Studying the Effect of Co-change Dispersion on Software Quality

Ehsan Kouroshfar
Computer Science Department, George Mason University
Fairfax, USA
ekourosh@gmu.edu

***Abstract*—Software change history plays an important role in measuring software quality and predicting defects. Co-change metrics such as number of files changed together has been used as a predictor of bugs. In this study, we further investigate the impact of specific characteristics of co-change dispersion on software quality. Using statistical regression models we show that co-changes that include files from different subsystems result in more bugs than co-changes that include files only from the same subsystem. This can be used to improve bug prediction models based on co-changes.**

***Index Terms*—mining software repository, bugs, changes.**

I. Problem and Motivation

Many researchers have used the information in source code, change history, and bug repositories to predict files that are likely to have defects in the future. Several studies indicate that prior modifications to a file are a good predictor of its fault potential [1, 2, 3, 4, 5]. For example, many studies show that the more a file is changed the more likely it is to contain faults. Sometimes several files are being changed together in a single commit to source code repository. We use the term co-change here as changes of files that are included in a single modification request. Several studies show that the number of co-changes is also a good predictor of faults [1, 2]. For example, Hassan shows that the more spread the changes, the higher is the complexity of making those changes, thereby resulting in more bugs [2]. In addition, co-changes can be an indicator of a cross-cutting concern and concern scattering is known to be correlated with the number of defects [6, 7].

Although co-change could be a predictor for cross-cutting concerns, we believe that not all the co-changes are the same. A co-change that includes modules from different subsystems is a stronger predictor of cross-cutting concerns than a co-change that changes modules from the same component. The difference between previous studies on co-changes and ours is that we are investigating the impact of co-changes on the software system's quality from an architectural standpoint. We hypothesize that co-changes contained within the same subsystem do not have the same effect as co-changes dispersed across different subsystems. We believe we can obtain better insight about the relationship between co-changes and software quality by analyzing the nature of co-changes in more detail.

A prerequisite in conducting this study is that we need to have insights into the subsystems comprising a software system under investigation. Unfortunately, most of the existing open source projects do not explicitly document the software architecture of the system. To address this challenge, we used Bunch, a reverse engineering tool that produces a subsystem decomposition model of a given software system by partitioning the system into a graph of entities (e.g., classes) and relations (e.g., function calls) in the source code [8].

Empirical investigation of four apache projects has verified our hypothesis that co-changes in the same subsystem result in fewer bugs than co-changes in the different subsystems. This new insight on the nature of co-changes can be used to improve bug prediction models based on co-changes (e.g., [1]).

II. Related Work

Shihab et al. show that the number of co-changed files is a good indicator of defects that appear in unexpected locations (surprise defects) [1]. Hassan predicts defects using the entropy (or complexity) of code changes [2]. The more spread the changes, the higher is the complexity. D'Ambros et al. show that there is a relationship between change coupling and software defects [9].

Breu uses co-changes to identify cross-cutting changes [6]. The idea is that a code change is likely to introduce a cross cutting concern if various locations are modified within a single code change. Eaddy et al. show that there is a strong statistically significant correlation between the degree of concern scattering and the number of defects [7]. Several empirical studies provide evidence that crosscutting concerns degrade code quality because they negatively impact internal quality metrics such as program size, coupling and separation of concerns [10].

III. Approach

Our hypothesis is that co-changes that are located across multiple subsystems result in more bugs than co-changes affecting only a single subsystem.

A. Data Collection

In this study we use four apache open source projects: Camel, OpenJPA, Hive, and HBase. The first step is to find out which files have been changed together. Fig. 1 shows how we collected data in a three-month interval. In the first 3 months we obtain the information of co-changes from the source code repository. Then in the next three months we find the files that have been changed to fix bugs (bug fixes). We want to have equal periods of time for collecting co-changes and collecting bug fixes in order to have a meaningful comparison of the results.

978-1-4673-3076-3/13 © 2013 IEEE

ICSE 2013, San Francisco, CA, USA
ACM Student Research Competition

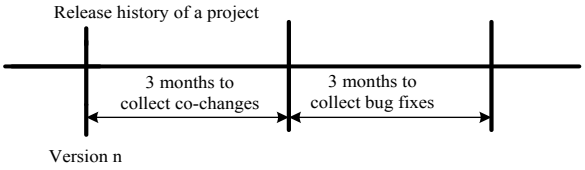

Figure 1. Data collection

B. Finding Subsystems

We used *Bunch,* which is a reverse engineering tool and provides subsystems (clusters) based on the dependencies between modules [8]. Bunch is a fast, scalable and easy to use tool for reverse engineering of software projects. The clustering results of bunch are based on the static structure of the source code. Bunch uses some search techniques and heuristics (e.g., hill climbing) and finds a good partition in a way that interdependent modules are grouped in the same subsystems and independent modules are assigned to separate ones.

C. Method of Study

We define two metrics of co-changes:

- *CCSameSub*: Number of co-changes for a file where the co-changes are made in a single subsystem
- *CCDiffSub*: Number of co-changes for a file where the co-changes are made across more than one subsystem

As an example, suppose that *a*, *b* and *c* are three of the files in the system and these are the co-changing files from three commits to the repository: *{a, b}*, *{a, b, c}*, *{b, c}*. All the files in the same set have been changed in a single commit. Suppose that files *a* and *b* are located in the same subsystem. Values of *CCSameSub* and *CCDiffSub* for the three files are shown in Table I.

The following multivariable linear regression formula is one plausible way of representing the relationship between the two types of co-changes and bugs for each file:

$$\alpha * CCDiffSub + \beta * CCSameSub = NumBugs \qquad (1)$$

NumBugs is the number of times that a file has been changed due to a bug fix as a proxy measure of software defects. α and β are coefficients that are computed from the linear regression model. We hypothesize co-changes in the different subsystems results in more bugs than co-changes in the same subsystem. If our hypothesis is true, α should be greater than β.

After we obtain the coefficients from the data collected in each data collection interval, we use t-Test to determine whether α is significantly greater than β or not.

IV. Results

We have empirically examined four apache open source projects: Camel, HBase, Hive and OpenJPA. Table II shows the values of α and β for the four projects we examined.

TABLE I. Values of metrics

	CCSameSub	CCDiffSub
a	1	1
b	1	2
c	0	2

TABLE II. Results of regression analysis

	OpenJPA		Camel		HBase		Hive	
	α	**β**	**α**	**β**	**α**	**β**	**α**	**β**
1	0.14	0.02	0.18	0.21	0.68	0.55	0.76	0.47
2	-0.52	-0.59	0.39	0.25	0.61	0.57	0.45	0.21
3	1.22	-0.03	0.29	0.22	0.54	0.40	0.16	0.00
4	0.07	0.21	0.44	0.12	1.12	-0.41	0.66	0.63
5	0.05	-0.27	0.33	0.20	0.56	-0.83	0.25	0.12
6	0.33	-0.09	0.44	0.32	1.52	-0.60	0.24	0.20
7	0.71	0.64	0.63	0.38	0.73	0.67	0.21	0.22
8	0.59	0.06	0.51	0.56	0.48	-0.51	0.32	0.24
9	0.59	0.64	0.44	0.20	1.42	2.49		
10	0.16	0.01	0.39	0.34	0.65	-0.23		
11	0.04	0.03			0.99	0.66		
12	0.13	-0.25			0.40	0.51		

Since the total duration of data collection is different for each project, we have different number of data points that were collected at three-month intervals for each project. Each row shows the result for each data collection point. For example, row 1 for OpenJPA shows the result of the study for period of the first three months.

The grey cells show the data points where α is greater than β. As it is shown, in most of the data points α is greater than β, which means that *CCDiffSub* has more impact on the number of bugs than *CCSameSub*.

Next to see if this relationship (i.e., α being greater than β) is statistically significant, we performed two-tail t-Test on the values of α and β from Table II. Table III shows the results. First two columns show the average values of α and β for each of the projects, while the last column shows the value of two tail t-Test. The results show that in all of the projects, with at least 95% confidence, α is greater than β.

TABLE III. Results of t-test

	α(average)	β(average)	t-Test
OpenJPA	0.29	0.03	0.03
Camel	0.40	0.28	0.01
HBase	0.81	0.27	0.05
Hive	0.38	0.26	0.01

V. Conclusions

We used a subsystem decomposition model of four open-source Apache projects to study how the locality of co-changes affects the system's defects. Our results show that co-changes that are localized in the same subsystem result in fewer bugs than co-changes crosscutting the different subsystems. This study could provide the foundation for enhancing several existing bug prediction models that take co-change information as an input. In addition, we believe the dispersion of co-changes among the subsystems could be used as an indicator of the architectural bad smells.

Acknowledgment

I would like to thank Dr. Sam Malek and Dr. Yonghee Shin for their advice on this research.

REFERENCES

[1] E. Shihab, A. Mockus, Y. Kamei, B. Adams, and A. E. Hassan, "High-impact defects: a study of breakage and surprise defects," In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (ESEC/FSE '11). ACM, New York, NY, USA, 300-310.

[2] A. E. Hassan, "Predicting faults using the complexity of code changes," In Proceedings of the 31st International Conference on Software Engineering (ICSE '09). IEEE Computer Society, Washington, DC, USA, 78-88.

[3] E. Arisholm and L. C. Briand, "Predicting fault-prone components in a java legacy system," In G. H. Travassos, J. C. Maldonado, and C. Wohlin, editors, ISESE, pages 8–17. ACM, 2006.

[4] T. L. Graves, A. F. Karr, J. S. Marron, and H. P. Siy, "Predicting fault incidence using software change history," IEEE Transactions on Software Engineering, 26(7):653–661, 2000.

[5] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," In Proceedings of the 27th International Conference on Software Engineering, pages 284–292, 2005.

[6] S. Breu and T. Zimmermann, "Mining Aspects from Version History," InProceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering(ASE '06). IEEE Computer Society, Washington, DC, USA, 221-230.

[7] M. Eaddy, T. Zimmermann, K. D. Sherwood, V. Garg, G. C. Murphy, N. Nagappan, and Alfred V. Aho, "Do Crosscutting Concerns Cause Defects?,". IEEE Trans. Softw. Eng. 34, 4 (July2008),497-515.

[8] B. S. Mitchell and S. Mancoridis, "On the Automatic Modularization of Software Systems Using the Bunch Tool," IEEE Trans. Softw. Eng. 32, 3 (March 2006), 193-208.

[9] M. D'Ambros, M. Lanza, and R. Robbes, "On the Relationship Between Change Coupling and Software Defects," In Proceedings of the 2009 16th Working Conference on Reverse Engineering (WCRE '09). IEEE Computer Society, Washington, DC, USA, 135-144.

[10] G. Kiczales, J. Irwin, J. Lamping, J.-M. Loingtier, C.V. Lopes, C. Maeda, and A. Mendhekar, "Aspect-Oriented Programming," ACM Computing Surveys, vol. 28, no. 4es, p. 154, 1996.