Towards Comprehensive Software Reliability Evaluation in Open Source Software

Hiroyuki Okamura and Tadashi Dohi
Department of Information Engineering
Graduate School of Engineering, Hiroshima University
1–4–1 Kagamiyama, Higashi-Hiroshima 739–8527 Japan
Email: {okamu,dohi}@rel.hiroshima-u.ac.jp

***Abstract*—This paper proposes a unified modeling framework for software reliability assessment in open source project. We combine the classical non-homogeneous Poisson process based software reliability growth model (SRGM) with a familiar regression scheme called the generalized linear model (GLM), and develop a novel framework not only to estimate software reliability measures, but also to investigate impacts of software metrics on the fault-detection process. The resulting GLM-based SRGM involves the common SRGMs as well as some existing metrics-based SRMs, such as logistic-regression-based SRGM and Poisson-regression-based SRGM, and possesses a great data fitting ability. We also provide an effective parameter estimation algorithm based on the EM (Expectation-Maximization) principle. Finally, it is shown through numerical experiments with actual open source project data that our approach can estimate software reliability measures with higher accuracy and can feedback the analysis results to improve the current software development projects.**

I. Introduction

Software reliability is one of the most important aspects of software quality. The software reliability is defined as the property that software continuously provides its service without any failure. During the last four decades, there are many model-based approaches to quantify the software reliability from statistical data. In particular, the software reliability growth model (SRGM) is the most popular to assess the software reliability from the data collected in testing phase [1]–[4].

Essentially, SRGMs are stochastic processes to represent the number of bugs detected in testing phase. In general, as testing progresses, the number of newly detected bugs decreases by fixing the detected bugs. Based on this property, SRGMs can provide the prediction of future behavior of the number of detected bugs. Also, we compute quantitative reliability measures with SRGMs. For example, one of the software reliability assessment activities is to estimate the number of residual bugs. If we know the accurate estimate of the residual faults, it is useful for determining software release timing to the market.

However, most of the existing SRGMs focus only on the event of bug detection, i.e., SRGMs can represent the number of bugs in testing phase, regardless of test activity and software design. Although this property makes it easy to handle the SRGMs, it is pointed out that the prediction with SRGM is not always accurate due to lack of information. On the other hand, in recent open source projects, we can utilize a variety of information on project activities and software products. In other words, by considering the other information in addition to the number of detected bugs, the accuracy of SRGMs is expected to be improved. Concretely, this paper focuses on how to use the information on software metrics in the context of SRGM.

Several papers have attempted to use software metrics regarding software source codes and testing efforts. Yamada et al. [5] presented a stochastic model using testing efforts by applying the concept of ordinary S-shaped SRGMs. Shibata et al. [6] proposed a multi-factor SRGM based on the discrete Cox's proportional hazard model [7]. Their model can represent the reliability growth phenomenon by using multiple testing metrics. In [6], the authors showed that the multi-factor SRGM is superior to the existing SRGMs in terms of estimation accuracy. Okamura et al. [8] proposed an SRGM using logistic regression, so-called the logistic-regression-based SRGM, with test efforts and indicated that the logistic-regression-based SRGM has almost same data fitting ability as Shibata's model [6]. Kuwa and Dohi [9], [10] also presented generalization of metrics-based SRGM [6]. Essentially, their generalization is to apply different link functions to logistic regression.

The above papers deal with the test efforts, i.e., one of the metrics on test activity. Dissimilar to these works, some papers focused on the software design metrics. Apart from the SRGM, Khoshgoftaar and Munson [11], Khoshgoftaar et al. [12], [13], Evanco and Locovara [14] and Evanco [15] discussed how to use source code metrics such as complexity and coupling of modules toward the software error prediction. Based on these idea, Okamura and Dohi [16] proposed an SRGM handling the software design metrics with Poisson regression. In [16], it is assumed that the design metrics affect the mean number of total bugs. By using the information on software design, Okamura and Dohi [16] show that the prediction ability of SRGM in early phase of testing can be improved.

This paper considers the metrics on both software deign and test activity. That is, we propose a generalized modeling framework including the logistic-regression-based SRGM [8] and the Poisson-regression-based SRGM [16] by applying the generalized linear regression model (GLM). From the viewpoint of mathematics, the proposed SRGM here is a natural extension from both SRGMs. On the other hand, since the proposed GLM-based SRGM can handle both software design metrics and test activity metrics, it becomes more complicated than the logistic-regression-based SRGM [8] and the Poisson-regression-based SRGM [16]. Thus we provide the efficient parameter estimation based on EM (expectation-maximization)

978-1-5090-0406-5/15/$31.00 ©2015 IEEE

algorithm. The proposed modeling and statistical framework will give a great impact on the model-based software reliability assessment.

This paper is organized as follows. In Section II, we introduce the ordinary non-homogeneous Poisson process based SRGM. In particular, we make some mathematical assumptions to develop the GLM-based SRGM. In Section III, we propose the GLM-based SRGM. After describing the regression analysis with GLM, we build the GLM-based SRGM under the mathematical assumptions. Section IV presents the statistical inference of GLM-based SRGM. By applying the EM algorithm, we give an estimation algorithm which consists of Poisson regression and binomial regression. Moreover, we also describe the variable selection method in GLM-based SRGM. In Section V, we devote to numerical analysis of open source projects using GLM-based SRGM. In experiments, we investigate the data fitting ability of GLM-based SRGM and the effect of metrics on software reliability measure. Finally the paper is concluded with remarks in Section VI.

II. SRGM

The software reliability growth model (SRGM) is one of the most classical models to evaluate quantitative software reliability. Goel and Okumoto [17] presented the earliest model to describe the number of software bugs detected in testing phase as a non-homogeneous Poisson process (NHPP). Let $N(t)$ be a counting process on the continuous time domain, which represents the cumulative number of software bugs detected before time t. When $N(t)$ is an NHPP, the probability mass function of $N(t)$, i.e., the probability that n bugs have been detected before time t, is given by

$$P(N(t)=n)=\frac{\Lambda(t)^n}{n!}\exp\left(-\Lambda(t)\right), \quad n=0,1,\ldots, \quad (1)$$

where $\Lambda(t)$ denotes the mean number of failures before time t and is called the mean value function of NHPP. Generally, it is known that the number of bugs decreases as testing progresses. Goel and Okumoto [17] assumed the following mean value function:

$$\Lambda(t)=a(1-e^{-bt}), \quad (2)$$

where a (>0) and b (>0) correspond to the mean number of total bugs and bug detection rate per unit time, respectively. Figure 1 illustrates the cumulative number of bugs reported in testing phase of a software project and the mean value function of Goel and Okumoto model with estimated parameters from reported data. As seen in the figure, by finding suitable parameters, we can predict the future behavior of the number of bugs. Also, since SRGMs can evaluate the maturity of software testing by estimating the number of remaining bugs, they are often used to make a decision of software release [18].

Since the Goel and Okumoto's work, many papers proposed a number of SRGMs with different mean value functions such as the delayed S-shaped model [19]. Langberg and Singpurwalla [20] presented a modeling framework to integrate almost all SRGMs under the following assumptions:

- The number of total bugs is finite and follows a Poisson random variable M with mean a.

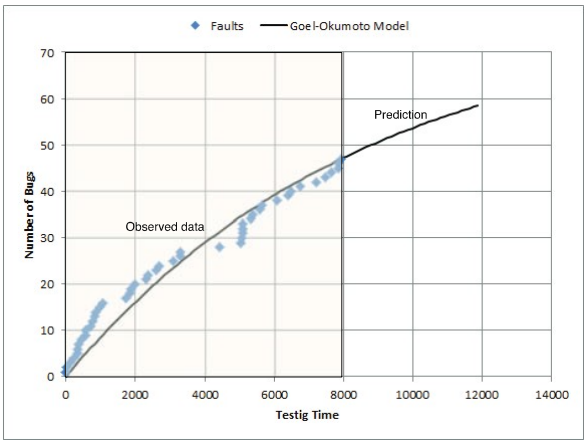

Fig. 1. An example of the SRGM fitting to data.

TABLE I. RELATIONSHIP BETWEEN SRGMS AND BUG-DETECTION TIME DISTRIBUTIONS.

SRGM	Distribution
Goel and Okumoto model [17]	Exponential
Delayed S-shaped model [19]	2-Erlang
Modified Duane model [21]	Pareto (second kind)
Truncated-normal model [22]	Truncated-normal
Log-normal model [22], [23]	Log-normal
Inflection S-shaped model [24]	Truncated-logistic
Log-logistic model [25]	Log-logistic
Gompertz model [26]	Gompertz
Goel model [27]	Weibull

- All of the bug detections occur at independent random times, which have an idetical cumulative distribution function (c.d.f.) $F(t)$.

Provided that the number of bugs is $M=m$, the number of detected bugs before time t is given by

$$P(N(t)=n|M=m)=\binom{m}{n}F(t)^n\overline{F}(t)^{m-n}, \quad n=0,1,\ldots,m, \quad (3)$$

where $\overline{F}(t)=1-F(t)$. Since M is a Poisson random variable with mean a, the unconditional p.m.f. of $N(t)$ becomes

$$\begin{aligned} P(N(t)=n) &= \sum_{m=n}^{\infty} e^{-\omega}\frac{a^m}{m!}\binom{m}{n}F(t)^n\overline{F}(t)^{m-n}, \\ &= \frac{(aF(t))^n}{n!}e^{-aF(t)}, \quad n=0,1,\ldots. \end{aligned} \quad (4)$$

Eq. (4) equals the p.m.f. of NHPP-based SRGM given in Eq. (1). Since $\lim_{t\to\infty}F(t)=1$, the mean value function $\Lambda(t)=aF(t)$ converges to a eventually. Also, it should be noted that the mean value function $aF(t)$ is determined by the bug-detection time distribution $F(t)$, i.e., a pattern of the number of detected bugs is dominated by the bug-detection time distribution. Table I is a summary of relationship between SRGMs and bug-detection time distributions.

In practice, we frequently encounter the number of detected bugs during a fixed time interval such as a working day to manage testing progresses. This leads to the model on discrete time

domain. For instance, we consider the following probability on the discritized time domain $0 = t_0 < t_1 < \cdots < t_k < \cdots$:

$$p_k = \frac{F(t_k) - F(t_{k-1})}{\overline{F}(t_{k-1})}, \quad k = 1, 2, \ldots, \tag{5}$$

which is the probability that a bug is detected at the k-th testing period provided that the bug has not been detected until the $k-1$-st testing period. In this paper, p_k is called the bug-detection probability at the k-the testing period. By using p_k, the bug-detection time distribution can be written in the form:

$$F(t_k) = 1 - \prod_{i=1}^{k}(1 - p_i). \tag{6}$$

The above SRGM is called the discrete SRGM in Yamada and Osaki [28].

III. GLM-BASED SRGM

As shown in the previous section, the classical SRGM can represent the number of detected bugs in testing phase. Although this gives us the mathematical tractability of SRGM, it is pointed out that the prediction accuracy and data fitting ability of SRGMs were insufficient from both theoretical and practical points of view. Furthermore, since model parameters of SRGMs can be determined after getting the bug reports in the early phase of testing, the common SRGMs are not used for the prediction of bug detection in the early phase.

On the other hand, software metrics have attractively been studied in software engineering. The software metrics are a set of quantitative measures to identify features of software. Typical metrics are lines of code and software complexity. The values measured in testing phase such as the number of bugs and the number of lines for a bug fix can also be regarded as software metrics. Based on such software metrics, many researchers have tried to explain properties of software system. In many cases, regression techniques are used to analyze the relationship between software metrics and software properties.

During the last four decades after SRGMs were proposed, the relationship between SRGM and regression-based software metrics analysis has not drawn considerable attention. In this paper, we present a modeling framework to integrate classical SRGMs and regression-based software metrics analysis.

The generalized linear model (GLM) is a generalized multiple regression framework to represent the relationship between dependent and independent variables. Also GLM contains logistic and Poisson regression models.

Let $\mathbf{X}_i = (X_{i,1}, \ldots, X_{i,L})$ and Y_i be independent and dependent variables in GLM, where independent and dependent variables are regarded as inputs and the corresponding output respectively. The GLM assumes

- Y_i follows an identical exponential-family distribution with different parameters.
- the mean value of Y is given by $g(\mathrm{E}[Y_i]) = \boldsymbol{\beta}\mathbf{x}_i$, where $\mathbf{x}_i$ is a vector of observations of $\mathbf{X}_i$ and $g(\cdot)$ is called a link function.

Under appropriate exponential-family distributions and link functions, we can reduce the GLM to the existing regression models. For instance, when the exponential-family distribution is the normal distribution and $g(\mu) = \mu$, the corresponding regression becomes an elementary Gaussian multiple regression. When the exponential-family distribution is Poisson distribution, the corresponding GLM is called the Poisson regression. When the exponential-family distribution is Bernoulli distribution, the GLM becomes the binomial regression. In particular, if $g(\mu) = \log(\mu/(1-\mu))$ is a logit function, the GLM corresponds to the logistic regression. The advantage of GLM is to provide an efficient parameter estimation scheme based on the iteratively weighted least squares.

To integrate GLM-based software metrics analysis and SRGMs, we consider the following model assumptions:

- Software testing is executed on discrete time domain; $k = 1, 2, \ldots$ and the detected bugs are fixed by the next testing period.
- Software is divided into several modules. The fault-detection probability of the i-the module at the k-th testing period is given by $p_{i,k}$, which is written by

$$g_p(p_{i,k}) = \boldsymbol{\alpha}_i \boldsymbol{\tau}_{i,k}, \tag{7}$$

  where $\boldsymbol{\tau}_{i,k}$ is a vector of metrics on software test activity for the i-th module at the k-th testing period, and $g_p(\cdot)$ is the link function for fault-detection probability.
- The number of total bugs in the i-th module, M_i, depends on metrics on software design of the i-th module, and the mean number of total bugs is given by

$$g_a(a_i) = \boldsymbol{\beta}\mathbf{x}_i, \tag{8}$$

  where $\mathbf{x}_i$ is a vector of metrics on software design of the i-th module and $g_a(\cdot)$ is the link function for the mean number of total bugs.

In the above, $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}$ are regression coefficients. According to Langberg and Singpurwalla's framework under the above assumptions, we develop GLM-based SRGM. For the i-th software module, the number of detected bugs before the k-th testing period can be described by the following p.m.f.

$$P(N_{i,k} = n) = \frac{\left(a_i\left(1 - \prod_{l=1}^{k}(1-p_{i,l})\right)\right)^n}{n!} \times \exp\left(-a_i\left(1 - \prod_{l=1}^{k}(1-p_{i,l})\right)\right), \tag{9}$$

where

$$g_a(a_i) = \boldsymbol{\beta}_i\mathbf{x}_i, \quad g_p(p_{i,k}) = \boldsymbol{\alpha}_i\boldsymbol{\tau}_{i,k}. \tag{10}$$

The typical link functions are:

$$g_a(\mu) = \mu, \qquad \text{(identity)}, \tag{11}$$
$$g_a(\mu) = \log\mu, \qquad \text{(log)}, \tag{12}$$

and

$$g_p(\mu) = \log\frac{\mu}{1-\mu}, \qquad \text{(logit)}, \tag{13}$$
$$g_p(\mu) = \Phi^{-1}(\mu), \qquad \text{(probit)}, \tag{14}$$
$$g_p(\mu) = \log(-\log(1-\mu)), \qquad \text{(cloglog)}, \tag{15}$$

where $\Phi(x)$ is the c.d.f. of the standard normal distribution and cloglog means a complementary log-log function. Also, typical examples of design metrics which can be handled by GLM-based SRGM are program metrics (statements, lines of code, complexity, etc.) and project metrics (the number of reviews, specifications, etc.). The examples of test activity metrics are the number of testcases, workers, testing days, test coverage and so on.

In the GLM-based SRGM, the software metrics affect both the mean number of total bugs and the bug-detection probability. That is, the model gives the relationship between software metrics and software reliability explicitly. Moreover, from the mathematical point of view, this is a natural extension of the logistic-regression-based SRGM [8] and the Poisson-regression-based SRGM [16], so that the GLM-based SRGM includes the Poisson regression based fault-prone module prediction [29], and Cox regression based SRGM [6], [9], [10] by using different link functions. The GLM-based SRGM is one of the most generalized models in the area of software reliability engineering.

IV. Statistical inference

A. EM algorithm

This section discusses the statistical inference of GLM-based SRGM. One of the challenges in GLM-based SRGM is to provide an efficient parameter estimation algorithm. In the software reliability engineering, the maximum likelihood (ML) estimation is commonly used to estimate parameters from the data. The ML estimation is to find the parameters maximizing the probability that observed data are drawn from the model. The probability is called the likelihood function.

Let $\mathcal{D} = \{n_1, n_2, \ldots, n_K\}$ be the number of detected bugs per working day. Then the probability that $\mathcal{D}$ is observed in the discrete SRGM, i.e., the likelihood function of the discrete SRGM, is given by

$$\mathcal{L}(a, p_1, \ldots, p_K) = \prod_{k=1}^{K} \frac{\left(a p_k \prod_{l=1}^{k-1}(1-p_l)\right)^{n_k}}{n_k!} \times \exp\left(-a\left(1 - \prod_{k=1}^{K}(1-p_l)\right)\right). \tag{16}$$

By finding the parameters which maximize $\mathcal{L}$, we determine the model parameters in the discrete SRGM.

In the case of GLM-based SRGM, since the mean numbers of total bugs for all modules are dominated by the regression coefficients $\boldsymbol{\beta}$, we should represent the likelihood function for all the modules simultaneously. Let $\mathcal{D}_i = \{n_{i,1}, n_{i,2}, \ldots, n_{i,K_i}\}$ be the number of detected bugs per testing period in the i-th module. Then we have

$$\begin{aligned}
&\log \mathcal{L}(\boldsymbol{\beta}, \boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_m) \\
&= \sum_{i=1}^{m}\sum_{k=1}^{K_i} n_{i,k} \log a_i + \sum_{i=1}^{m}\sum_{k=1}^{K_i} \log p_{i,k} \\
&+ \sum_{i=1}^{m}\sum_{k=1}^{K_i}\sum_{l=1}^{k-1} \log(1-p_{i,l}) - \sum_{i=1}^{m}\sum_{k=1}^{K_i} \log n_{i,k}! \\
&- \sum_{i=1}^{m} a_i \left(1 - \prod_{k=1}^{K_i}(1-p_{i,k})\right),
\end{aligned} \tag{17}$$

where

$$g_a(a_i) = \boldsymbol{\beta}_i \mathbf{x}_i, \quad g_p(p_{i,k}) = \boldsymbol{\alpha}_i \boldsymbol{\tau}_{i,k}. \tag{18}$$

It is not easy to maximize the above (log) likelihood function directly. In this paper, we consider an approach in which the parameter estimation of GLM-based SRGM can be reduced to that of the ordinary GLM and the ordinary SRGM.

Concretely, we apply EM (expectation-maximization) algorithm to compute the model parameters of GLM-based SRGM. The EM algorithm is a scheme to obtain ML estimates for the model with unobserved data. Instead of the direct maximization of log-likelihood function, EM algorithm focuses on the maximization of the expected log-likelihood function.

Here we consider the assumption that the numbers of total bugs for all the modules $M_1, \ldots, M_m$ are unobserved. If $M_1, \ldots, M_m$ are observable, the original log-likelihood function can be reduced into the following simple formulas:

$$\log \mathcal{L}_a(\boldsymbol{\beta}) = \sum_{i=1}^{m} M_i \log a_i - a_i, \tag{19}$$

and

$$\begin{aligned}
\log \mathcal{L}_p(\boldsymbol{\alpha}_i) &= \sum_{k=1}^{K_i} n_{i,k} \log p_{i,k} \\
&+ \sum_{k=1}^{K_i}\left(M_i - \sum_{l=1}^{k-1} n_{i,l}\right)\log(1-p_{i,k}).
\end{aligned} \tag{20}$$

In the above formulas, we omit the constant terms because they do not affect the estimates. The above formulas correspond to log-likelihood functions for Poisson regression with link function $g_a(\cdot)$ and binomial regression with link function $g_p(\cdot)$, respectively. However, since $M_1, \ldots, M_m$ are unobserved, the expected values of $M_1, \ldots, M_m$ are applied in the EM algorithm. The expected values of M_i is given by [30]

$$\mathrm{E}[M_i] = \sum_{k=1}^{K_i} n_{i,k} + a_i \prod_{k=1}^{K_i}(1-p_{i,k}). \tag{21}$$

Finally, the one step of EM algorithm for GLM-based SRGM is described in Algorithm 1. In this algorithm, $\mathrm{PR}(\cdot; g_a)$ and $\mathrm{BR}(\cdot; g_p)$ are the functions that provide ML estimates of regression coefficients in Poisson and binomial regression models with link functions g_a and g_p, respectively. They are utilized in the standard statistical analysis package such as R. We repeatedly execute the EM-step until the coefficients converge.

Algorithm 1 EM-step for GLM-based SRGM.

{E-step: computation of expected values}
for all $i \in [1, m]$ **do**
  $a_i \leftarrow g_a^{-1}(\boldsymbol{\beta}\mathbf{x}_i)$
  for all $k \in [1, K_i]$ **do**
    $p_{i,k} \leftarrow g_p^{-1}(\boldsymbol{\alpha}_i\boldsymbol{\tau}_{i,k})$
  end for
  $M_i \leftarrow \sum_{k=1}^{K_i} n_{i,k} + a_i \prod_{k=1}^{K_i}(1 - p_{i,k})$
end for
{M-step: maximization of expected log-lielihood}
$\boldsymbol{\beta} \leftarrow \mathrm{PR}(M_1, \ldots, M_m, \mathbf{x}_1, \ldots, \mathbf{x}_m; g_a)$
for all $i \in [1, m]$ **do**
  $\boldsymbol{\alpha}_i \leftarrow \mathrm{BR}(\mathcal{D}_i, M_i, \boldsymbol{\tau}_{i,1}, \ldots, \boldsymbol{\tau}_{i,K_i}; g_p)$
end for

Algorithm 2 Variable selection for GLM-based SRGM.

for all $i \in [1, m]$ **do**
  Estimate $a_i, \boldsymbol{\alpha}_i$ without software design metrics
  Select variables on test activity
end for
Select variables on software design with $T_1, \ldots, T_m$.

B. Variable selection

The GLM-based SRGM has a large number of parameters than the ordinary SRGM. Thus the GLM-based SRGM often causes the so-called overfitting problem. In general, there are two directions to overcome this problem. One approach is to apply the variable selection. In the context of regression, we can select the variables and their coefficients to enhance the generalization ability of the model. Another approach is to use regularization such as Tikhonov regularization [31].

In this paper, we focus on the variable selection. Particularly, we consider AIC (Akaike information criterion) based variable selection [32]. The AIC is defined as the following formula:

$$\begin{aligned}\mathrm{AIC} = &-2(\text{maximum log-likelihood}) \\ &+ 2(\text{degrees of freedom}). \end{aligned} \tag{22}$$

In most models, the degrees of freedom becomes the number of parameters. The AIC gives a penalty for the degrees of freedom of the model. In the AIC-based variable selection, we find the combination of variables that minimizes the AIC. Since it is difficult to check all the combinations in terms of time complexity, stepwise approach is used to determine the good combination. However, GLM-based SRGM has two kinds of coefficients for metrics on test activity and software design. Thus we propose a heuristic approach to find the good combination of variables.

Algorithm 2 is our model selection approach in this paper. In the first part, we select the variables on test activity for each module independently. In this step, a_i is regarded as a parameter which does not depend on design metrics. In the second part, we select the variables on software design under the variables that are determined in the first part.

V. Experiments

In this section, we examine the data fitting ability of GLM-based SRGM by comparing the ordinary SRGM with our new models; the logistic-regression-based SRGM with test activity metrics and the Poisson-regression-based SRGM with design metrics. In the experiments, we compute AICs for all the models and can investigate the model representation; how well the model fits to the true model.

TABLE II. Software modules in Tomcat 7.

Module	Description
Manager	Web application for administration
Connectors	Interaction between Tomcat and Apache
Catalina	Servlet container core
Jasper	JSP page compiler and runtime engine
Servlets	Servlet programs

TABLE III. Software modules in Lenya.

Module	Description
Access Control (AC)	Access control for sites
Form Editor (FE)	WYSIWYG editor
Kupu Integration (KI)	Interaction to WYSIWYG XHTML editor (Kupu)
Lucene Integration (LI)	Interaction to search engine (Lucene)
Navigation Framework (NF)	Automatic generator for navigation items
Site Management (SM)	Lenya core
TinyMCE Integration (TI)	Interaction to TinyMCE editor
Usecase Framework (UF)	A framework for the usecase with JX template and Java
Workflow (WF)	Workflow engine with XML

We use the data collected from the open source projects; Tomcat and Lenya in ASF (Apache Software Foundation). Apache Tomcat provides Java Servlets and their components are written by Java and C languages. In ASF Bugzilla, Tomcat has 5 modules shown in Table II. We focus on Tomcat 7 and collected the bug reports from June 2010 to November 2013. Apache Lenya is Java/XML based contents management system. Similar to Tomcat, Lenya has the software modules in Table III. We collect the bug reports from May 2003 to January 2013.

In our experiments, we need to measure software metrics on both software design and test activity. The design metrics were measured directly from source code with a tool; Source Monitor[1]. Using Source Monitor, we measure the software design metrics in Table IV. Tables V and VI present the design metrics for Tomcat 7 and Lenya, respectively.

On the other hand, the metrics on test activity were obtained from Bugzilla. Table VII shows the metrics collected in our experiments. Both the number of bugs and the metrics on test activity are collected for every month, i.e., one test period is defined as a month. Tables VIII and IX present summaries of the number of bugs and the metrics on test activity collected in Tomcat 7 and Lenya, respectively. The columns indicate the number of bugs, the number of comments and the number of votes counted in the corresponding months from the beginning of testing period. The upper and lower parts of these tables show the data at the points when 75% of total months and 100% of total months. From these tables, Tomcat 7 project has still been active for finding bugs, but Lenya project has already ended in terms of bug detection.

For these data, we apply the ordinary SRGM, logistic-regression-based SRGM [8] and Poisson-regression-based SRGM [16]. The ordinary SRGM is based on only the number

[1] http://www.campwoodsw.com/sourcemonitor.html

TABLE V. DESIGN METRICS FOR TOMCAT 7.

Module	Fl	Ln	St	Br	Ca	Cm	Cl	Me/Cl	St/Me	MCx	MDp	ADp	ACx
Manager	17	7350	3091	20.9	2259	20.1	14	13.36	13.78	35	9+	2.54	5.12
Connectors	22	12951	4539	20.8	2540	23.1	38	17.37	5.13	296	9+	2.19	3.12
Catalina	619	188585	68583	19.5	37977	30.2	784	11.27	5.65	296	9+	2.25	2.93
Jasper	123	47715	21410	19.8	12266	26.7	237	9.48	7.02	100	9+	2.61	3.29
Servlets	141	23407	3305	8.1	1211	50.2	115	7.1	1.63	25	9+	1.45	1.59

TABLE VI. DESIGN METRICS FOR LENYA.

Module	Fl	Ln	St	Br	Ca	Cm	Cl	Me/Cl	St/Me	MCx	MDp	ADp	ACx
AC	63	8970	3631	11.9	2153	34.5	57	6.88	5.88	24	7	1.77	2.18
FE	66	9271	4302	20.5	2010	17.2	18	31.83	6.63	103	9+	2.57	4.37
KI	357	20964	9592	13.2	4686	12.7	9	78.11	12.48	35	9+	1.97	3.22
LI	67	6419	2181	15.3	1026	31.3	38	6.26	5.47	25	8	1.98	2.63
NF	15	799	369	10.8	205	26.5	9	4.11	5.3	12	8	1.61	2.45
SM	87	6045	1936	12.7	1379	19.4	33	5.15	7.19	12	7	1.83	2.61
TI	26	986	302	13.6	136	26.3	2	21	5.19	18	6	1.63	3.04
UF	52	5847	2358	13.9	1187	34.6	33	9.18	4.99	21	8	1.88	2.23
WF	44	4252	1637	12.2	941	30.2	31	4.94	6.51	19	8	1.82	2.36

TABLE IV. SOFTWARE DESIGN METRICS MEASURED BY SOURCE MONITOR.

Metric	Description
Files (Fl)	The number of files
Lines (Ln)	Lines of code
Statements (St)	The number of statements
%Branches (Br)	Ratio of branches
Calls (Ca)	The number of calls for methods
%Comments (Cm)	Ratio of comments
Classes (Cl)	The number of classes
Methods/Class (Me/Cl)	The number of methods per class
AvgStmts/Method (St/Me)	The average number of statements per method
MaxComplexity (MCx)	The maximum of MaCabe complexity
MaxDepth (MDp)	The maximum of nests
AvgDepth (ADp)	The average of nests
AvgComplexity (ACx)	The average of MaCabe complexity

TABLE VII. TEST ACTIVITY METRICS.

Metric	Description
Time (T)	Days for a month
Cumulative time (CT)	Cumulative days
Comments (CM)	The number of messages subscribed to discussion pages
Votes (VT)	The number of votes

TABLE VIII. SUMMARY OF BUG DATA AND TEST ACTIVITY (TOMCAT 7).

Module	months	bugs	comments	votes
Manager	32	29	128	1
Connectors	32	74	438	69
Catalina	32	462	2156	48
Jasper	32	87	419	4
Servlets	32	46	259	6
Manager	42	40	180	1
Connectors	42	98	570	70
Catalina	42	542	2511	51
Jasper	42	101	488	4
Servlets	42	62	340	6

TABLE IX. SUMMARY OF BUG DATA AND TEST ACTIVITY (LENYA).

Module	months	bugs	comments	votes
AC	87	77	380	0
FE	87	51	197	0
KI	87	46	251	0
LI	87	31	118	0
NF	87	21	81	0
SM	87	118	535	0
TI	87	23	102	0
UF	87	19	91	0
WF	87	42	165	0
AC	117	77	380	0
FE	117	52	198	0
KI	117	46	251	0
LI	117	31	118	0
NF	117	21	81	0
SM	117	121	545	0
TI	117	23	102	0
UF	117	19	91	0
WF	117	44	169	0

of detected bugs, and does not use any software metrics. In addition, the ordinary SRGM handles the data of one software module. The logistic-regression-based SRGM [8] focuses only on the metrics on test activity, and it is the same as the GLM-based SRGM for only one module in which the link function g_p is the logit function. Thus, similar to the ordinary SRGM, the logistic-regression-based SRGM can handle the data of only one software module. The Poisson-regression-based SRGM [16] is also a subclass of GLM-based SRGM which does not use test activity metrics. The coefficients of design metrics in Poisson-regression-based SRGM are estimated from the data of all the modules.

Tables X and XI present the AIC for the ordinary SRGM, logistic-regression-based SRGM, Poisson-regression-based SRGM and GLM-based SRGM for 100% testing periods in Tomcat 7 and Lenya, where we apply 'log' and 'logit' for the link functions to design and test activity metrics, respectively. Furthermore, to estimate the parameters of GLM-based SRGM, we use the statistical algorithm described in Section 3. Tables XII and XIII show the selected variables for Tomcat 7 and Lenya, respectively. Since we use the heuristic approach to determine the variables, the selected variables on test activity of logistic-regression-based SRGM and GLM-based SRGM are exactly same. But the selected design metrics are different between Poisson-regression-based SRGM and GLM-based SRGM.

Looking at the AIC results for all the modules, we find that GLM-based SRGM is the best among them in terms of data fitting ability. Also, it can be seen that AIC when we consider the metrics on test activity becomes much smaller than that in the case of the metrics on software design. However, in general, if AIC of one model is less than another model by 2, there is a significant difference between them. That is, although

TABLE X. AIC RESULTS FOR TOMCAT 7 (42 MONTHS).

Module	SRGM	Logistic	Poisson	GLM
Manager	117.07	90.49	—	—
Connectors	168.82	129.44	—	—
Catalina	247.91	210.87	—	—
Jasper	167.34	137.57	—	—
Servlets	148.33	110.57	—	—
All modules	849.47	678.94	847.40	675.86

TABLE XI. AIC RESULTS FOR LENYA (117 MONTHS).

Module	SRGM	Logistic	Poisson	GLM
AC	218.76	179.56	—	—
FE	196.62	137.79	—	—
KI	152.79	121.29	—	—
LI	142.98	95.48	—	—
NF	103.06	84.49	—	—
SM	272.55	233.99	—	—
TI	100.06	85.92	—	—
UF	88.06	85.67	—	—
WF	167.69	146.29	—	—
All modules	1442.56	1170.48	1440.44	1168.48

TABLE XII. SELECTED VARIABLES IN TOMCAT 7 (42 MONTHS).

Module	Test	Design	
		Poisson	GLM
Catalina	CM, VT	Fl, Ca, Cm	St, Cm
Connectors	CM, VT		
Jasper	CM		
Manager	CM, VT		
Servlets	CT, CM, VT		

TABLE XIII. SELECTED VARIABLES IN LENYA (117 MONTHS).

Module	Test	Design	
		Poisson	GLM
AC	CT, CM	Ln, St, Br, Ca, St/Me, MCx	Fl, St, Br, Ca, Cm, Cl, MCx
FE	CM		
KI	CT, CM		
LI	CT, CM		
NF	CT, CM		
SM	CT, CM		
TI	CT, CM		
UF	T, CT, CM		
WF	CM		

the effectiveness of design metrics is smaller than test activity metrics, it is sufficiently meaningful to consider the design metrics in SRGM. Also, in Tables XII and XIII, we find that comments (CM) are always selected as a variable among the metrics on test activity. In the past literature, the ordinary SRGM handles only the time factor, i.e., time (T) or cumulative time (CT). The result in Table XII is an evidence that using only the time factor is insufficient to explain the bug-detection process and that the factor quantifying the active of testing such as the number of comments can more directly explain the bug-detection process. On the other hand, in the result of Lenya, the cumulative time (CT) is selected. This result indicates that the bug-detection probability depends on testing time. However, by considering the result of Tomcat 7, other factors should be measured to explain bug-detection probability directly, since we focus on only comments and votes in this experiment.

On the design metrics, the volume metrics such as Fl, Ln and St are selected as variables. This is not a surprising result, because the module with many codes potentially includes a number of bugs. However, the measures on complexity are only selected in Lenya. Compared to Tomcat 7, the bug-detection process of Lenya project converges. In such a case, there remain the bugs that are difficult to find and they are generally introduced in the program with high complexity. Thus the complexity measures are selected to explain the number of total bugs in Lenya.

Based on GLM-based SRGM, we provide the reliability measures. In the scheme of GLM-based SRGM, we cannot predict future behavior of bug-detection process without future test activity metrics. Instead of evaluating the future behavior, we consider the residual faults at the current testing period. That is, when obtaining model parameters at the current testing period, we derive the probability that the current software module does not have any bug. In the paper, this probability is called the fault (bug) free probability (FFP). The FFP at the k-th testing period for the i-th module is given by

$$\text{FFP} = \exp\left(-\hat{a}_i \prod_{k=1}^{K_i}(1-\hat{p}_{i,k})\right), \tag{23}$$

where $\hat{a}_i$ and $\hat{p}_{i,k}$ are estimated parameters by using the data before the k-th testing period. Notice that, since FFP is defined by $\hat{a}_i$ and $\hat{p}_{i,k}$, FFP can also be computed in the ordinary SRGM, logistic-regression-based SRGM and Poisson-regression-based SRGM.

Tables XIV and XV present FFPs of Tomcat 7 and Lenya modules at 75% of total testing periods. In Table XIV, all the models provide the low FFPs for Manager, Connectors, Catalina and Jasper. The FFP of Servlets is small in the logistic-regression-based SRGM and the GLM-based SRGM. In fact, looking at Table VIII, we find that a number of bugs have been detected in all the modules from 75% to 100% of testing periods. FFP has been improved by using the metrics on test activity. On the other hand, in Table XV, FFPs of FE, LI and UF are less than 0.5 in the ordinary SRGM and the Poisson-regression-based SRGM, and FFPs of FE and WF are less than 0.5 in the logistic-regression-based SRGM and GLM-based SRGM. From Table IX, we find that new bugs have been detected in FE, SM and WF. Although FFP of SM are high in all the models, FFPs of LI, UF and WF are also improved by considering the metrics on test activity.

VI. CONCLUSIONS

In this paper, we have proposed a generalized modeling framework of software reliability growth model (SRGM) to handle software metrics of open source projects. The presented framework utilizes the generalized linear model (GLM) and is a natural extension of the existing SRGMs. The advantage of this framework is to provide an efficient algorithm for estimating model parameters. The idea behind our algorithm is to use ready-made estimation modules for Poisson regression and binomial regression (logistic regression) by applying the EM algorithm. The proposed algorithm is quite simple so that it can easily be implemented.In the experiments, we have investigated the data fitting ability of GLM-based SRGM by comparing the other models in terms of AIC. As a result, it can be seen that the GLM-based SRGM possessed the highest data fitting ability among the ordinary SRGM, logistic-regression-based SRGM and Poisson-regression-based SRGM. Also through the bug and metrics data for open source projects, we have verified that the metrics on both software design

TABLE XIV. FFP FOR TOMCAT 7 (32 MONTHS).

Module	SRGM	Logistic	Poisson	GLM
Manager	0.001	0.000	0.001	0.000
Connectors	0.000	0.000	0.000	0.000
Catalina	0.000	0.000	0.000	0.000
Jasper	0.000	0.000	0.000	0.000
Servlets	0.870	0.153	0.870	0.153

TABLE XV. FFP FOR LENYA (87 MONTHS).

Module	SRGM	Logistic	Poisson	GLM
AC	0.991	0.789	0.991	0.791
FE	0.000	0.000	0.000	0.000
KI	0.476	0.989	0.476	0.989
LI	0.000	0.803	0.000	0.809
NF	0.525	0.923	0.524	0.927
SM	0.943	0.994	0.943	0.994
TI	0.985	0.857	0.985	0.856
UF	0.396	0.998	0.390	0.998
WF	0.999	0.000	0.999	0.000

and test activity affected the software reliability assessment. In particular, the software reliability strongly depends on the metrics on test activity and the software reliability measure can be improved by considering the test activity metrics in the experiments. This is because the number of modules is relatively small. For instance, Tomcat 7 project has only 5 modules and Lenya project has only 9 modules. As the number of modules increases, the effect of design metrics may become clear.

The GLM-based SRGM presented in this paper is an excellent framework to handle software metrics in the software reliability growth model from the viewpoint of mathematics, since the GLM-based SRGM includes almost all the existing models and can provide the effective parameter estimation algorithm. Moreover, there is a possibility for GLM-based SRGM to be applied to analyze agile and incremental development (see [33]).

However, there are several problems on data analysis. That is, there still remains the problem what metrics will affect the software reliability best. For example, in this paper, we have used program metrics as design metrics. However, in our modeling framework, it is assumed that design metrics are related to the number of bugs at the beginning of testing. However, the metrics measured in previous development phase, such as the number of specifications and the number of reviews, might be more suitable. Unfortunately, since we could not obtain such metrics data, we just used the program metrics in this paper. Also, on the test activity metrics, it is known that test coverage is quite effective in [6], [8]. In future, we will try to perform the data analysis by using GLM-based SRGM with the other types of metrics and will study what kinds of metrics affect the software reliability.

ACKNOWLEDGMENTS

This research is a part of "Research Initiative on Advanced Software Engineering in 2013" supported by SEC (Software Reliability Enhancement Center), IPA (Information technology Promotion Agency Japan).

REFERENCES

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

[2] M. R. Lyu, Ed., *Handbook of Software Reliability Engineering*. New York: McGraw-Hill, 1996.

[3] J. D. Musa, *Software Reliability Engineering*. New York: McGraw-Hill, 1999.

[4] H. Pham, *Software Reliability*. Singapore: Springer, 2000.

[5] S. Yamada, H. Ohtera, and H. Narihisa, "Software reliability growth with testing-effort," *IEEE Transactions on Reliability*, vol. R-35, no. 1, pp. 19–23, 1986.

[6] K. Shibata, K. Rinsaka, and T. Dohi, "Metrics-based software reliability models using non-homogeneous Poisson processes," in *Proceedings of 17th International Symposium on Software Reliability Engineering*. IEEE CS Press, 2006, pp. 52–61.

[7] D. R. Cox, "Regression models and life-tables," *Journal of the Royal Statistical Society, Series B*, vol. 34, no. 2, pp. 187–220, 1972.

[8] H. Okamura, Y. Etani, and T. Dohi, "A multi-factor software reliability model based on logistic regression," in *Proceedings of 21st International Symposium on Software Reliability Engineering (ISSRE2010)*. Los Alamitos: IEEE Computer Society Press, 11 2010, pp. 31–40.

[9] D. Kuwa and T. Dohi, "Generalized logit-based software reliability modeling with metrics data," in *Proceedings of The 37th Annual International Computer Software and Applications Conference (COMPSAC 2013)*. IEEE CPS, 2013, pp. 246–255.

[10] ——, "Generalized Cox proportional hazards regression-based software reliability modeling with metrics data," in *Proceedings of The 19th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2013)*. IEEE CPS, 2013, pp. 328–337.

[11] T. M. Khoshgoftaar and J. C. Munson, "Predicting software development errors using software complexity metrics," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 2, pp. 253–261, 1990.

[12] T. M. Khoshgoftaar, J. C. Munson, B. B. Bhattacharyya, and G. D. Richardson, "Predictive modeling techniques of software quality from software measures," *IEEE Transactions on Software Engineering*, vol. 18, no. 11, pp. 979–987, 1992.

[13] T. M. Khoshgoftaar, B. B. Bhattacharyya, and G. D. Richardson, "Predicting software errors, during development, using nonlinear regression models: a comparative study," *IEEE Transactions on Reliability*, vol. 41, no. 3, pp. 390–395, 1992.

[14] W. M. Evanco and R. Locovara, "A model-based framework for the integration of software metrics," *Journal of Systems and Software*, vol. 26, pp. 77–86, 1994.

[15] W. M. Evanco, "Poisson analyses of defects for small software components," *Journal of Systems and Software*, vol. 38, pp. 27–35, 1997.

[16] H. Okamura and T. Dohi, "A novel framework of software reliability evaluation with software reliability growth models and software metrics," in *Proceedings of The 15th IEEE International Symposium on High Assurance Systems Engineering (HASE 2014)*. IEEE CPS, 2014, pp. 97–104.

[17] A. L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE Transactions on Reliability*, vol. R-28, pp. 206–211, 1979.

[18] S. Yamada and S. Osaki, "Cost-reliability optimal release policies for software systems," *IEEE Transactions on Reliability*, vol. R-34, pp. 422–424, 1985.

[19] S. Yamada, M. Ohba, and S. Osaki, "S-shaped reliability growth modeling for software error detection," *IEEE Transactions on Reliability*, vol. R-32, pp. 475–478, 1983.

[20] N. Langberg and N. D. Singpurwalla, "Unification of some software reliability models," *SIAM Journal on Scientific Computing*, vol. 6, no. 3, pp. 781–790, 1985.

[21] B. Littlewood, "Rationale for a modified duane model," *IEEE Transactions on Reliability*, vol. R-33, no. 2, pp. 157–159, 1984.

[22] H. Okamura, T. Dohi, and S. Osaki, "Software reliability growth models with normal failure time distributions," *Reliability Engineering and System Safety*, vol. 116, pp. 135–141, 2013.

[23] J. A. Achcar, D. K. Dey, and M. Niverthi, "A Bayesian approach using nonhomogeneous Poisson processes for software reliability models," in *Frontiers in Reliability*, A. P. Basu, K. S. Basu, and S. Mukhopadhyay, Eds. Singapore: World Scientific, 1998, pp. 1–18.

[24] M. Ohba, "Inflection S-shaped software reliability growth model," in *Stochastic Models in Reliability Theory*, S. Osaki and Y. Hatoyama, Eds. Berlin: Springer-Verlag, 1984, pp. 144–165.

[25] S. S. Gokhale and K. S. Trivedi, "Log-logistic software reliability growth model," in *Proc. 3rd IEEE Int'l. High-Assurance Systems Eng. Symp. (HASE-1998)*. IEEE CS Press, 1998, pp. 34–41.

[26] K. Ohishi, H. Okamura, and T. Dohi, "Gompertz software reliability model: estimation algorithm and empirical validation," *Journal of Systems and Software*, vol. 82, no. 3, pp. 535–543, 3 2009.

[27] A. L. Goel, "Software reliability models: Assumptions, limitations and applicability," *IEEE Transactions on Software Engineering*, vol. SE-11, pp. 1411–1423, 1985.

[28] S. Yamada and S. Osaki, "Discrete software reliability growth models," *Journal of Applied Stochastic Models and Data Analysis*, vol. 1, no. 1, pp. 65–77, 1985.

[29] T. M. Khoshgoftaar, K. Gao, and R. M. Szabo, "An application of zero-inflated Poisson regression for software fault prediction," in *Proceedings of 12th International Symposium on Software Reliability Engineering*. IEEE CS Press, 2001, pp. 66–73.

[30] H. Okamura, A. Murayama, and T. Dohi, "EM algorithm for discrete software reliability models: a unified parameter estimation method," in *Proc. 8th IEEE Int. Symp. High Assurance Systems Eng.*, 2004, pp. 219–228.

[31] A. N. Tikhonov, A. S. Leonov, and A. G. Yagola, *Nonlinear Ill-Posed Problems*. Chapman and Hall, 1998.

[32] H. Akaike, "Information theory and an extension of the maximum likelihood principle," in *Proc. 2nd Int. Symp. Inform. Theory*, B. N. Petrov and F. Csaki, Eds., 1973, pp. 267–281.

[33] T. Fujii, T. Dohi, and T. Fujiwara, "Towards quantiative software reliability assessment in incremental development processes," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE 2011)*. New York: ACM, 2011, pp. 41–50.