Effect of Coupling on Defect Proneness in Evolutionary Open-Source Software Development

A. Güneş Koru[1], Dongsong Zhang[2], and Hongfang Liu[3]

[1] Department of Information Systems, UMBC gkoru@umbc.edu
[2] Department of Information Systems, UMBC zhangd@umbc.edu
[3] Department of Biostatistics, Bioinformatics, and Biomathematics, Georgetown Medical Center hl224@georgetown.edu

Abstract. Previous research on closed-source software found that highly coupled software modules were more defect prone, which makes it important to understand the effect of coupling on defect proneness in open-source software (OSS) projects. For this purpose, we used Cox proportional hazards modeling with recurrent events. We found that the effect of coupling was significant, and we quantified this effect on defect proneness.

Key words: Open-source software, object-oriented software, defect proneness, coupling, Cox proportional hazards model, recurrent events, Mozilla.

1 Introduction

Coupling is the degree to which a program element is related to or interacts with other program elements. The higher the average coupling of elements in software, the more complex and defect prone it is considered to be [7]. The previous research on closed-source software has shown that highly coupled software modules are more defect prone compared to less coupled ones [2, 5]. Therefore, it is important to build statistical models to understand the relationship between coupling and defect-proneness in OSS.

However, the evolutionary aspects of OSS development processes require specialized modeling techniques. The structural characteristics of OSS modules (e.g., coupling) can change in the post-release period. Making the situation even more complicated, new modules can be added or some modules might be removed from a system shortly after measurement time. The traditional approaches to quality modeling, which measure specific system snapshots and relate them to future defect counts, cannot accommodate these special characteristics of OSS.

The main research contribution of this study is to develop and evaluate a statistical model in order to understand the effect of coupling on defect proneness while taking the dynamic aspects of OSS development into account. For

Please use the following format when citing this chapter:

Koru, A.G., Zhang, D., and Liu, H., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 271–276.

this purpose, we adopted Cox proportional hazards modeling with recurrent events.

In the rest of the paper, we first explain our modeling approach and the data used in the study. Then, we present our modeling results. Finally, we discuss the implications of this work and conclude the paper.

2 Cox Proportional Hazards Model for Recurrent Events

Cox proportional hazards model [3] (henceforth *Cox model*) has become the most common technique used for various time-to-event analysis purposes in many fields [4, 8]. Cox model is connected to the counting process and Martingale theory [1], which makes it suitable for recurrent events. In recurrent event modeling, an event of interest is observed for a subject multiple times during a follow-up period [4].

In our study, each defect fix made to a class was considered an *event*. Being more defect prone meant having a higher risk of having events. We had a single time-dependent covariate, Coupling Between Objects (CBO), denoted by $x(t)$ below. CBO for a class C is defined as the number of methods and instance variables of other classes used by C.

We specify the *hazard function*, which is the instantaneous risk of an event for class i at time t, as:

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

β is the coefficient for $x_i(t)$ and λ_0 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, when $\beta = 0$.

Cox model is semi-parametric because it does not explicitly describe a baseline hazard function. It 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 β should remain constant over time. This is an important assumption of any Cox model, known as proportional hazards assumption. We checked this assumption for our model (see Section 4). The details of the Cox model, such as the estimation of β, can be found in [4, 8].

3 Data for Recurrent Event Modeling

Table 1 presents hypothetical data for demonstration purposes. The subjects are classes, and the events of interest here are defect fixes. Each new class introduced to the system 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 *observation*s, which correspond to the rows in Table 1.

name	start	end	event	CBO	state
A	0	10	0	5	0
A	10	30	1	8	0
A	30	50	0	9	1
B	0	20	1	3	0
B	20	80	0	2	1
B	80	120	1	5	1
B	120	150	0	5	1
.	.	.	.	.	
.	.	.	.	.	

Table 1. Data Layout Used in the Study

Each modification creates a new observation with a $(start, end]$ time interval, where *start* is a time infinitesimally greater than the modification time; *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 at any *end* time t, the observation that has t in its *end* column should be used in the internal computations of the Cox model. For example, for $t = 50$, the third row should be used. Open and closed brackets enable us to model non-overlapping observations. They carry no meaning about the timings of other data items, which are explained below.

When a class is added to the 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. CBO is a *time-dependent covariate* and its column carries the coupling measurements of the class at *start*. Its value may change during successive observations but can remain constant like a *fixed covariate* too. The *state* column in Table 1 is used to create a conditional Cox model. For any class, *state* is initially set to 0, and it becomes 1 after the class experiences an event, and always remains at 1 thereafter.

We developed Perl scripts to extract data from the CVS (Concurrent Versions System) of the Mozilla project between May 29, 2002 (with the release of Mozilla 1.0) and Feb 22, 2006, which was the observation period of our study. We obtained a complete measurement history of every single C++ class introduced to Mozilla during this observation period. 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. Our manual examination of 100 randomly collected CVS logs showed that the accuracy of the automated approach was 98%. Once a CVS commit was classified as a corrective change, the effected classes were determined with their most recent observations. The *event* field of

those observations was set to 1. At the end, we obtained 15,545 observations that belonged to 4,089 classes.

4 Modeling and Results

The resulting conditional Cox model is shown in Figure 1. The model shows that CBO is highly significant with a very large z-statistic and a zero p value when entered using log transformation. This functional form of CBO was determined by inspecting the plots obtained by using the Poisson approximation [8]. 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 coefficient for the $log1p(CBO)$[1] is 0.661, and its standard error estimate is 0.0117. The robust sandwich estimate of the standard error, which takes the intra-subject correlation into account, is 0.0297. Both of these standard error estimates are small, therefore, we can safely use $\hat{\beta} = 0.661$.

```
  n= 15545
            coef exp(coef) se(coef) robust se    z p
log1p(CBO) 0.661      1.94   0.0117    0.0297 22.3 0

           exp(coef) exp(-coef) lower .95 upper .95
log1p(CBO)      1.94      0.516      1.83      2.05

Likelihood ratio test= 3200  on 1 df,   p=0
Wald test            = 497  on 1 df,    p=0
Score (logrank) test = 3271  on 1 df,   p=0,   Robust = 148  p=0
```

Fig. 1. Modeling results using log CBO

There was no interaction between log CBO and time ($p = 0.93$). Therefore, the proportional hazards assumption of the Cox model was satisfied. An Arjas plot between the cumulative expected and cumulative actual number of events was drawn to see the overall fitness of the model. This plot closely followed the 45^o line, which showed good fitness. We also looked at the correlations between the expected and actual events. The Spearman's correlation was 0.77 and the Somer's D_{xy} rank correlation was 0.72. As a result, the model shown in Figure 1 has passed all the tests for a good fitting model.

5 Implications

The model in Figure 1 indicates that one unit of increase in the natural log of coupling caused Mozilla classes to experience a defect fix at a rate 94% higher.

[1] To accommodate $CBO = 0$, the natural log was taken after adding 1.

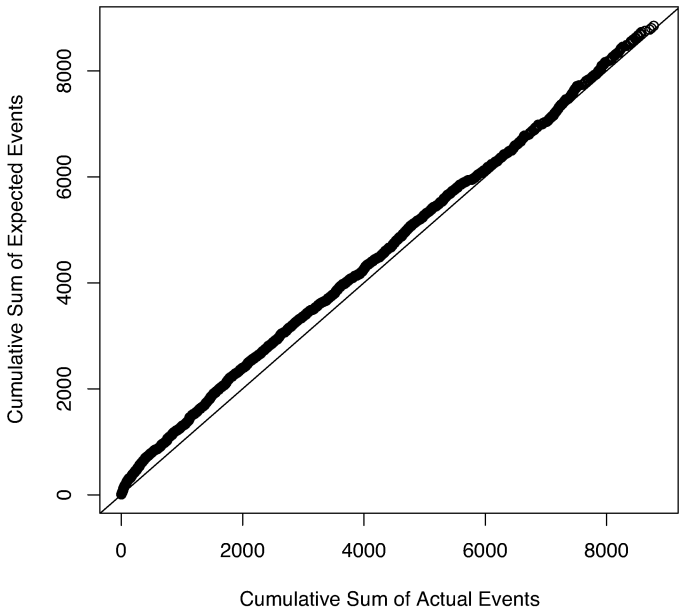

Fig. 2. Plot of cumulative sum of actual events versus cumulative sum of expected events

Our results have important implications considered the recent findings about the coupling in some OSS products.

Schach and Offutt [6] found that the degree of common coupling in the Linux kernel posed risks to the first release, and this situation deteriorated during the successive releases of this kernel. Yu et al. [9] performed a categorization of common coupling within kernel-based software and applied this categorization to the Linux kernel. They argued that without preventive actions, the maintainability of the Linux kernel would continue to be problematic. In a comparison of coupling in different OSS products, Yu et al. [10] found that the Linux kernel compared unfavorably with respect to three BSD kernels, namely, FreeBSD, NetBSD, and OpenBSD.

The above findings combined with our results show that OSS developers can take some preventive actions to improve quality. First, the quality assurance activities, such as inspections and testing, can be focused on highly coupled modules. Second, restructuring OSS software to reduce coupling can also improve quality in the long run.

6 Conclusion

The dynamic nature of OSS development requires a dynamic modeling approach to understand the relationship between coupling and defect proneness well. The traditional approaches that measure systems snapshots and count future defects cannot accommodate changing measures, added modules, deleted modules, etc.

Our modeling results showed that coupling has a significant effect on defect-proneness. Therefore, coupling should be monitored and managed in OSS projects to produce reliable and maintainable OSS products. The modeling approach explained here can be tightly integrated into an evolutionary OSS development in a seamless manner and can be used at any time while building models. The capabilities of the existing OSS tools can be easily combined for this purpose.

As the future work, we plan to collect data from additional OSS products and projects to generalize the identified relationship between coupling and defect proneness across a set of different OSS products.

References

1. Per Kragh Andersen, Ornulf Borgan, Richard D. Gill, and Niels Keiding. *Statistical Models Based on Counting Processes*. Springer-Verlag, 1993.
2. Lionel C. Briand, Jürgen Wüst, John W. Daly, and D. Victor Porter. Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems. *Journal of Systems and Software*, 51(3):245–273, 2000.
3. David R. Cox. Regression models and life tables. *Journal of the Royal Statistical Society*, 34:187–220, 1972.
4. Jr. David W. Hosmer and Stanley Lemeshow. *Applied Survival Analysis :Regression Modeling of Time to Event Data*. John Wiley & Sons, Inc., 1999.
5. 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.
6. Stephen R. Schach and Jefferson A. Offutt. On the Nonmaintainability of Open-Source Software. In *Meeting Challenges and Surviving Success: The 2nd Workshop on Open Source Software Engineering*, pages 52 – 54, Orlando, Florida, May 2002.
7. Katherine J. Stewart, David P. Darcy, and Sherae L. Daniel. Opportunities and challenges applying functional data analysis to the study of open source software evolution. *Statistical Science*, 21:167–178, 2006.
8. Terry M. Therneau and Patricia M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. Springer-Verlag, 2000.
9. Liguo Yu and Kai Chen. Categorization of common coupling and its application to the maintainability of the linux kernel. *IEEE Trans. on Software Engineering*, 30(10):694–706, 2004. Member-Stephen R. Schach and Member-Jeff Offutt.
10. Liguo Yu, Stephen R. Schach, Kai Chen, Gillian Z. Heller, and A. Jefferson Offutt. Maintainability of the kernels of open-source operating systems: A comparison of linux with freebsd, netbsd, and openbsd. *Journal of Systems and Software*, 79(6):807–815, 2006.