Software Reliability for Multi-Type Defect: Applications to Modern Bug Databases

Technical Report No.ASD/2012/9
Dated: 7 September, 2012

Vignesh T Subrahmaniam
Software Sciences and Analytics
GE Global Research, Bangalore, 560066, India
vignesh.ts@ge.com

Anup Dewanji
Applied Statistics Unit
Indian Statistical Institute,
Kolkata 700108
dewanjia@isical.ac.in

And

Bimal K. Roy
Applied Statistics Unit,
Indian Statistical Institute,
Kolkata 700108
bimal@isical.ac.in

Software Reliability for Multi-Type Defects: Applications to Modern Bug Databases

Vignesh T Subrahmaniam
Software Sciences and Analytics
GE Global Research, Bangalore, 560066, India
vignesh.ts@ge.com

Anup Dewanji
Applied Statistics Unit
Indian Statistical Institute, Kolkata, 700108, India
dewanjia@isical.ac.in
and
Bimal K Roy
Applied Statistics Unit
Indian Statistical Institute, Kolkata, 700108, India
bimal@isical.ac.in

Abstract

Software bug databases provide an important source of data for assessing the reliability of a software. Statistical analysis of these databases has been a challenge due to lack of information on usage and reporting rates of the defects. This paper proposes a novel semi-parametric analysis that makes use of defect classifications into multiple types to enable estimation of a model without making strict assumptions about the underlying usage rate of the software. The proposed model can account for differences in reporting rates of different classes of defects and the self-exciting nature of the user driven defect discovery process. Estimation of the proposed model can be done using generalized linear model procedures. New reliability metrics whose computation does not depend on the unknown usage rate of the software have been proposed and methods for estimating them have been developed. The proposed model has been applied to data retrieved from the bug database of a popular scripting language, named Python.

Keywords: Beta-testing, Multivariate Counting Process, Logarithmic Poisson Model, Partial Likelihood, Reliability Metrics.

technometrics tex template (do not remove)

1 Introduction

Bug databases containing user-reported defects have become ubiquitous for both commercial and open source software and they are viewed as an important data source for software reliability assessments. In fact after the release of a software, bug databases can be the only source of data for assessing the reliability of a software in the field. Traditionally, software reliability models have focused on estimating the reliability of the software based upon data generated from in-house testing, where the testing methodology and reporting of defects is strictly controlled. These reliability models cannot be directly applied to data retrieved from bug databases as they contain defects recorded due to uncontrolled software usage and reporting of software defects. The usage rate of a software is typically a function of time and is in general unknown. The reporting pattern also depends on the type of users. The distribution of the severity/type of defects reported by users may be quite different from those detected by a controlled testing procedure. There is a tendency on the part of an average user to discover behaviour defects more often as compared to, say, critical security defects. Such tendencies can complicate the development of a software reliability model.

In this article, we use the classification of user-reported software defects into multiple types (based, for example, on their severity) to formulate and estimate a software reliability model that is non-parametric with respect to the usage rate and takes into account differences in reporting rates of different types of defects. A partial likelihood approach is used for model estimation. Based upon the proposed model, reliability metrics are proposed that do not depend upon the usage rate. An additional advantage of the proposed model is that it can be estimated using generalized linear model procedures found in most statistics packages. Section 2 formulates the model and describes the method of partial likelihood for estimating the parameters of interest along with estimation of the reliability metrics. Section 3 considers a simulation study to investigate the finite sample properties of the estimators and Section 4 considers an analysis of a publicly available bug database to illustrate our methodology. Section 5 ends with some concluding remarks.

1.1 Background

Controlled testing of a software by a team of software engineers is expensive and is limited by the number of test runs of the software. To increase the number of test runs, it is common for the software to be released (before it's final release) to a select user community as a "beta" version for a limited period of time for usage and testing. Such testing is called beta-testing (see http://searchcio-midmarket.techtarget.com/definition/beta-test for an explanation). Beta-testing is an important example of user-driven defect discovery and reporting. The benefit of recording software defects observed by users of a software on a continuing basis has been recognized by both commercial and open source software communities. Their efforts have led to the creation of specialized databases called bug databases, where any user of the software can report a defect using the internet.

Software defects can be classified into multiple types and certain defect types are of great importance to the software community, an example being defects which are related to security loopholes in the software (Musa 2005 pp. 198). Software reliability models can be expected to answer questions such as "Whether newer versions of a software are more reliable with respect to security faults when compared to older versions?" If the answer to such a question is negative, a root cause analysis (RCA) could be initiated to explain the lower reliability (See, for example, Yu 1998, Leszak et al. 2000). Inference such as this, when made using user-reported bug databases, needs to be done carefully as multiple factors can confound the inference of a statistically significant result. Usage rates can confound statistical inference because a version of software that is used more often is bound to have more defects reported as compared to a version that is used less often. Usage rates not only vary across versions of software, but are functions of time and could be stochastic in nature. A more subtle confounding effect is due to the differences in the discovery and reporting rates of different types of defects. For example, it is common to discover a large number of behaviour related defects, which are defects caused by the software not behaving in a manner expected by the user, as opposed to "crash" related defects such as "a segmentation fault while sequentially importing a database and TCP/IP module". This is because an average user of the software is more likely to test it's behavioural aspects, while to test and discover crash related defects requires specialized software knowledge, which may not be that common in the user community. On the other hand, crash related defects are more likely to be reported by a specialist user than a behaviour related defect by an average user. Hence a software reliability model applicable to user reported defects needs to accommodate unknown software usage rate and differences in reporting of different types of defects.

1.2 Related Work

In the past, substantial work has been done to assess the reliability of a software using data from controlled software testing procedures (See Jelinski and Moranda 1972, Moranda 1975, Musa et al. 1987, Goel and Okomoto 1978, Yamada et al. 1985, Yang and Chao 1995 among others). When data from software testing is obtained in a controlled environment, the modeling assumptions made by these authors are justified. In the past, reliability models incorporating time-varying reporting rates of defects either assumed a NHPP model with a parametric form for the defect reporting rate (Schlick and Wolverton 1978, Goel and Okomoto 1979, Ohba and Yamada 1984, Singpurwalla and Wilson 1999), or considered a stochastic model for the discovery rates (Littlewood and Veral 1973, Basu and Ebrahimi 2003, Tamura and Yamada 2007). These models may not be universally applicable as the assumed parametric forms do not always calibrate to the unknown usage rate of the software. Tamura and Yamada (2007) consider the analysis of a bug database from an open source software by using a parametric stochastic process to model the unknown usage rate. Wang et al. (2007) considered nonparametric estimation of the defect discovery rate under a NHPP model using kernel regression. Their model could be more appropriate for user-driven defect discovery when compared to parametric models, but an appropriate reliability metric is difficult.

The approach presented in this article is different from existing models as it uses the classification of defects into multiple types and introduces the usage rate as a nonparametric time dependent component of the model. Also, the notion of reliability that we will develop relates to the incidence of one or more special types of defects (e.g., security or crash related defects) instead of software failures due to defects assumed to be equally important irrespective of their types. Unequal discovery and reporting rates for different defect types are modeled through the use of multivariate counting process with different intensity functions. The effect of time dependent usage rates is incorporated through a proportionality model of the intensity functions. In order to remove the confounding effect of the unobserved time dependent usage rates, a partial likelihood method (Cox 1975) is considered by conditioning on the calendar times when new defects are discovered and reported. As a result, reliability metrics are not defined on a time scale, as is usually done (Singpurwalla and Wilson 1994, Mathur 2008 pp. 10), but on the scale of the number of new discoveries. Specifically, reliability is defined as the probability of discovering no new defects of a special type (e.g., crash related defects) out of, say, N new defects to be discovered in the future, regardless of the time required for their discovery. Also, analogous to mean time to failure (MTTF), we consider the expected number of additional defects that need to be observed to discover a defect of a special type for the first time. In this context, expected number of new defects of a special type out of, say, N new defects to be observed may also be of interest.

2 The Modeling and the Method

Usually a bug database provides information on each defect that is discovered and reported. For each new defect, there is a record carrying information on (i) time of reporting (typically the calendar time of when the defect was first reported), (ii) the version of the software in which the defect was found, (iii) priority of releasing a fix to the defect (urgent, high, medium and low), (iv) software components affected by the defect and (v) classification of the defect in one of several types. Most defect classification schemes contain security and crash related defects. From now on we will write reporting of a defect to mean "discovering and reporting" of the defect by a user of the software.

Consider data related to all new defects reported up to a certain calendar time S from the release date of a software. This may be represented as the sequence of tuples $(T_1, Z_1), \ldots,$ (T_n, Z_n), where T_i is the reporting time of the i^{th} new defect reported since the release of the software and Z_i is the type of the defect, for $i = 1, ..., n$, where n is the number of new defects reported till time S. Suppose there are m special types of defects, denoted by $1, ..., m$, and all other types are pooled into one "baseline" type denoted by 0. Then

Z_i takes values in $\{0, \cdots, m\}$. Alternatively, such a data set can be represented by the sequence of tuples $(T_1, \mathbf{N}(T_1)) \ldots, (T_n, \mathbf{N}(T_n))$, where the vector $\mathbf{N}(t) = [N_0(t), \ldots, N_m(t)]$ denotes the multivariate counting process with $N_i(t)$ being the number of new defects of type i reported upto and including time t, for $i = 0, \cdots, m$.

2.1 The Model

Existing software reliability models for software failure data usually incorporate the debugging process directly or indirectly into the failure rate. Since a bug database contains records corresponding only to new defects that are reported, the debugging process can be ignored for the purpose of modeling such data. The modeling may be done through the multivariate counting process $\mathbf{N}(t) = [N_0(t), \ldots, N_m(t)]$. A natural model is to describe each $N_i(t)$ as a self-exciting point process with intensity function $\lambda_i(t)$.

The intensity of discovering a new software defect at time t is a function of the number of defects already discovered up to time t along with the type of defect and usage rate of the software. The self-exciting nature of the process $N_i(t)$ is due to possible dependence of $\lambda_i(t)$ on the history of the multivariate process $\mathbf{N}(t)$. Suppose, for each $N_i(t)$, there is an underlying intensity function $\gamma_i(t)$ depending only on the usage rate and reporting rate of the ith type of defect. The intensity $\gamma_i(t)$ may be calibrated to model the effect of the history of the process $\mathbf{N}(t)$ on the intensity process $\lambda_i(t)$ through a proportional intensity model as given by

$$\lambda_i(t) \quad = \quad \gamma_i(t) f_i(\mathbf{N}(t-)), \text{ for } i = 0, \cdots, m, \tag{1}$$

where $\gamma_i(t)$ is considered to be unknown and arbitrary and $f_i() \geq 0$ with $f_i(\mathbf{0}) = 1$. We intend to model $f_i(\mathbf{N}(t))$ parametrically leading to a semiparametric model for $\lambda_i(t)$. In particular, if $f_i(\mathbf{N}(t))$ depends on $\mathbf{N}(t)$ only through $N_i(t)$, the component processes $N_i(t)$'s are independent. Note that the set of functions $\{f_i(\mathbf{N}(t)), i = 0, \cdots, m\}$ can be used to obtain reliability metrics that are agnostic to the usage and reporting rates of the defects. For example, the rate at which f_i or $log(f_i)$ decreases for every additional defect discovery, can be interpreted as a measure of reliability of the software with respect to the ith defect type (See Section 2.4). A further simplification can be achieved by assuming that each $\gamma_i(t)$

is proportional to a common rate parameter $\gamma(t)$ across all types of defects. This assumption greatly simplifies the estimation of model parameters apart from enabling inference on the reliability with respect to a given type of defect. Therefore, with $\gamma_0(t) = \gamma(t)$, we may assume

$$\gamma_i(t) = \gamma(t)e^{\alpha_i}, \text{ for } i = 1, \cdots, m. \tag{2}$$

The parameters $\alpha_i's$ can be interpreted as the differences in reporting rates of different types of new defects. Noting that $f_i(\mathbf{N}(t))$ represents multiplicative changes in $\lambda_i(t)$ due to reporting of different types of new defects, it may be natural to assume that $f_i(\mathbf{N}(t))$ decreases with each component of $\mathbf{N}(t)$. However, because of the user-reported nature of new defects, the nature of this decrease is not clear. There may be several choices for $f_i(\mathbf{N}(t))$. In this context, one may recall the Jelinski and Moranda (1972) model for software testing data, which postulates a linear decrease in the intensity of reporting a new defect with the number of already detected defects, and consider a linear decrease model as given by

$$f_i(\mathbf{N}(t)) = 1 - \sum_{j=0}^{m} \beta_{ji} N_j(t). \tag{3}$$

.

Alternatively, a non-linear decrease model in the spirit of the logarithmic Poisson model (Musa and Okumoto 1975), as given by

$$f_i(\mathbf{N}(t)) = exp\left(-\sum_{j=0}^{m} \beta_{ji} N_j(t)\right) \tag{4}$$

may also be considered. In the special case, when the component processes $N_i(T)$'s are independent, it would be appropriate to consider $f_i(\mathbf{N}(t)) = exp(-\beta_i N_i(t))$ with a scalar β_i, for $.i = 0, \ldots, m$. The two choices in (3) and (4) result in

$$\lambda_i(t) = \gamma(t)exp(\alpha_i)\left[1 - \sum_{j=0}^{m} \beta_{ji} N_j(t-)\right] \tag{5}$$

and

$$\lambda_i(t) = \gamma(t)exp\left(\alpha_i - \sum_{j=0}^{m} \beta_{ji} N_j(t-)\right), \tag{6}$$

respectively, for $i = 0, 1, 2 \ldots, m$, with $\alpha_0 = 0$. Write $\alpha = \{\alpha_1, \cdots, \alpha_m\}$, $\beta = \{\beta_0, \cdots, \beta_m\}$ with $\beta_i^t = [\beta_{0i}, \cdots, \beta_{mi}]$ and $\gamma(.)$ as the infinite dimensional intensity $\gamma(t)$ to determine the set of unknown parameters of the model. The logarithmic Poisson model has been noted as an extensively applied software reliability model (Farr 1996) and shown to provide accurate predictions for large software systems (Jones 1991, Derriennic and Le Gall 1995), leading us to use it as the underlying model of choice for illustration of the method developed in this article.

2.2 Partial Likelihood Estimation

Let us write $T^{(j)} = (T_j, T_{j-1}, \ldots, T_1)$ and $Z^{(j)} = (Z_j, Z_{j-1}, \ldots, Z_1)$. The likelihood of the data sequence $(T_1, Z_1), \ldots, (T_n, Z_n)$ is proportional to

$$\prod_{j=1}^{n} P_{T_j|T^{(j-1)},Z^{(j-1)}}(.|\alpha, \beta, \gamma(.)) \times \prod_{j=1}^{n} P_{Z_j|T^{(j)},Z^{(j-1)}}(.|\alpha, \beta). \tag{7}$$

The second product in the above equation, under the modeling assumptions (1) and (2), does not depend on $\gamma(t)$, and is the partial likelihood for estimating α and β (Cox 1975). Note that the conditional probability $P_{Z_j|T^{(j)},Z^{(j-1)}}(.|\alpha, \beta)$ in the second product can be derived as the following multinomial probability under assumptions (1) and (2):

$$P(Z_j = i|T^{(j)}, Z^{(j-1)}) = \frac{e^{\alpha_i} f_i(\mathbf{N}(T_j-))}{\sum_{k=0}^{m} e^{\alpha_k} f_k(\mathbf{N}(T_j-))}, \tag{8}$$

for $i = 0, 1, \ldots, m$. In the special case given by (4), this partial likelihood simplifies to the likelihood of the multinomial logistic regression model as given by,

$$\prod_{j=1}^{n} P(Z_j|T^{(j)}, Z^{(j-1)}) = \prod_{j=1}^{n} \prod_{i=0}^{m} \left(\frac{e^{\alpha_i - \beta_i' \mathbf{N}(T_j-)}}{\sum_{k=0}^{m} e^{\alpha_k - \beta_k' \mathbf{N}(T_j-)}} \right)^{I(Z_j=i)}. \tag{9}$$

Maximizing the partial likelihood to estimate α and β can now be performed through a multinomial logistic regression between Z_j and the vector $\mathbf{N}(T_j-)$. A Newton-Raphson procedure can be used to maximize the log partial-likelihood in order to estimate the parameters. Alternatively, any standard software package to analyse a multinomial logistic regression model (e.g., *multinom* function in *nnet* package of R) can be used. The estimated asymptotic variance-covariance matrix of the parameter estimates can be obtained from the observed information matrix based on the partial likelihood. The asymptotic normality of the partial likelihood estimates (Wong 1986) can be used to perform tests of significance and obtain confidence intervals for the parameters of interest and functions thereof.

2.3 Goodness of Fit

The proposed modeling consists of the forms for the set of functions $\{f_i(\mathbf{N}(t)), i = 0, \ldots, m\}$ as defined in (1) and the proportionality of the baseline rates $\gamma_i(t)$'s as defined in (2). Violations of these assumptions would lead to $P\left(Z_j|T^{(j)}, Z^{j-1}\right)$ being incorrectly specified. Hence, it suffices to check the goodness of fit for the form of $P\left(Z_j|T^{(j)}, Z^{j-1}\right)$ for different j. In the special case when the model is being fit with only two types of defects (i.e., Z_j takes binary values) and the choice of $f_0(\mathbf{N}(t))$ and $f_1(\mathbf{N}(t))$ is the logarithmic Poisson form as given in (4) leading to (9), the Hosmer and Lemeshow (1980) test can be used to assess the goodness of fit of $P\left(Z_j|T^{(j)}, Z^{j-1}\right)$. For data sets with more than two types of defects, with the logarithmic Poisson form for $\{f_i(\mathbf{N}(t)), i = 0, \ldots, m\}$, we would need to assess the goodness of fit of a multinomial logistic regression model. For this, we appeal to tests proposed by Begg and Gray (1984), Pigeon and Heyse (1999), or Fagerland et al. (2008).

For a graphical check, one may consider comparing $\sum_{j=1}^{k} P\left(Z_j = i|T^{(j)}, Z^{(j-1)}\right)$ with the observed number of defects of type i up to time T_k, given by $N_i(T_k)$. A plot of $\sum_{j=1}^{k} \hat{P}\left(Z_j = i|T^{(j)}, Z^{(j-1)}\right)$ (as predicted) and $N_i(T_k)$ (as observed) over k can be used to visually check the goodness of fit, where $\hat{P}(.|.)$ denotes the estimate of the corresponding $P(.|.)$, evaluated with the parameter estimates.

2.4 Reliability Metrics

As mentioned earlier, certain defect types are of more importance to the software community than others, an example being defects which are related to security loopholes or crashes in the software. A traditional metric for the reliability of a software with respect to, say, crash defects would be the probability of observing no crash defects in the next one year, or alternatively the mean time to observing a crash defect. These metrics depend upon the usage rate of the software. For example, if the usage of the software were to increase manifold, these metrics would no longer be correct. Hence, there is a need for a metric that does not depend upon future usage rate of the software. An intuitive metric that does not depend upon the usage rate is the probability of discovering no crash related defect (corresponding to, say, type m) in the next N defects of any type. Let us denote this by $R(N)$. This metric, under the modeling of Section 2, would not depend upon the usage rate of the software. Similarly, we can also think of the expected number of defects to be observed before observing a new crash related defect, denoted by $MNDF$, and use it as an alternative to mean time to failure ($MTTF$).

Theoretical derivations of these reliability metrics in general can be a challenge because of the stochastic nature of the model (1). For example, the probability of no crash related defect requires a huge sum of m^N joint probability terms each corresponding to an ordered set of N defects of types other than crash related defect. This may be calculated when N is small; however, even for moderate N (say, 10 or 15) and with $m = 2$ or 3, this is computationally difficult. One option is to pool all other defect types into one type resulting in only two types (with $m = 1$) so that there is only one joint probability term as given by

$$R(N) = \prod_{j=n+1}^{n+N} P\left(Z_j = 0 | T^{(j)}, Z^{(j-1)}; \alpha, \beta\right). \tag{10}$$

Note that $Z^{(j-1)}$ in the conditioning event in each term of (10) is given by $(Z_1, \ldots, Z_n, Z_{n+1} = 0, \ldots, Z_{j-1} = 0)$. Also, each such term is independent of $T^{(j)}$, except through $\mathbf{N}(T_j)$ as in (8). In this case, the other reliability measure MNDF can be calculated as the infinite sum

$$MNDF = \sum_{l=n+1}^{\infty} \prod_{j=n+1}^{l} P\left(Z_j = 0 | T^{(j)}, Z^{(j-1)}; \alpha, \beta\right). \tag{11}$$

For the independent model with $f_i(\mathbf{N}(t)) = exp(\alpha_i - \beta_i N_i(t))$, for $i = 0, 1$, with $\alpha_0 = 0$, the individual probability term in (10) and (11) is given by

$$P(Z_j = 0 | T^{(j)}, Z^{(j-1)}, \alpha, \beta) = \frac{e^{-\beta_0(n_0+j-1)}}{e^{-\beta_0(n_0+j-1)} + e^{\alpha_1 - \beta_1 n_1}}, \tag{12}$$

for $j = n+1, n+2, \ldots$, where $n_i = N_i(T_n)$, for $i = 0, 1$. Since these probability terms are decreasing in j, the MNDF in (11) has a finite value. In the special case, when $f_0(\mathbf{N}(t)) = 1$, or $\beta_0 = 0$, meaning that there is no effect of history on $\lambda_0(t)$, we have $MNDF = exp(-\alpha_1 + \beta_1 n_1)$. In this special case, $R(N)$ simplifies to $[1 + exp(\alpha_1 - \beta n_1)]^{-N}$. The metrics in (10) and (11) are to be estimated by evaluating them at the partial likelihood estimates of α, β. The standard errors of these estimates can be obtained by applying the delta method.

In general, both $R(N)$ and $MNDF$ can be estimated by simulating Z_{n+i}'s successively using (8) and the estimates of the model parameters. For example, the reliability metric, $R(N)$ can be estimated by the proportion of times none of $Z_{n+1}, \ldots, Z_{n+N}$ equals m over, say, $M = 1000$ simulations of $Z_{n+1}, \ldots, Z_{n+N}$. Similarly, for the reliability metric MNDF, one needs to simulate Z_{n+i}'s till $Z_{n+L+1} = m$, where L is the minimum number of detected defects before observing a type m defect in the simulation. Then, MNDF is estimated by the mean of L values over the M simulations. If a simulation approach were to be used for estimating the metrics, standard errors can be obtained by using a parametric bootstrap method (Efron and Tibshirani 1986) as follows. Simulate the bth bootstrap sample comprising of the defect types $(Z_j^{(b)}, j = 1, 2, \cdots, n)$ using the estimates of α and β, and (8). Based on this bootstrap sample, obtain estimates $\hat{\alpha}^{(b)}$ and $\hat{\beta}^{(b)}$ of α and β, respectively, using the method of Section 2.2. Estimate the reliability metrics using $\hat{\alpha}^{(b)}$ and $\hat{\beta}^{(b)}$ by the method of simulation as described above and denote them by $R(N)^{(b)}$ and $MNDF^{(b)}$, respectively. Repeat the bootstrap process for, say, $B = 500$ times. The standard deviations of the estimated reliability metrics $R(N)^{(b)}$ and $MNDF^{(b)}$ over the B bootstrap samples estimate their corresponding standard errors.

3 A Simulation Study

While planning a simulation study, it is to be noted that, for the purpose of estimating the model parameters, information on only the defect types $Z_1, \cdots, Z_n$ is needed and the same on the reporting times $T_1, \cdots, T_n$ may be ignored. Also, while estimating the reliability metrics $R(N)$ and $MNDF$ by simulation, it is enough to simulate only the future defect types $Z_{n+1}, Z_{n+2}, \ldots$. Note that simulation of Z_i's can be successively carried out using (8). In our simulation study, we consider two types of defects (i.e., $m = 1$) and type 1 is assumed to correspond to an important defect type such as a crash related defect. We consider the independent logarithmic Poisson model given by $f_i(\mathbf{N}(t)) = exp(-\beta_i N_i(t))$, for

Table 1: Results of the simulation study.

	n=500				n=2000			
Parameter	True	Estimate	SE×10	Coverage	True	Estimate	SE×10	Coverage
α	3.00	3.56	7.40	91.4%	3.00	3.22	0.41	93.1%
β_0	0.03	0.06	0.37	88.2%	0.03	0.03	0.08	92.1%
β_1	0.01	0.02	0.08	88.3%	0.01	0.01	0.02	92.2%
R(N)	0.06	0.08	0.38	95.0%	0.05	0.05	0.16	95.6%
MNDF	3.16	3.51	7.20	96.5%	2.91	2.95	3.50	95.4%

$i = 0, 1$, with $\beta_0 = 0.03$, $\beta_1 = 0.01$ and the type-specific differential rate parameter $\alpha_1 = 3$. For each simulation, $Z_1, , \ldots, Z_n$ are generated using (8) with $n = 500$ and 2000 to reflect moderate to large sample sizes. The maximum partial likelihood estimates of α_1, β_0 and β_1 are then obtained by maximizing (9) along with the corresponding standard errors and the asymptotic 95% confidence intervals of the parameters using normal approximation. Reliability metrics $R(N)$ (with $N = 10$) and $MNDF$ are also estimated using the simplified forms in (10) and (11), respectively, with each probability term given by (12), along with their standard errors and asymptotic 95% confidence intervals. This simulation is repeated 1000 times and mean and standard deviation of the estimates over the 1000 simulations are obtained. The coverage probabilities of the asymptotic 95% confidence intervals are also estimated. Results of this simulation study are presented in Table 1. As the average of the standard errors over 1000 simulations is close to the standard deviation of the corresponding estimates over the 1000 simulations, we report only the latter quantity for the measure of standard error. The true values corresponding to $R(N)$ and $MNDF$ are computed from (10) and (11) with (12) evaluated at the true parameter values, given the simulated $Z^{(n)}$ giving values of n_0 and n_1. The corresponding values in Table 1 are averages of those over the 1000 simulations. The results indicate consistency of the estimates and the standard errors (SE) decrease with n, as expected. Also, the estimated coverage probability is closer to 95% for larger n. Interestingly, the estimated coverage probability for the reliability metrics turns out to be better than those for the model parameters.

In order to understand the effect of model mis-specification for $f_i(\mathbf{N}(t))$ on the estimation of the reliability metrics, we consider simulation of $Z^{(n)}$ from the independent linear decrease model with two types of defects, given by $f_i(\mathbf{N}(t)) = 1 - \beta_i N_i(t)$, for $i = 0, 1$. We assume $\beta_0 = 0.001$, $\beta_1 = 0.00005$ and $\alpha_1 = 0.5$ for the purpose of simulation. This parameter setting implies that there are $1/\beta_0 = 1000$ bugs of type 0 and $1/\beta_1 = 20000$ bugs of type 1 in the beginning. For each simulation of $Z^{(n)}$ with $n = 1000$, estimation of the model parameters α_1, β_0 and β_1 is carried out by fitting the incorrectly specified independent logarithmic Poisson model. Reliability metrics $R(N)$ with $N = 10$ and $MNDF$ are estimated by using (10) and (11), respectively, with each probability term given by (12), as before. For the purpose of comparison, we also compute (10) and (11) but with each probability term given by the correct model and using the true parameter values and simulated $Z^{(n)}$. The averages of these two sets of estimates of $R(N)$ and $MNDF$, over 1000 simulations, are compared to understand the effect of model mis-specification. While the estimates for $R(N)$ and $MNDF$ obtained by using the correct model are 0.11 and 4.17, respectively, the corresponding estimates obtained by using the incorrect model are 0.10 and 3.85, with standard errors 0.03 and 0.61, respectively. Therefore, in this limited study, the effect of model mis-specification on the estimates of $R(N)$ and $MNDF$ seems to be minimal.

4 Analysis of Python Software[1]

Python is a general purpose scripting language that is extensively used in a variety of applications. It is an open source software which is maintained and developed by it's community. The Python project maintains a bug database that records defects in the software reported by its user community. When a defect is reported by the user community, it is classified and tagged with auxiliary information. Every bug reported by the user community is classified into multiple types. Defects of type "crash" or "security" are of significant importance. We shall analyze the corresponding bug database to estimate the reliability metrics for these two types of defects.

Python's bug database provides a method for querying its database (http://bugs.python.org/issue?@template=search). Only those defects whose resolution

[1]Python is a trademark of the Python Software Foundation

was “fixed” or “fixed and accepted” as on 31 January 2012 are retrieved. This is because defects with other resolutions comprise of duplicates of already reported defects or, defects that have not been confirmed as genuine. In addition to the date of first reporting of each defect, information on the defect type is also extracted. Python version 2.7 has 2273 reported defects, while Python version 2.6 has 1975 defects reported until 31st Jan 2012. Table 2 contains a summary of software defects for Python versions 2.7 and 2.6 until 31 January 2012.

Table 2: Summary of defects in the two versions of Python

	Python 2.7		Python 2.6	
Defect Type	Count	Percentage	Count	Percentage
Crash	130	5.7 %	124	6.3 %
Security	19	0.8%	16	0.8%
Others	2124	93.5 %	1835	92.9%
Total Bugs	2273	100%	1975	100 %

We first analyze the data using all three types of defects, namely, crash defects, security defects and defects of other types. Let us denote these defect types as type 2, type 1 and type 0, respectively. We use the independent logarithmic Poisson model given by $f_i(\mathbf{N}(t)) = exp(\alpha_i - \beta_i N_i(t))$, for $i = 0, 1, 2$, with $\alpha_0 = 0$. For the analysis of data from each version using the method of Section 2.2, we only need the vector $Z^{(n)} = (Z_1, \ldots, Z_n)$ with $n = 2273$ and 1975 for versions 2.7 and 2.6, respectively, where each Z_i takes values 0,1 or 2. The parameters $\alpha_1, \alpha_2, \beta_0, \beta_1, \beta_2$ are estimated along with their variance-covariance matrix, by performing a multinomial logistic regression between Z_j and n_{0j}, n_{1j}, n_{2j}, where $n_{ij} = N_i(T_j-) = \sum_{l=1}^{j-1} I(Z_l = i)$ is the number of type i defects reported till prior to time T_j, for $i = 0, 1, 2$. These parameter estimates are used to estimate reliability metrics $R(N)$ with $N = 10$ and $MNDF$ using simulation, as described in the end of Section 2.4. The corresponding standard errors are obtained by using parametric bootstrap method, as described therein. The estimates of the parameters and the reliability metrics along with their standard errors are presented in the top panel of Table 3. Note that these estimated reliability metrics are for crash related defects. Similar estimates for security related defects are also obtained, but not reported here.

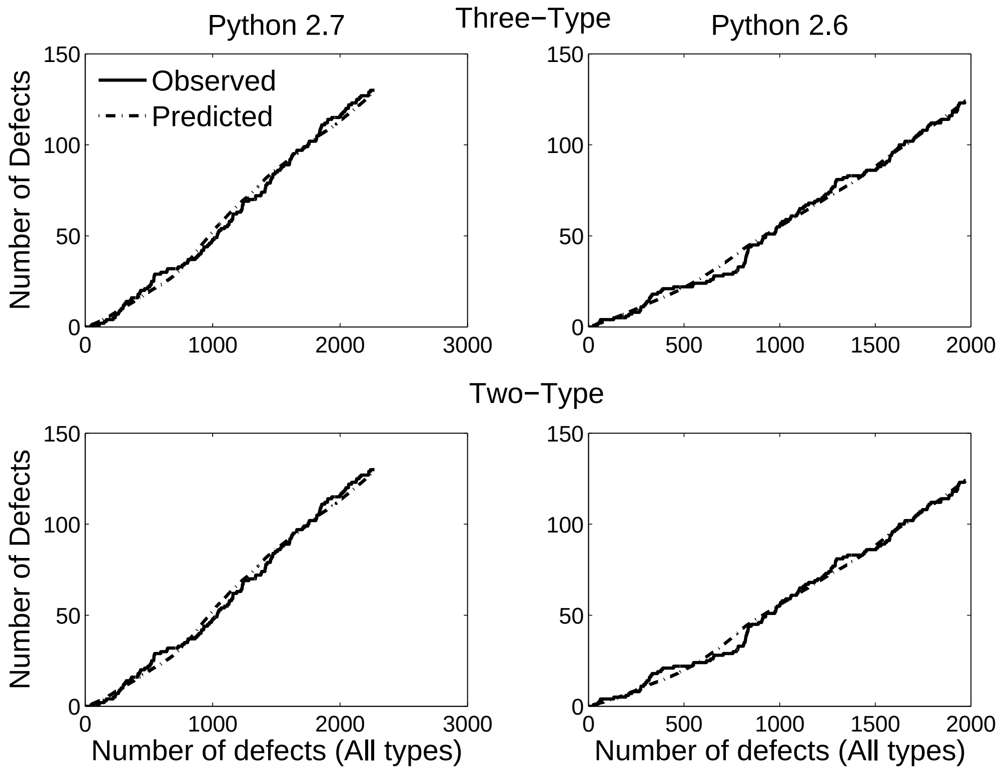
Table 3: Estimates of the parameters and reliability metrics with their standard errors.

		Python 2.7		Python 2.6	
Analysis	Parameters	Estimate	SE	Estimate	SE
Three-type	α_1	-5.60	0.49	-5.88	0.53
	α_2	-3.87	0.32	-3.36	0.26
	$\beta_0(\times 10^2)$	0.56	0.15	0.25	0.11
	β_1	0.64	0.18	0.22	0.15
	β_2	0.08	0.02	0.03	0.01
	$R(N)$	0.43	0.10	0.49	0.13
	$MNDF$	12.02	2.67	15.06	5.50
Two-type	α_1	-3.85	0.37	-3.56	0.33
	$\beta_0(\times 10^2)$	0.55	0.18	0.35	0.15
	β_1	0.08	0.02	0.04	0.02
	$R(N)$	0.44	0.07	0.47	0.07
	$MNDF$	11.23	2.09	12.28	2.26

From the estimates of $R(N)$ and $MNDF$, Python 2.6 seems more reliable with respect to "crashes" when compared to Python 2.7. However, by looking at the standard errors, we cannot conclude that the reliabilities of the two versions of the software are significantly different. Note that the coefficients $\beta_0, \beta_1, \beta_2$ are all statistically significant except for β_1 in the analysis of Python 2.6 data. These β-coefficients are positive implying that every new defect discovery decreases the probability of detecting more defects of the same type, as expected. Also, in the analysis of both versions of the software, the parameters α_1 and α_2, which correspond to differences in the reporting rates of security and crash related defects, respectively, when compared to other defect types, are significantly different with $\alpha_2 > \alpha_1$. This indicates that security related defects are less likely to be reported when compared to crash related defects and, surprisingly, to other types of defects as well. Note that the parameters α_1 and α_2 are not significantly different across the two versions of the software, while they themselves are significant.

The goodness of fit for a multinomial logistic regression model, as suggested by Fagerland et. al (2005), is carried out to assess the fit of the form of $P\left(Z_j|T^{(j)},Z^{(j-1)}\right)$, as derived from the assumed independent logarithmic Poisson model. The test is conducted by splitting the estimated $P\left(Z_j=2|T^{(j)},Z^{(j-1)}\right)$, for $j=1,2,\ldots,n$, into ten probability deciles (i.e., the crash related defect is used as the grouping variable), which leads to an asymptotic chi-square distribution for the test statistic with 16 degrees of freedom. The corresponding p-value for Python 2.7 is 0.08, while the same for Python 2.6 is 0.02. This indicates that the proposed three-type defect model fits reasonably well for Python 2.7 while the fit for Python 2.6 may be questionable. To visually assess the fit of the model, plots of $\sum_{j=1}^{k}\hat{P}\left(Z_j=i|T^{(j)},Z^{(j-1)}\right)$ and n_{ik} over k (See Section 2.3), for $i=0,1,2$, are considered. The top panel of Figure 1 shows the plots of crash related defects (i.e., for $i=2$) for both versions of the software. The plots indicate reasonable fit of the model for crash related defects.

Figure 1: Plots of predicted and actual numbers of crash related defects.



Apparent lack of fit of the three-type model to the data from Python 2.6 with non-significance of β_1 corresponding to security related defects in the three-type analysis, and the simplicity of the two-type model for the calculation of reliability metrics (see Section 2.4) lead us to pool security related defects with other types, and consider a two-type model involving just crash related defects ($i = 1$) and other defects ($i = 0$). The independent logarithmic Poisson model given by $f_i(\mathbf{N}(t)) = exp(\alpha_i - \beta_i N_i(t))$, for $i = 0, 1$, with $\alpha_0 = 0$, is once again used. As in the three-type analysis, we only need the vector $Z^{(n)} = (Z_1, \ldots, Z_n)$, where each Z_i takes values 0 or 1. The parameters α_1, β_0 and β_1 are estimated along with their variance-covariance matrix, by performing a logistic regression between Z_j and n_{0j}, n_{1j}, where n_{ij}'s are similar to those defined before but with the new labeling of type ($i = 0, 1$). The estimates of the parameters and the reliability metrics $R(N)$, with $N = 10$, and $MNDF$, along with their standard errors, are presented in the bottom panel of Table 3. In order to assess the fit of the form of $P\left(Z_j | T^{(j)}, Z^{(j-1)}\right)$, the Hosmer-Lemeshow test for logistic regression is performed. The test is conducted by splitting the estimated $P\left(Z_j = 1 | T^{(j)}, Z^{(j-1)}\right)$ into ten probability deciles, which leads to an asymptotic chi-square distribution for the test statistic with 8 degrees of freedom. The corresponding p-values for Python 2.7 and 2.6 are 0.45 and 0.55, respectively. This indicates a good fit of the model to the data. To visually assess the fit, plots of $\sum_{j=1}^{k} \hat{P}\left(Z_j = 1 | T^{(j)}, Z^{(j-1)}\right)$ and n_{1k} over k are considered and shown in the bottom panel of Figure 1. The plots indicate a reasonably good fit of the model for both the versions. Comparison of the reliability metrics for crash related defects across the two versions, along with their standard errors, indicates that these are not significantly different across the two versions. It is interesting to note that the estimated reliability metrics for crash related defects are quite similar to those obtained from the three-type analysis presented earlier. The parameters β_0 and β_1 turn out to be statistically significant and positive for both versions of the software, as before. The estimate of β_1 is similar to the corresponding estimate (i.e., of β_2) in the three-type analysis. The estimate of α_1 is also similar to the corresponding estimate (i.e., of α_2) in the three-type analysis.

5 Concluding Remarks

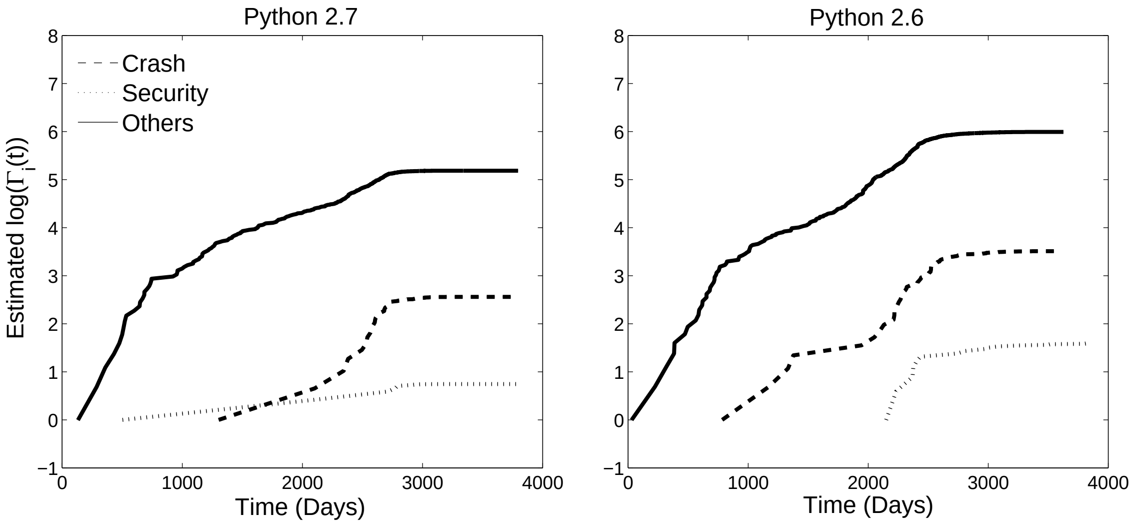
Bug databases, which record user reported defects, have become a norm for commercial software. Statistical models for analyzing this increasingly important data source will help end-users to objectively assess the reliability of a software. This paper makes an attempt in that direction by considering, possibly for the first time, an analysis of software defects with multiple types of classification. Usage and reporting rate of the software, which is modeled by an infinite dimensional confounding factor $\gamma(t)$ in the analysis of such data, is considered non-parametrically. The partial likelihood approach facilitates estimation of important model parameters. An added advantage of the proposed method is that it can be carried out using standard statistics software packages for easy implementation by practitioners. The proposed reliability metrics are easily interpretable and can be used to compare different software versions. For example, in the analysis of Python software, we gather from the estimate of $MNDF$ for Python 2.7 that, on an average, about 12 non-crash related defects will be discovered before the discovery of a crash related defect, which is about the same when compared to that of Python 2.6. This indicates that there might not be any additional gain in reliability with respect to "crashes" in using Python 2.7 over Python 2.6. Such analysis may help decision makers to objectively choose between migrating from one version of a software to another. It is interesting to note that, in the limited simulation study, the estimated reliability metrics seem to have better asymptotic convergence properties than the model parameters themselves. Note that defects could potentially belong to more than one type. This can be dealt with by considering an additional defect type.

The integrated intensity function $\Gamma(t) = \int_0^t \gamma(u)du$ (See Section 2) may be estimated using a Breslow type estimator (Breslow 1972), as given by, for the logarithmic Poisson model,

$$\hat{\Gamma}(t) = \int_0^t \frac{dN_{.}(s)}{\sum_{i=0}^m exp(\hat{\alpha}_i - \hat{\beta}_i N_i(s-))} ds, \quad 0 < t \leq S, \tag{13}$$

where $N_{.}(t) = \sum_{i=0}^m N_i(t)$. Extrapolation of the estimated $\Gamma(t)$ through a parametric model may be used to compute time based reliability metrics (Wang et al. 2007). This type of estimator may also be used for testing the proportionality assumption (2). For example,

Figure 2: Plot of $log(\hat{\Gamma}_i(t))$ vs t.



the individual integrated intensity $\Gamma_i(t) = \int_0^t \gamma_i(u)du$ may be estimated by

$$\hat{\Gamma}_i(t) = \int_0^t \frac{dN_i(s)}{exp(-\hat{\beta}_i N_i(s-))} ds, \quad 0 < t \leq S, \tag{14}$$

for $i = 0, \ldots, m$. Plots of $log(\hat{\Gamma}_i(t))$ for different i on the same graph should be near parallel if (2) is true. These plots for the three-type analysis of Python software is presented in Figure 2 with time 0 being the date when the first defect was reported. The plots give evidence in favor of the proportionality assumption (2) except for security related defects in version 2.7.

The method considered here may have applications in many areas other than software reliability. As an example, in the context of disease epidemiology, diseased individuals in a geographic location may have different disease types (for example, drug-resistant and non drug-resistant Tuberculosis). The voluntary reporting of a software defect by users corresponds to diseased individuals voluntarily reporting to a hospital to seek treatment, while usage rate may correspond to unknown exposure of the individual. Using an independent logarithmic Poisson model, one could determine whether a particular disease type is spreading or not by checking the sign of the corresponding β_i. The proposed reliability metrics may also be relevant in this epidemiological context. For example, the metric MNDF, could be interpreted as the expected number of patients with the non drug-resistant disease who will report to the hospital before a patient with the drug-resistant disease reports.

References

[1] Basu, A. and Ebrahimi, N. (2003), Bayesian Software Reliability Models Based on Martingale Processes, Technometrics, 54, 150-158.

[2] Begg, C. B. and Gray, R. (1984), Calculation of Polychotomous Logistic Regression Parameters using Individualized Regressions, Biometrika, 71, 11-18.

[3] Breslow, N. (1972), Comment on D. R. Cox (1972) paper, Journal of the Royal Statistical Society: Series B, 34, 216-217.

[4] Cox, D. R. (1975), Partial Likelihood, Biometrika, 62, 269-276.

[5] Derrennic, H. and Le Gall, G. (1995), Use of Failure-Intensity Models in the Software-Validation Phase for Telecommunications, IEEE Transactions on Reliability, 44, 658-665.

[6] Efron, B. and Tibshirani, R. (1986), Bootstrap Methods for Standard Errors, Confidence Intervals and Other Measures of Statistical Accuracy, Statistical Science, 1, 54-77.

[7] Fagerland, M. W. Hosmer, D. W. and Bofin, A. M. (2008), Multinomial Goodness-of-fit Tests for Logistic Regression Models, Statistics in Medicine, 27, 4238-4253.

[8] Farr, W. (1996), Software Reliability Modeling Survey. In Handbook of Software Reliability Engineering, ed. Lyu, M, McGraw-Hill, New York, 71-117.

[9] Goel, A.L. and Okumoto, K. (1978), An Analysis of Recurrent Software Failures on a Real-time Control System, Proceedings of the ACM Annual Technical Conference, Washington D.C., 496-500.

[10] Goel, A.L. and Okumoto, K. (1979), Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures, IEEE Transactions on Reliability, R-28, 206-211.

[11] Hosmer, D. W. and Lemeshow. S. (1980), Goodness of Fit Tests for the Multiple Logistic Regression Model, Communications in Statistics, A9, 1043-1069.

[12] Jones, W. D. (1991), Reliability Models for Very Large Software Systems in Industry, Proceedings of the Second International Symposium on Software Reliability Engineering, 35-42.

[13] Jelinsky, Z. and Moranda P. B. (1972), Software Reliability Research, Statistical Computer Performance Evaluation, ed. Freiberger W., 465-484.

[14] Leszak, M., Perry, D. E. and Stoll, D. (2000), A Case Study in Root Cause Defect Analysis, Proceedings of the 2000 International Conference on Software Engineering, 428-437.

[15] Littlewood, B. and Verall, J. L. (1973), A Bayesian Reliability Growth Model for Computer Software, Applied Statistics, 22, 332-46.

[16] Mathur, A. P. (2008), Foundations of Software Testing, Volume 1, Pearson Education, New Delhi.

[17] Moranda, P.B. (1975), Prediction of Software Reliability and its Applications, Proceedings of the Annual Reliability and Maintainability Symposium, 1975, 327-32.

[18] Musa, J. D. (2005), Software Reliability Engineering, First Edition, Tata McGraw Hill, New Delhi.

[19] Musa, J. D. and Okumoto, K. (1975), A Logarithmic Poisson Execution Time Model for Software Reliability Measurement, Proceedings of the 7th IEEE Conference on Software Engineering, 230-238.

[20] Musa, J.D., lannino, A. and Okumoto, K. (1987), Software Reliability: Measurement Prediction and Application, Wiley, New York.

[21] Pigeon, J. G. and Heyse, J. F. (1999), An Improved Goodness of Fit Statistic for Probability Prediction Models, Biometrical Journal, 41, 71-82.

[22] Schick, G. J. and Wolverton, R. W. (1978), An Analysis of Competing Software Reliability Models, IEEE Transactions on Software Engineering, 104-120.

[23] Singpurwalla, N. D. and Wilson, S. P. (1994), Software Reliability Modeling, International Statistical Review, 62, 289-317.

[24] Singpurwalla, N. D. and Wilson, S. P. (1999), Statistical Methods in Software Reliability, Springer, New York.

[25] Tamura, Y. and Yamada, S. (2007), Software Reliability Growth Model Based on Stochastic Differential Equations for Open Source Software, Proceedings of the 2007 International Conference on Mechatronics, Kumamoto, Japan, 1-6.

[26] Wang, Z., Wang, J. and Liang, X. (2007), Nonparametric Estimation for NHPP Software Reliability Models, Journal of Applied Statistics, 34, 107-119.

[27] Wong, W. H. (1986). Theory of Partial Likelihood, Annals of Statistics, 14, 88-123.

[28] Yamada, S., Ohba, M. and Osaki, S. (1983), S-Shaped Reliability Growth Modeling for Software Error Detection, IEEE Transactions on Reliability, R-32, 475-484.

[29] Yamada, S. and Osaki, S. (1985), Software Reliability Growth Modeling: Models and Applications, IEEE Transactions on software engineering, 2, 1431-1437.

[30] Yang, M. C. K. and Chow, A. (1995), Reliability Estimation & Stopping-Rules for Software Testing, Based on Repeated Appearances of Bugs, IEEE Transactions on Reliability, 44, 315-321.

[31] Yu, W. D. (1998). A Software Prevention Approach in Coding and Root Cause Analysis, Bell Labs Technical Journal, 3, 3-21.