

# Modeling the Effect of Size on Defect Proneness for Open-Source Software

A. Güneş Koru

Department of Information Systems, UMBC  
Baltimore, MD, USA  
gkoru@umbc.edu

Dongsong Zhang

Department of Information Systems, UMBC  
Baltimore, MD, USA  
zhangd@umbc.edu

Hongfang Liu

Department of Biostatistics, Bioinformatics, and Biomathematics  
Georgetown Medical Center  
Washington D.C., USA  
hl224@georgetown.edu

## Abstract

*Quality is becoming increasingly important with the continuous adoption of open-source software. Previous research has found that there is generally a positive relationship between module size and defect proneness. Therefore, in open-source software development, it is important to monitor module size and understand its impact on defect proneness. However, traditional approaches to quality modeling, which measure specific system snapshots and obtain future defect counts, are not well suited because open-source modules usually evolve and their size changes over time. In this study, we used Cox proportional hazards modeling with recurrent events to study the effect of class size on defect-proneness in the Mozilla product. We found that the effect of size was significant, and we quantified this effect on defect proneness.*

## 1 Introduction

A broad range of general-purpose and domain-specific open-source software (OSS) products have become an alternative to closed-source proprietary software for many organizations. Therefore, the quality of OSS is becoming increasingly important for adoption decisions. Especially, as OSS software spreads from general-purpose applications to some domain-specific applications, such as, health-care applications [13, 28] quality requirements increase because software failures can result in more serious consequences.

Without empirical evidence, some OSS pioneers and proponents have often claimed that OSS is of high quality [31] because public source code is believed to be peer-reviewed extensively resulting in effective removal of defects. However, recent findings suggest that these arguments are questionable. In a survey given to OSS developers, Stark [37] found that 50% of code was either not or little peer-reviewed. More than 55% of the respondents agreed that peer-reviews in OSS projects were less organized compared to those used in closed-source settings. McDonnell [24] expressed concerns about the effectiveness of peer-reviews that took place in OSS projects, stating that some code snippets were peer-reviewed redundantly or skipped entirely in non-systematic reviews.

Currently, OSS projects, in general, do not employ systematic approaches to guide the peer-review and testing activities. In their survey, Zhao and Elbaum [42] noted that almost 30% of OS projects estimated their code coverage for testing to be less than 30%. Less than 5% of the respondents used a testing tool to assess coverage accurately. Koru et al. [21] performed a survey of the quality assurance activities in biomedical OSS projects and concluded that the level of peer-review and testing in those projects was not satisfactory and sufficient to put them in use for mission-critical purposes. In this context, there is a strong need to understand the characteristics of defect-prone OSS modules so that *focused* preventive actions can be taken, and quality assurance efforts can be prioritized with maximum efficiency and effectiveness and minimum cost. There is a large body of work on

characterization and prediction of defect-prone modules in closed-source settings [40]. There is a general agreement that software size is positively associated with defect-proneness [9]. Indeed, El Emam reported that, when controlled by class size, none of the object-oriented measures they studied was associated with defect-proneness anymore [10]. Knowing the significance of size from the studies conducted on closed-source software, it is important to build statistical models to increase our understanding of the relationship between the size and defect proneness of OSS.

On the other hand, OSS projects usually have an important characteristic, *evolutionary product development* [12]. OSS modules usually change in size and in other structural measures during field operation when their defects are identified. For example, changes in the average structural measures of Mozilla was reported by Gyimothy et al. [15] during its successive releases. During OSS evolution, corrective changes and functional enhancements take place concurrently and continuously. In traditional development settings, developers often perform some requirements analysis and design activities before implementation. The resulting product often becomes a planned, designed, and tested product at the time of release. Therefore, when developing statistical models for traditional development settings, one can assume that module size does not change significantly after a certain release, and therefore can use the size measurements of that release as inputs to various statistical models. Usually, the post-release defect count is used as the response (or dependent) variable. However, since OSS often evolves, the size measurements of OSS modules often change over time (sometimes dramatically). In addition, some OSS modules might be removed from a system shortly after the measurement of a release. As a result, their defect counts can be smaller. New modules can be added at any time. Moreover, at the measurement time, OSS modules in a release can have various levels of maturity in terms of quality. Because of all these reasons, traditional static modeling approaches are not well suited to model the relationship between size and defect for OSS products.

The objective of this study is to model the relationship between size and defect proneness while addressing the unique dynamic characteristics of OSS projects. We adopted Cox proportional hazards modeling with recurrent events [38]. Defect fixes made to C++ classes in the open-source Mozilla project were modeled as recurrent events, and time to event was used as an indicator of defect-proneness. The classes that received more frequent defect fixes were considered more defect prone. Note that it is important to perform this

analysis at the class level, not at the file level, because classes represent the logical structure and decomposition of an object-oriented system like Mozilla. Here, defect-proneness means the instantaneous hazard of receiving a defect fix for a C++ class. Class size was measured in LOC (Lines of Code) excluding the blank and comment lines. During a long observation period, the C++ classes introduced to Mozilla were followed up, and their entire size and event histories were obtained from the CVS (Concurrent Versions System) repository of the project.

In the rest of the paper, we start by summarizing the related work in this area. Then, we discuss Cox proportional hazards modeling for recurrent events in Section 3. After that, we describe the data used in the study and present our modeling and results in Sections 4 and 5, respectively. Finally, we conclude the paper and present the future research directions in Section 6.

## 2 Related Work

In this section, we discuss the related work on the relationship between software size and defects. As mentioned earlier, traditionally, many studies in the field of software quality have examined this relationship by measuring certain system snapshots and obtaining the future defect count. Some studies did not examine this relationship and used a derived measure, *defect density*, to make quality comparisons which inherently assumed a linear relationship between size and defects.

A number of studies [17, 18, 41] used defect density reported an optimal size for software modules that could maximize quality, an approach known as *Goldilock's conjecture*. When defect density was plotted against size, a U-shape curve was obtained. Therefore, developers were advised to produce modules of intermediate size, *not too small or too big*. El Emam et al. [9] conducted an extensive survey of these studies and reported that there were also studies that validated only left half of the U-shape curve [4, 27, 34] or only its right half [5]. They stated that Goldilock's conjecture was an arithmetic artifact that stemmed from plotting a variable against its inverse.

Fenton and Ohlsson observed no relationship between defect density and size in their study of a system at Ericsson [11]. They stated that the only *general* conclusion that can be drawn from the previous studies in the field was that defect count would *generally* increase with size. A linear relationship between size and defect count is not always guaranteed. In fact, such a linear relationship would have resulted in a flat line in Hatton's studies [17, 18] when defect density was plotted against size. Size was among the predictor

variables in a large number of studies that developed regression models (e.g., [16]) or various machine learning models [25, 22] to predict defect-prone modules.

In the OSS context, the relationship between size and defects was not studied systematically. A number of studies made quality comparisons among OSS products by using defect density, which assumed an inherent linear relationship between size and defects. Mockus et al. [26] used post-release defect density to compare the quality of Apache with four of its closed-source counterparts. They found that Apache had inferior quality compared to all four closed-source products. Dinh-Trong and Bieman [8] repeated this study for a different OSS product, FreeBSD, with the same measures. They found that the post-release defect density for FreeBSD was lower than those of Apache and four commercial products examined by Mockus et al [26]. Sherriff [35] et al. found that defect density was related to the four development process metrics in Glaskow Haskell Compiler project.

In summary, the results of the studies about the relationship between size and defects are not conclusive, although it is argued that there is generally a positive relationship. Making assumptions about the nature of this relationship is problematic not only because the research results are inconclusive, but also because OSS has a unique dynamic characteristic, which calls for different modeling approaches. Different from the above studies, in this research, the instantaneous hazard of receiving a defect fix, rather than post-release defect count, was considered defect-proneness. In addition, the entire history of size measurements was used for C++ classes rather than looking at the measurements at a single snapshot.

### 3 Cox Proportional Hazards Model for Recurrent Events

Recurrent events, also called repeated measurements, multiple events, or recurring events, refer to the events that recur at intervals. For example, a vehicle can break, get repaired, and break again for multiple times; alcoholics can be treated but they may go back to their old habits more than once. At any event moment, all of the subjects that are under follow up constitute a risk set. Cox proportional hazards model [6] (henceforth *Cox model*) is one of the most commonly used techniques to create time-to-event models [7, 38]. It is associated with the counting process and Martingale theory [1, 2], which makes it suitable for the analysis of recurrent events. Cox model has been used in the OSS research. For example, Singh and Tan [36] used a conditional hazard function approach to study

the time that an OSS team might take to develop the first stable release. Therefore, we chose to use Cox model to predict the effect of size on defect proneness.

We use the notation of the Cox model presented by Therneau and Grambsch [38] in the rest of the paper. In this study, each defect fix made to a class is considered an event. There is a single time-dependent covariate, size, denoted by  $x(t)$ . We specify the hazard function, which is the instantaneous risk of an event for class  $i$  at time  $t$ , as:

$$\lambda_i(t) = \lambda_0(t)e^{\beta x_i(t)}. \quad (1)$$

$\beta$  is the regression coefficient for  $x_i(t)$  and  $\lambda_0(t)$  is an *unspecified* non-negative function of time called the *baseline hazard function*. It is the instantaneous hazard of having an event without any covariate effect (i.e. when  $\beta = 0$ ).

Cox model is called semi-parametric because baseline hazard function is not described. As a result, time to event values are used in a comparative sense. Cox model is proportional because the hazard ratio for two subjects would only depend on the differences in their covariate values. If one writes the right side of the Equation 1 for two subjects, say classes  $j$  and  $k$ , and takes their ratio, the result should be  $e^{\beta(x_j(t) - x_k(t))}$ , which is the instantaneous relative risk. Note that  $\beta$  is a constant over time. This is an important assumption of any Cox model, known as proportional hazards assumption, which needs to be checked by model developers. If this assumption is violated,  $\beta$  will be a time-varying *coefficient*,  $\beta(t)$ , and the hazard functions for two classes will not be proportional. We check this assumption in our models as discussed in Section 5.

To deal with recurrent events, we define two functions:

$$N_i(t) = \left\{ \begin{array}{l} \text{The number of observed events for} \\ \text{class } i \text{ in a time period of } (0, t] \end{array} \right\} \quad (2)$$

$$Y_i(t) = \left\{ \begin{array}{l} 1, \text{ if class } i \text{ is under observation} \\ \quad \text{and at risk at time } t \\ 0, \text{ otherwise} \end{array} \right\} \quad (3)$$

$N(t)$  is an unpredictable stochastic process called counting process.  $N_i(t)$  can be greater than one.  $Y(t)$  is a predictable process, which means that its value at time  $t$  is known infinitesimally before  $t$ , denoted by  $(t-)$ .  $N(t)$  can only be known infinitesimally after  $t$  (denoted by  $t+$ ). We also define two corresponding processes aggregated over all classes:

$$\bar{Y}(t) = \sum_i Y_i(t). \quad (4)$$

$$\bar{N}(t) = \sum_i N_i(t). \quad (5)$$

$\bar{Y}(t)$  is the total number of classes at risk at time  $t$ , and  $\bar{N}(t)$  is the total number of events until and including time  $t$ . When we focus on *expected* number of events at  $t$ , we are interested in the *estimated cumulative hazard* denoted by  $\hat{\Lambda}(t)$ , which is also easier to estimate compared to the hazard function [38]. For a very small time interval  $(s, s + h]$ , the hazard in this interval can be estimated by the ratio of the number of events occurred in that interval over the number of classes at risk at  $s$ , which is  $[\bar{N}(s + h) - \bar{N}(s)]/\bar{Y}(s)$ . Summing such hazards during  $(0, t]$ , we obtain the estimated number of events:

$$\hat{\Lambda}(t) = \int_0^t \frac{d\bar{N}(s)}{\bar{Y}(s)}. \quad (6)$$

$d\bar{N}$  is a simpler way of writing the discrete and continuous part of the process together. It can be written as  $d\bar{N}(t) = \Delta\bar{N}(t) + n(t)dt$ .  $\Delta\bar{N}(t)$  is the number of events occurring exactly at time  $t$ , which is  $\bar{N}(t) - \bar{N}(t-)$ .  $n(t)dt$  is the change in the continuous portion.

It can be noted that while developing the Equation 6, we did not consider any covariate. When there is a covariate  $x$  with the coefficient estimate  $\hat{\beta}$ , the Nelson-Aalen estimate of the *baseline* cumulative hazard will be an extension of Formula 6:

$$\hat{\Lambda}_0(t, \hat{\beta}) = \int_0^t \frac{d\bar{N}(s)}{\sum_i Y_i(s) \hat{r}_i(s)}, \quad (7)$$

where  $\hat{r}_i$  is called the *risk score* equal to  $e^{x_i(t)\hat{\beta}}$ . Following the Equation 1, the estimated cumulative hazard for class  $i$  can be written as:

$$\hat{\Lambda}(t, \hat{\beta}) = \hat{\Lambda}_0(t, \hat{\beta}) \hat{r}_i(t). \quad (8)$$

In the above discussion, we skipped the lengthy explanation of the estimation of  $\hat{\beta}$ , which can be found in [6]. For computation, we used the survival package [29] of Therneau and the Hmisc [20] and Design [19] packages of Harrell under the statistical environment, R [30].

Finally, it is worth to mention a couple of useful points:

1.  $R^2$  adjusted values calculated for Cox models are not appropriate measures of model adequacy shown by a detailed study of Schemper and Star [32] because of the censored data (data from subjects without events). Hosmer and Lemeshow argues that a perfectly adequate model might have,

at face value, a low  $R^2$  adjusted value because of the censored observations [7]. The second problem with  $R^2$  adjusted values is that the residuals in Cox models are not normally distributed. In this paper, we still show the  $R^2$  adjusted value but use other approaches to check model adequacy (see Section 5).

2. The terms of a Cox model are computed at unique event times [38]. Since at any given event time, at most one observation of a class is used in the internal computations (e.g. in Equation 7), the intra-subject correlations do not constitute a problem in the calculation of the Cox model terms. More importantly, the size values of the classes that are in the risk set at any given event time should not be correlated (inter-subject correlation). We have no reason to believe that there are such correlations. The intra-subject correlations should be taken into account while examining the functional form and estimating the variance of  $\hat{\beta}$ . Therefore, when studying the functional form, we used the Poisson regression method suggested by Grambsch et al. [14] rather than simply plotting size against the residuals of the null model, which is normally done for usual Cox models as suggested by Therneau et al. [39]. In the variance estimation of  $\hat{\beta}$ , we also calculated the robust sandwich estimates using the jackknife [38].

## 4 Data for Recurrent Event Modeling

In this section, we explain the data used for recurrent event modeling, which was formatted as shown in Table 1. This table presents hypothetical observations for demonstration purposes. The subjects, Y and Z, are classes, and the events of interest here are defect fixes. Each new class introduced to the system

name	start	end	event	size	state
Y	0	50	0	75	0
Y	50	100	1	200	1
Y	100	200	0	300	1
Z	0	200	1	250	0
Z	200	800	0	180	1
Z	800	1400	1	400	1
Z	1400	1800	0	300	1
.	.	.	.	.	.
.	.	.	.	.	.

**Table 1. Data Layout Used in the Study**

during an *observation period* is followed up until the observation period ends or until the class is deleted. Modifications made during the follow-up time are entered as *observations*, which correspond to the rows in Table 1. Each modification creates a new observation with a  $(start, end]$  time interval, where *start* is a time infinitesimally greater than the modification time; and *end* is either the time of the next modification, or the end of the observation period, or the time of deletion, whichever comes first. The open bracket on the left and the closed bracket on the right mean that, for any class, at *end* time  $t$ , the observation that has  $t$  in its *end* column should be used in the internal computations for the Cox model. For example, for class Y and  $t = 200$ , the third row should be used. Open and closed brackets just enable us to model non-overlapping observations. They are not related to the timing of the other data items, which is explained below.

At this point, it is useful to distinguish between calendar time and analysis (also called study) time. It should be noted that, the *start* and *end* values shown in Table 1 are analysis times. Classes are usually introduced to the system at different calendar times. However, for analysis purposes, these different calendar times are all considered time 0. As a result, the *end* times for the last observations will be usually different too. From the analysis viewpoint, they correspond to the times when follow up ends.

When a class is introduced to a system, a new observation is entered with  $start = 0$ . The event cell is set to 1 if an event (defect fix) takes place at the time represented by *end*, or 0 otherwise. A class deletion is handled easily by entering a final observation whose event is set to 1 if the class is deleted for corrective maintenance, or 0 otherwise. *size* is a *time-dependent covariate* and its column carries the source LOC values of the classes at time *start*. *size* might change during successive observations or remain constant like a *fixed covariate* too. The *state* column in Table 1 is used to create a conditional model as discussed in Section 5 in more detail. For any class, *state* is initially set to 0, and it becomes 1 after the class experiences an event, and remains at 1 thereafter.

In Table 1, Class Y has 75 LOC when it is first introduced. This class is modified at time 50. However, the change is not a corrective one, which is indicated by a zero in the *event* column. *size* of Y becomes 200 LOC, and a new observation for this class starts. A corrective change happens at time 100, represented by a 1 in the corresponding cell of the event column. This corrective change makes *size* 300 LOC. Then, one of the two things happens: either our observation period ends when the follow-up time for this class is 200, or

the class is deleted at time 200 without an event. The history of Class Z can be interpreted in a similar manner.

We developed Perl scripts to extract data from the CVS of the Mozilla project between May 29, 2002 (with the release of Mozilla 1.0) and Feb 22, 2006, which was the observation period for our study. For each CVS commit, our programs detected the changed files and the changed lines in them. Then, they checked whether these changes fell into the boundaries of any existing C++ classes. When drawing the boundaries of a class, we took into account that a class can spread over header and implementation files. During this process, we also kept track of the boundary changes during successive modifications of a class and made sure that the most recent boundary information was used at all times. For each CVS commit, our programs updated the *project database* kept by a reverse engineering tool, called *Understand for C++* [33], and extracted size values of the changed classes.

By following the above procedure, we obtained a complete measurement history of every single C++ class introduced to Mozilla during the observation period. To our best knowledge, no study in the literature obtained such detailed measurement and defect data from a large OSS project. At the end, our data set looked like the hypothetical one shown in Table 1. The start and end times were computed in *minutes* based on the time tags of the CVS commits. The event data were obtained by automatically parsing the log portions of CVS commits and searching for the words 'defect', 'fix', and 'bug' in a non-case-sensitive manner to detect corrective changes. Once a CVS commit was classified as a corrective change, the affected classes were determined together with their most recent observations. The *event* fields of those observations were set to 1. This automated approach was preferred because of the large number of CVS commits (10,459) that took place during our long observation period. To examine the effectiveness of this approach, we sampled 100 CVS logs randomly. We went through them manually and classified them as either corrective or non-corrective changes. Then, we compared our manual classification results with those of the automated approach. This comparison showed that the classification accuracy of the automated approach was 98%. There is also evidence that Mozilla developers entered meaningful CVS logs because the project had a defect handling process in place. Koru and Tian [23] performed a survey of defect handling practices in OSS projects in 2002. They reported that Mozilla scored the highest in defect recording consistency and completeness. Therefore, the collected event data were of high quality and

Min.	25%	Median	Mean	75%	Max.	Std. Dev.
1	36	126	434	418	12,360	911.24

**Figure 1. Summary of the size data in LOC**

appropriate for research purposes.

At the end, we obtained 15,545 observations that belonged to 4,089 classes created during our observation period. 8,828 observations out of 15,545 had an event. The data set is publicly available at the PROMISE data repository. Table 1 shows the summary of the size data for all observations.

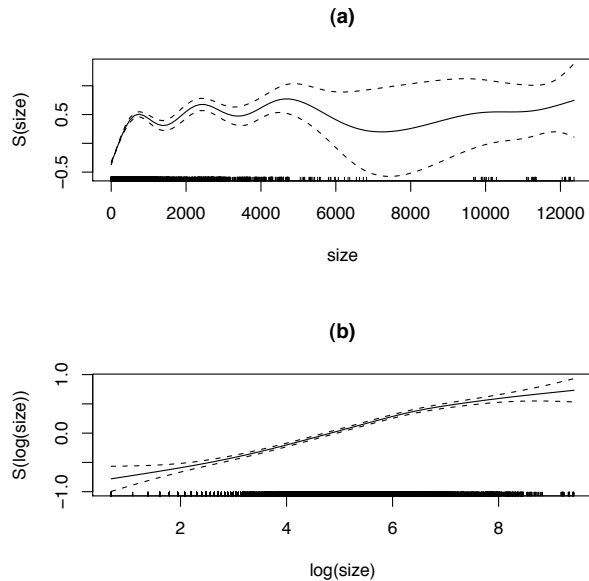
## 5 Modeling and Results

We started by examining the functional form of size. Equation 1 requires that the hazard ratios of any class pairs that have the same size difference be the same. In practice, this may or may not be the case. Therefore, we need to understand whether size can be directly used in our models, and if not, which functional form of size can be used. If we take the natural log of both sides of Equation 1, we obtain:

$$\ln(\lambda_i(t)) = \ln(\lambda_0(t)) + \beta x_i(t) \quad (9)$$

As seen here,  $x_i(t)$  should be *linearly* related to log hazard. If not, and if there is a functional transformation of it,  $f(x_i(t))$ , that is linearly related to the log-hazard, we want to find it and use it instead of  $x_i(t)$ . Therneau et al. [39] suggested plotting the martingale residuals from a null model, a model where  $\beta$  is zero for covariates, against each covariate separately. However, when there are multiple observations about a subject, a different approach should be taken because of the intra-subject correlations. Trying to follow the simple method of Therneau et al. [39], we could sum up the Martingale residuals per subject. However, we cannot sum up the covariate values in the same way or choose only one of them per subject. Therefore we used the Poisson approach suggested by Grambsch et al. [14]. Since a counting process is a very slow Poisson process, a Poisson regression model can be used for approximation to the Cox model.

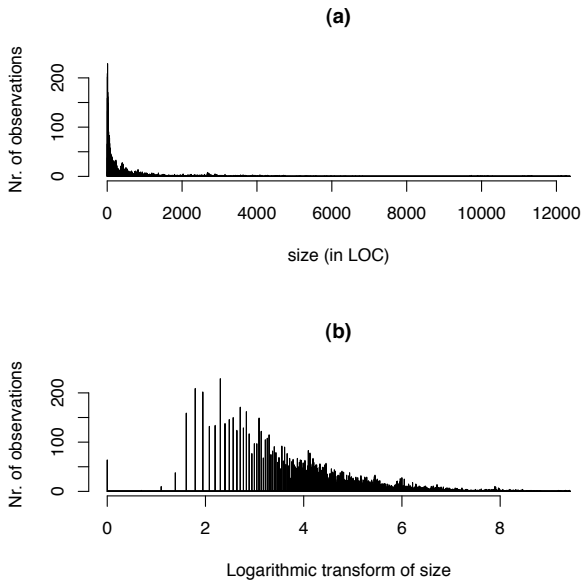
The two plots generated using the Poisson approach to study the functional form of size are shown in Figure 2. Let us discuss Figure 2(a) first. In this plot, the  $x$ -axis represents size. The  $y$ -axis is the smooth function of size in the Poisson regression model, denoted by  $S(\text{size})$ , which was created to have a linear relationship with the dependent variable of the Poisson model, *event*. The dashed lines represent the confidence interval. There is a tick mark at the bottom



**Figure 2. Functional form of (a) size in LOC (b) logarithmic transform of size**

for each unique size value in the data set. This plot clearly shows a logarithmic form for size. The plot starts to curve down for size greater than 5,000 LOC. However, the percentage of those observations is very small (0.4%), and the confidence interval gets much wider for the extreme large sizes. The second plot in Figure 2(b) shows the functional form when size is log-transformed. The curve is very close to linear. Therefore, the logarithmic form was very appropriate, and we used it in our model. Another advantage of using the logarithmic transform is that its distribution is closer to normal distribution compared to size distribution, which is usually preferred in statistical analysis. Note that Cox model is semi-parametric for only it treats time to event in a comparative sense. Therefore, having a distribution closer to normal is still preferred. The histograms of size and its logarithmic transform are shown in Figures 3(a) and 3(b), respectively.

Before developing our Cox model, we examined whether the numbers of events in non-overlapping time intervals were independent or not. We observed that the intensity of the counting process changed as more events were experienced. Time to first event was usually longer compared to the time to successive events. Once classes became defective, they became more susceptible to defect fixes. Therefore, we created a *conditional model* by using two *states*, *non-defective* and *defective*. In conditional models, subjects make state



**Figure 3. Histograms of (a) size (b) logarithmic transform of size.**

transitions, and they are considered at different risk sets according to their state. A nice characteristic of any Cox model is that it can incorporate categorical variables as strata and create a different baseline hazard for each category. In our case, the two different baseline hazards corresponded to the non-defective and defective states. The model is conditional in the sense that a class (subject) starts in the non-defective state, and it cannot enter the defective state and be at risk for that state before it experiences an event. The state column in Table 1 served the purpose of creating these strata.

The resulting Cox model is seen in Figure 4. The model shows that size is highly significant with a very large  $z$ -statistic and a zero  $p$  value when entered using log transformation. The entire model is also very significant as shown by the Likelihood ratio, Wald, score, and robust score tests. Both normal and robust estimates show this significance. The estimate of the coefficient,  $\hat{\beta}$ , for  $\log(size)$  is 0.368, and its standard error estimate is 0.00732. The robust sandwich estimate of the standard error, which takes the intra-subject correlation into account, is 0.018. Both of these standard error estimates are small, therefore we can safely use  $\hat{\beta} = 0.368$ .

At this step, we check the proportional hazards assumption by testing whether  $\log(size)$ 's interaction with time is significant:

	coef	exp(coef)	se(coef)
$\log(size)$	0.368	1.44	0.00732
	robust se	z	p
$\log(size)$	0.018	20.4	0

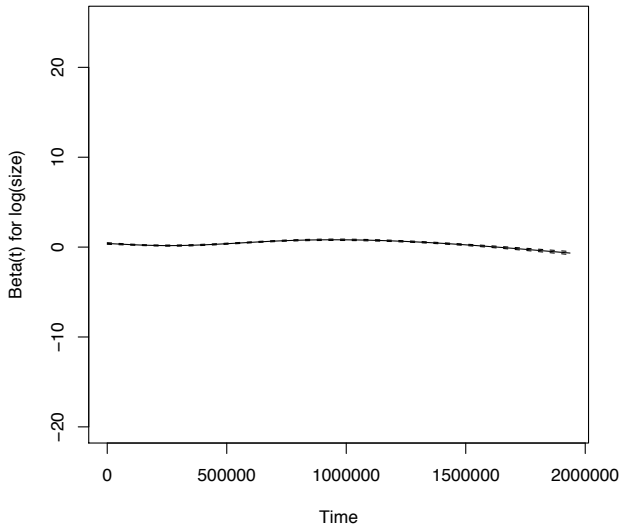
Rsquare= 0.152 (max possible= 1)  
 Likelihood ratio test= 2565 on 1 df, p=0  
 Wald test = 416 on 1 df, p=0  
 Score (logrank) test = 2565 on 1 df, p=0,  
 Robust Score = 142 p=0

**Figure 4. Modeling results using logarithmic transform of size**

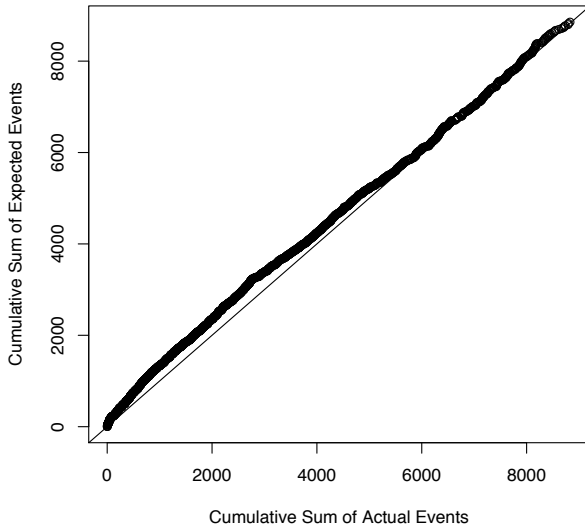
	rho	chisq	p
$\log(size)$	-0.000859	0.0433	0.835

As shown by  $p = 0.835$ , the result of this test is highly insignificant. Therefore, we accept the null hypothesis that  $\log(size)$  has no interaction with time.  $\beta$  is constant over time, which satisfies the proportional hazards assumption of the Cox model. For visual inspection of this assumption, it is common to plot a graph of the scaled Schoenfeld residuals, along with a smooth curve that represents  $\beta(t)$ . As shown in Figure 5, the smoother is almost a perfectly straight curve. We plotted only the smoother in Figure 5 because the scaled Schoenfeld residuals would obstruct the smoother otherwise. To check the overall adequacy of this model, we plotted the cumulative expected event counts against the cumulative actual defect counts only for classes that had at least one non-censored observation. Before plotting, the classes were sorted in the increasing order of their probability of experiencing an event at the end of their last observation. If a model is adequate, the plot should be close to  $45^\circ$  line. This method was recommended by Arjas [3], and it is appropriate for censored data such as time to event data. The estimated number of events per subject was calculated using the survtest function of Harrell in the Design library [19], which created a survival curve for each class by taking its entire size history into account. For any individual class, the estimated cumulative hazard derived from its curve gave the expected number of events.

The resulting plot, shown in Figure 6, closely follows the  $45^\circ$  line. We also examined the correlations between the expected and actual events. The Spearman's correlation was 0.77 and the Somer's  $D_{xy}$  rank correlation was 0.71. As a result, the model shown in Figure 4 successfully passes all the tests for a good



**Figure 5. Plot of  $\beta$  over time measured in minutes**



**Figure 6. Plot of cumulative sum of actual events versus cumulative sum of expected events**

fitting model. Using this model, the effect of size is interpreted as follows:

*For Mozilla classes, one unit of increase in the natural log of size caused the rate of defect fix to increase by 44%.*

We are also interested in finding what would happen if, for each class, we simply use the initial value of its size for all of its observations. In this case, classes A and B in Table 1 would have fixed covariate values, which are the size values of 75 and 300, respectively. Statistically, is there enough evidence to reject the fixed covariate model when our conditional model is the alternative model?

To answer this question, we regenerated the data with fixed covariate values and produced another model. The log partial likelihood of the conditional model,  $L_c$ , was  $-60,713.4$ , and that of the fixed covariate model,  $L_{fc}$ , was  $-61056.51$ . The log partial likelihood ratio test under the null hypothesis that the fixed-covariate model fits better was:

$$G = 2(L_c - L_{fc}) = 686.22 \quad (10)$$

The significance level for the test was  $Pr(\chi^2(1) \geq 686.22) \simeq 0$ , so we reject the null hypothesis and the fixed-covariate model.

## 6 Conclusion

The traditional quality modeling approaches use measurements from specific system releases (snapshots) and post-release defect counts. We used Cox proportional hazards modeling with recurrent events. Our approach addresses the evolutionary aspects of OSS development, such as changes in the module size, and addition and deletion of software modules over time. Rather than using the number of defect fixes, we used the instantaneous hazard of receiving defect fixes as an indicator of defect-proneness.

When related to defect-proneness, the functional form of size was logarithmic. We found that there was a significant relationship between the logarithmic transformation of size and defect-proneness, and we quantified this effect. We also compared our model with the fixed covariate model and found that our model was superior, underlying the importance of taking the measurement changes into account.

The modeling approach that we suggested has an important advantage over traditional modeling approaches. With the traditional approaches, it is necessary to wait until the first release and then count the post-release defects. Those approaches cannot provide useful benefits in OSS development settings until

the second release, when what is learned from the first model can be used for preventive actions. In addition, defect counts obtained between the first and second releases would be biased by the highly possible feature enhancements.

Our proposed model can be tightly integrated into OSS development. It is possible to automate this approach by keeping a database of the classes in an object-oriented OSS system, obtaining measurements for the changed classes at each CVS commit, and collecting defect data from developers. The capabilities of existing OSS tools can be combined for this purpose.

As a future extension to this work, we plan to collect data from additional OSS products and projects to generalize the observed relationship between size and defect proneness across a set of different OSS development environments.

## Acknowledgments

We are thankful to Dr. Frank E. Harrell, who extended his *survest* function in the Design package for us to develop survival curves for classes that had recurrent events data.

## References

- [1] Per Kragh Andersen, Ornulf Borgan, Richard D. Gill, and Niels Keiding. *Statistical Models Based on Counting Processes*. Springer-Verlag, 1993.
- [2] Per Kragh Andersen and Richard D. Gill. Cox's regression model for counting processes: A large sample study. *Annals of Statistics*, 10:1100–1120, 1982.
- [3] Elja Arjas. A graphical method for assessing goodness of fit in Cox's proportional hazards model. *Journal of the American Statistical Association*, 83:204–212, 1988.
- [4] Victor Basili and Barry T. Perricone. Software errors and complexity: an empirical investigation. *Communications of the ACM*, 27(1):42–52, January 1984.
- [5] David N. Card and Robert L. Glass. *Measuring Software Design Quality*. Prentice Hall, Englewood Cliffs, New Jersey, 1990.
- [6] David R. Cox. Regression models and life tables. *Journal of the Royal Statistical Society*, 34:187–220, 1972.
- [7] Jr. David W. Hosmer and Stanley Lemeshow. *Applied Survival Analysis :Regression Modeling of Time to Event Data*. John Wiley & Sons, Inc., 1999.
- [8] Trung T. Dinh-Trong and James M. Bieman. The FreeBSD Project: A Replication Case Study of Open Source Development. *IEEE Transactions on Software Engineering*, 31(6):481–494, 2005.
- [9] Khaled El Emam, Saida Benlarbi, Nishith Goel, Walcelio Melo, Hakim Lounis, and Shesh N. Rai. The Optimal Class Size for Object-Oriented Software. *IEEE Trans. on Software Engineering*, 28(5):494–509, 2002.
- [10] Khaled El Emam, Saïda Benlarbi, Nishith Goel, and Shesh N. Rai. The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics. *IEEE Trans. on Software Engineering*, 27(7):630–650, July 2001.
- [11] Norman E. Fenton and Niclas Ohlsson. Quantitative Analysis of Faults and Failures in a Complex Software System. *IEEE Trans. on Software Engineering*, 26(8):797–814, August 2000.
- [12] Michael W. Godfrey and Qiang Tu. Evolution in open source software: A case study. In *ICSM '00: Proceedings of the International Conference on Software Maintenance (ICSM'00)*, page 131, Washington, DC, USA, 2000. IEEE Computer Society.
- [13] Michael Goulde and Eric Brown. Open Source Software: A Primer for Health Care Leaders. (prepared for ): California Health Care Foundation, March 2006.
- [14] Patricia M. Grambsch, Terry M. Therneau, and Thomas R. Fleming. Proportional hazards tests and diagnostics based on weighted residuals. *Biometrika*, 81:515–526, 1994.
- [15] Tibor Gyimothy, Rudolf Ferenc, and Istvan Siket. Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. *IEEE Trans. Softw. Eng.*, 31(10):897–910, 2005.
- [16] Rachel Harrison, Steve J. Counsell, and Reuben V. Nithi. An Investigation into the Applicability and Validity of Object oriented Design Metrics. *Journal of Empirical Software Engineering*, 3(3):255–273, September 1998.
- [17] Les Hatton. Reexamining the Fault Density-Component Size Connection. *IEEE Software*, 14(2):89–97, March/April 1997.

- [18] Leslie Hatton. Does OO sync with the way we think? *IEEE Software*, 15(3):46–54, May/June 1998.
- [19] Frank E Harrell Jr. *Design: Design Package*, 2005. R package version 2.0-12.
- [20] Frank E Harrell Jr and with contributions from many other users. *Hmisc: Harrell Miscellaneous*, 2006. R package version 3.1-2.
- [21] A. Güneş Koru, Khaled El-Emam, Angelica Neisa, and Medha Umarji. A survey of quality assurance practices in biomedical open-source software projects. *Journal of Medical Internet Research*, In press, 2007.
- [22] A. Güneş Koru and Hongfang Liu. Building effective defect prediction models in practice. *IEEE Software*, 22(6), November/December 2005.
- [23] A. Güneş Koru and Jeff Tian. Defect Handling in Medium and Large Open Source Projects. *IEEE Software*, 21(4):54–61, July/August 2004.
- [24] Steve McConnell. Open Source Methodology: Ready for Prime Time? – From the Editor. *IEEE Software*, 16(4):1–6, Jul-Aug 1999.
- [25] Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, 2007.
- [26] Audris Mockus, Roy T. Fielding, and James Herbsleb. Two case studies of Open Source Software Development: Apache and Mozilla. *ACM Trans. on Software Engineering and Methodology*, 11(3):309–346, July 2002.
- [27] K. Moeller and D.J. Paulish. An Empirical Investigation of Software Fault Distribution. In *First International Software Metrics Symposium*, pages 82–90, may 1993.
- [28] Harvey J. Murff and Joseph Kannry. Physician Satisfaction with Two Order Entry Systems. *Journal of the American Medical Informatics Association*, 8:499–511, 2001.
- [29] S original by Terry Therneau and ported by Thomas Lumley. *survival: Survival analysis, including penalised likelihood*. R package version 2.29.
- [30] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2003. ISBN 3-900051-00-3.
- [31] Eric S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O’Reilly and Associates, Sebastopol, CA, 95472, USA, 1999.
- [32] Michael Schemper and Janez Stare. Explained variation in survival analysis. *Statistics in Medicine*, 15:1999–2012, 1996.
- [33] Scientific Toolworks, Inc. *Understand for C++: User Guide and Reference Manual*. 321 N. Mall Drive Suite I-201, St. George, UT 84790, January 2003.
- [34] Richard W. Selby and Victor R. Basili. Analyzing Error-Prone System Structure. *IEEE Trans. on Software Engineering*, 17(2):141–152, 1991.
- [35] M. Sherriff, L. Williams, and M. Vouk. Using In-Process Metrics to Predict Defect Density in Haskell Programs. In *15th International Symposium on Software Reliability Engineering*, St-Malo, France, November 2004.
- [36] Param Vir Singh and Yong Tan. Planning to first release: A conditional hazard function approach for investigating open source software development time. In V. Ramesh and Atish Sinha, editors, *Proceedings of the Sixteenth Annual Workshop on Information Technologies and Systems, WITS 2006*, pages 127–132, Milwaukee, Wisconsin, USA, December 2006.
- [37] Jacqueline E. Stark. *Peer Reviews in Open-Source Software Development*. Thesis, School of Computing and Information Technology, Griffith University, October 2001.
- [38] Terry M. Therneau and Patricia M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. Springer-Verlag, 2000.
- [39] Terry M. Therneau, Patricia M. Grambsch, and Thomas R. Fleming. Martingale based residuals for survival models. *Biometrika*, 77:147–160, 1990.
- [40] Jeff Tian. Risk Identification Techniques for Defect Reduction and Quality Improvement. *Software Quality Professional*, 2(2):32–41, March 2000.
- [41] Carol Withrow. Error Density and Size in Ada Software. *IEEE Software*, 7(1):26–30, 1990.
- [42] Luyin Zhao and Sebastian Elbaum. Quality Assurance Under the Open Source Development Model. *Journal of Systems and Software*, 66(1):65–75, 2003.