and computational difficulties associated with them have been discussed.

References

Antoniak, C. E. (1974). Mixtures of Dirichlet processes with applications to Bayesian non-parametric problems. *Ann. Statist.*, **2**, 1152-1174.

Barghout, M., Abdel-Ghaly, A. A. and Littlewood, B. (1998). A non-parametric order statistics software reliability model. *Software Testing, Verification and Reliability*,**8**, 113-132.

Barlow, R. E. and Singpurwalla, N. D. (1985). Assessing the reliability of computer software and computer networks: An opportunity for partnership with computer scientists. *American Statistician*, **39**, 88-94.

Basu, S. and Ebrahimi, N. (2003). Bayesian software reliability models based on martingale processes. *Technometrics*, **45**, 150-158.

Bather, J.A. (1965). Invariant conditional distributions. *Ann. Math. Statist.*, **36**, 829-46.

Boland, P. J. and Singh, H. (2002). Determining the optimal release time for software in the geometric Poisson reliability model. *Intl. Jour. of Reliability, Quality and Safety Engineering*, **9**, 201-213.

Campodónico, S. and Singpurwalla, N.D. (1994). A Bayesian analysis of the logarithmic-Poisson execu-

tion time model based on expert opinion and failure data. *IEEE Trans. Soft. Eng.*, **SE-20**, 677-683.

Campodonico, S. and Singpurwalla, N. D. (1995). Inference and predictions from Poisson point processes incorporating expert knowledge. *J. of the Am. Statist. Assoc.*, **90**, 220-226.

Chen, Y. and Singpurwalla, N.D. (1994). A non-Gaussian Kalman filter model for tracking software reliability. *Statistica Sinica*, **4**, 535-548.

Chen, Y. and Singpurwalla, N.D. (1997). Unification of software reliability models via self-exciting point processes. *Advances in Applied Probability*, **29**, 337-352.

Chib, S. (1995). Marginal likelihood from the Gibbs output. *J. of the Am. Statist. Assoc.*, **90**, 1313-1321.

Chib, S., and Greenberg, E. (1995). Understanding the Metropolis-Hastings algorithm. *American Statistician*, **49**, 327-335.

Dalal, S.R. and Mallows, C.L. (1986). When should one stop testing software? *J. Amer. Statist. Assoc.*, **83**, 872-79.

Deely, J. J. and Lindley, D. V. (1981). Bayes empirical Bayes. *J. Amer. Statist. Assoc.*,**76**, 833-841.

van Dorp, J. R., Mazzuchi, T. A. and Soyer, R. (1997). Sequential inference and decision making during product development. *Journal of Statistical Planning and Inference*, **62**, 207-218.

Durand, J. B. and Gaudoin, O. (2005). Software reliability modelling and prediction with hidden Markov chains. *Statistical Modelling*, **5**, 75-93.

Dykstra, R. L. and Laud, P. W. (1981). A Bayesian nonparametric approach to reliability. *Ann. Statist.*, **9**, 356-367.

Ebrahimi, N., Soofi, E. S. and Soyer, R. (2010). Information measures in perspective. *Int. Statist. Rev.*, **78**, 383-412.

El-Aroui, M. A. and Soler, J. L. (1996). A Bayes nonparametric framework for software reliability analysis. *IEEE Trans. Reliability*, **R-45**, 652-660.

Erkanli, A., Mazzuchi, T.A., and Soyer, R. (1998). Bayesian computation for a class of reliability growth models. *Technometrics*, **40**, 14-23.

Escobar, M. D. and West, M. (1995). Bayesian density estimation and inference using mixtures. *J. Amer. Statist. Assoc.*, **90**, 577-588.

Fearnhead, P. and Sherlock, C. (2006). An exact Gibbs sampler for the Markov-modulated Poisson process. *Journal of the Royal Statistical Society, Series B*, **68**, 767-784.

Ferguson, T. S. (1973). A Bayesian analysis of some non-parametric problems. *Annals of Statistics*, **1**, 705-711.

Fischer, W. and Meier-Hellstern, K. (1992). The Markov-modulated Poisson process cookbook. *Performance Evaluation*,**18**, 149-171.

Forman, E. H. and Singpurwalla, N.D. (1977). An empirical stopping rule for debugging and testing

computer software. *J. Amer. Statist. Assoc.*, **72**, 750-57.

Forman, E. H. and Singpurwalla, N. D. (1979). Optimal time intervals for testing hypotheses on computer software errors. *IEEE Transactions in Reliability*, **R-28**, 250-253.

Gelfand, A. E. and Smith, A. F. M. (1990). Sampling-based approaches to calculating marginal densities. *J. Amer. Statist. Assoc.*, **85**, 398-409.

Gelfand, A. E. (1999). Approaches for Bayesian semi-parametric regression. in *Asymptotics, Nonparametrics and Time Series*, (Ed. S. Ghosh), 615-638. New York:Marcel Dekker.

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

Goel, A. L. (1985). Software reliability models: Assumptions, limitations, and applicability. *IEEE Trans. Soft. Eng.*, **SE-11**, 1411-23.

Hjort, N. L. (1990). Non-parametric Bayes estimates based on Beta processes in models for life history data. *Ann. Statist.*, **18**, 1259-1294.

Hudson, A. (1967). Program errors as a birth and death process. *Technical Report SP-3011.* Santa Monica, CA: Systems Development Corporation.

Iannino, A., Musa, J.D., Okumoto, K. and Littlewood, B. (1984). Criteria for software reliability model comparison. *IEEE Trans. Soft. Eng.*, **SE-10**, 687-91.

Jelinski, Z. and Moranda, P. (1972). Software reliability research. in *Statistical Computer Performance Evaluation*, Ed. W. Freiberger, 465-84. New York: Academic.

Joe, H. and Reid, N. (1985). On the software reliability models of Jelinski-Moranda and Littlewood. *IEEE Transactions in Reliability*, **R-34**, 216-218.

Kass, R. E. and Raftery, A. E. (1995). Bayes factors. *J. Amer. Statist. Assoc.*, **90**, 773-795.

Kingman, J. F. C. (1964). On doubly stochastic Poisson processes. *Proceedings of the Cambridge Philosophical Society*, **60**, 923–960.

Koch, G. and Spreij, P. (1983). Software reliability as an application of martingale and filtering theory. *IEEE Transactions in Reliability*, **R-32**, 342-345.

Kuo, L. and Yang, T. Y. (1995). Bayesian computation of software reliability. *Journal of Computational and Graphical Statistics*, **4**, 65-82.

Kuo, L. and Yang, T. Y. (1996). Bayesian computation for nonhomogeneous Poisson processes in software reliability. *J. Amer. Statist. Assoc.*, **91**, 763-773.

Kuo, L. and Ghosh, S. K. (2001). Bayesian nonparametric inference for nonhomogeneous Poisson Processes. *Technical Report*, North Carolina State University. Institute of Statistics, Mimeo Series No. 2530.

Landon, J. , Özekici, S. and Soyer, R. (2009). A Markov modulated Poisson model for software reliability. *Technical Report TR-2009-1*, I^2SDS, The George Washington Unuversity.

Langberg, N. and Singpurwalla, N.D. (1985). A unification of some software reliability models. *SIAM J. Sci. Statist. Comput.*, **6**, 781-90.

Lindley, D. V. (1980). Approximate Bayesian methods. *Trabajos Estadistica*, **31**, 223-237.

Littlewood, B. and Verall, J. L. (1973). A Bayesian reliability growth model for computer software. *Appl. Statist.*, **22**, 332-46.

MacEachern, S. N. (1998). Computational methods for mixture of Dirichlet process models, in *Practical Nonparametric and Semiparametric Bayesian Statistics*, (Eds. D. Dey, P. Muller, D. Sinha), 23-44, Springer-Verlag.

Mazzuchi, T.A. and Soyer, R. (1988). A Bayes empirical Bayes model for software reliability. Practical Nonparametric and Semiparametric Bayesian Statistics, **R-37**, 248-54.

McDaid, K. and Wilson, S. P. (2001). Deciding how long to test software. *The Statistician*, **50**, 117-134.

Meinhold, R. J. and Singpurwalla, N. D. (1983a). Bayesian analysis of a commonly used model for describing software failures. *Statistician*, **32**, 168-173.

Meinhold, R. J. and Singpurwalla, N. D. (1983b). Understanding the Kalman filter. *American Statistician*, **37**, 123-127.

Miller, D. R. (1986). Exponential order statistic models of software reliability growth. *IEEE Trans. Soft. Eng.*, **SE-12**, 12-24.

Miller, D. R. and Sofer, A. (1986). A nonparametric approach to software reliability, using complete monotonicity. in *Software Reliability: A State of the Art Report*, (A. Bendell, P. Mellor, eds), 183-195; Pergammon Press.

Miller, D. R. and Sofer, A. (1991). A nonparametric software reliability growth model. *IEEE Trans. Reliability*, **R-40**, 329-337.

Miller, J. E. and Smith, R. L. (1986). A non-Gaussian state space model and application to prediction of records. *Journal of the Royal Statistical Society, Series B*, **48**, 79-88.

Morali, N. and Soyer, R. (2003). Optimal stopping in software testing. *Naval Research Logistics*, **50**, 88-104.

Moranda, P.B. (1975). Prediction of software reliability and its applications. *Proceedings of the Annual Reliability and Maintainability Symposium*, pp. 327-32.

Mukhopadhyay, S. and Gelfand, A. E. (1997). Dirichlet process mixed generalized linear models. *J. Amer. Statist. Assoc.*, **92**, 633-639.

Musa, J. D. (1979). Software reliability data. *IEEE Computing Society Repository.*

Musa, J. D. (1993). Operational profiles in software reliability engineering. *IEEE Software*, **10**, 14-32.

Musa, J. D. (1996). The operational profile. in *Reliability and Maintenance of Complex Systems*, 332-343, Springer Verlag.

Musa, J.D. and Okumoto, K. (1982). Software reliability models: concepts, classification, comparison and practice, in *Electronic Systems Effectiveness and Life Cycle Costing*, Ed. J. K. Skwirzynski, 395-423, Springer-Verlag.

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

Musa, J. D. and Okumoto, K. (1984). A logarithmic Poisson execution time model for software reliability measurement. *Proceedings of the 7th International Conference on Software Engineering*, Orlando, 230-237.

Okumoto, K. and Goel, A. L. (1980). Optimum release time for software systems, based on Reliability and Cost Criteria. *Journal of Systems Software*, **1**, 315-318.

Ozekici, S. and Catkan, N. A. (1993). A dynamic software release model. *Computational Economics*, **6**, 77-94.

Ozekici, S. and Soyer, R. (2001). Bayesian testing strategies for software with an operational profile. *Naval Research Logistics*, **48**, 747-763.

Ozekici, S. and Soyer, R. (2003). Reliability of software with an operational Profile. *European Journal of Operational Research*, **149**, 459-474.

Ozekici, S. and Soyer, R. (2006). Semi-Markov modulated Poisson process: Probabilistic and statistical analysis. *Mathematical Methods of Operations Research*, **64**, 125-144.

Pievatolo, A., Ruggeri, F. and Soyer, R. (2010). A Bayesian hidden Markov model for imperfect debugging, under review.

Ravishanker, N., Liu, Z. and Ray, B. K. (2008). NHPP models with Markov switching for software reliability. *Computational Statistics and Data Analysis*, **52**, 3988-3999.

Ray, B., Liu, Z. and Ravishankar, N. (2006). Dynamic reliability models for software using time dependent covariates. *Technometrics*, **48**, 1-10.

Robert, C. P., Celeux, G. and Diebolt, J. (1993). Bayesian estimation of hidden Markov chains: A stochastic implementation. *Statistics and Probability Letters*, **16**, 77-83.

Ross, S. M. (1985). Software reliability: The stopping rule problem. *IEEE Trans. Soft. Eng.*, **SE-11**, 1472-1476.

Ruggeri, F. and Soyer, R. (2008). Advances in Bayesian software reliability modeling. in *Advances in Mathematical Modeling for Reliability*, (Eds. T. Bedford, J. Quigley, L. Walls, B. Alkali, A. Daneshkhah and G. Hardman), 165-176, Amsterdam:IOS Press.

Samaniego, F.J. and Wilson, S. P. (2007). Nonparametric analysis of the order-statistic model in software reliability. *IEEE Trans. Soft. Eng.*, **SE-33**, 198-208.

Schick, G. J. and Wolverton, R.W. (1978). Assessment of software reliability. *Proc. Oper. Res.*, 395-422, Wirzberg-Wien: Physica-Verlag.

Singpurwalla, N. D. (1991). Determining an optimal time interval for testing and debugging software. *IEEE Trans. Soft. Eng.*, **SE-17**, 313-319.

Singpurwalla, N. D. (1989). Preposterior analysis in software testing. in *Statistical Data Analysis and Inference*. (Y. Dodge, Editor), 581-595, New York:Elsevier.

Singpurwalla, N. D. (1997). Gamma processes and their generalizations: an overview. in *Engineering Probabilistic Design and Maintenance for Flood Protection*, Ed. R. Cooke et al, 67-75, New York:Kluwer.

Singpurwalla, N.D. and Soyer, R. (1985). Assessing (software) reliability growth using a random coefficient autoregressive process and its ramifications. *IEEE Trans. Soft. Eng.*, **SE-11**, 1456-64.

Singpurwalla, N.D. and Soyer, R. (1992). Non-homogeneous autoregressive processes for tracking (software) reliability growth, and their Bayesian analysis. *J. Roy. Statist. Soc. B*, **54**, 145-56.

Singpurwalla, N. D. and Soyer, R. (1996). Assessing the reliability of software: An overview. in *Reliability and Maintenance of Complex Systems*, 389-408, Springer Verlag.

Singpurwalla, N.D. and Wilson, S. P. (1994). Software reliability modeling. *International Statistical Review*, **62**, 289-317.

Singpurwalla, N.D. and Wilson, S. P. (1999). *Statistical Methods in Software Engineering*, New York: Springer Verlag.

Soyer, R. (2010). A nonparametric Bayesian model for software failures. *Technical Report TR-2010-16*, I^2SDS, The George Washington Unuversity.

Soyer, R. (2011). Markov and hidden Markov models. in *Encyclopedia of ORMS*, (J. J. Cochran, Ed.), Wiley, forthcoming.

Tanner, T. A. and Wong, W. H. (1987). The calculation of posterior distributions by data augmentation. *J. Am. Statist. Ass.*, **82**, 528-549.

Tierney, L. and Kadane, J. B. (1986). Accurate approximations for posterior moments and marginal densities. *J. Amer. Statist. Assoc.*, **81**, 82-86.

West, M. and Harrison, J. (1997). *Bayesian Forecasting and Dynamic Models*. 2nd Edition, Springer-Verlag.

Wiper, M. P. (2007). Software reliability: Bayesian analysis, in *The Encyclopedia of Statistics in Quality and Reliability*, Eds. F. Ruggeri, R.S. Kenett and F.W. Faltin, **4**, 1859-1863, Chichester: John Wiley and Sons.

Yamada, S. (1991). Software quality/reliability measurement and assessment: Software reliability growth models and data analysis. *J. Inform. Process.*, **14**, 254-66.

Yamada, S., Narihisa, H. and Osaki, S. (1984). Optimum release policies for a software system with a scheduled software delivery time. *Int. J. Systems Science*, **15**, 905-914.