Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 46 (2015) 443 – 448

Procedia Computer Science

International Conference on Information and Communication Technologies (ICICT 2014)

Measurement of Package-Changeability by Mining Change-History

Anshu Parashar[a,]*, Jitender Kumar Chhabra[a]

[a]*Department of Computer Engineering, National Institute of Technology, Kurukshetra, 136118, India*

Abstract

During the development of object-oriented software system; huge amount of software-development data especially change-history is stored in software-repositories. This change-history can be mined to measure the quality of the software systems. In this paper, we present change-history based package-changeability measures namely; a) Package Change-Proximity, b) Change-Coupling Index and c) Package-Changeability. These measures are defined and computed on the basis of change-coupling among packages. A case study has been performed and findings derived from the results are discussed. Our results show that these package-changeability measures can be very helpful for development team to predict and quantify the change-coupling among packages.

Keywords: Change-history; Coupling; Changeability; Object-Oriented Software Engineering.

1. Introduction

Change is an inherent and important task of software maintenance. The change-coupling among software components need to be measured for effectively maintaining the software. As the size and complexity of software system increases, it becomes very much necessary to predict and implement the change efficiently. The software changes without predicting or knowing their ripple effect can lead to poor change management[1,3]. Now days, usually, software systems are developed using configuration or version management system. Numerous types of information such as revision-history or changes are logged in through versioning-system to keep track of changes[2]. These change-logs can be mined and analyzed to find out the change-coupling among software components (i.e. packages).

* Corresponding author. Tel.: +91-989-626-2553 ; fax: +91-174-628-0711
E-mail address: anshulphd@gmail.com

doi:10.1016/j.procs.2015.02.042

An object-oriented system consists of packages to carry out specified functionality of the software system. To accomplish changes to the packages of a software system, package-changeability measurement is very important task. It involves examining change-coupling or high-level dependencies at package level[4]. Change-history based package-changeability measurement helps to determines how packages are co-evolved or change-coupled[1-5]. By going into more details, one can also understand how elements of packages (sub-package, class etc.) are also co-evolved. The change-coupling can produce a ripple effect. It means a change in a package P_a requires package P_b to be modified because both P_a and P_b are change-coupled. A change to a system is to be implemented as a modification to components or elements of the system. We, here, refer package as a component of the object-oriented system. A change in a package may affect other packages if its subcomponents i.e. classes or sub-packages are connected or linked with classes or sub-packages of other packages. These connections or links among the packages can be explored through their change-history.

In this paper, we present package-changeability measures on the basis of software change-history. We define metrics for the computation of change-proximity and change-coupling index for packages. Further, the changeability of packages is also ranked as per their past change-coupling pattern. In our work, we consider package-changeability as the ability of a package to absorb changes. A package can be changed easily if it is less change-coupled with other packages. Apart from change-coupling, there may be some more factors that can affect the changeability of a package. But in this study, we are only concentrating on change-coupling because it can be a major factor for the poor maintenance or change-management. As we define history-based metrics; so the link among packages are observed and mined from their past co-changed or co-evolved pattern. If two packages are co-changed in a single commit then it exhibit change-coupling relationship among them. In next sub-section, we describe the process of extracting package-change-reports.

1.1. Mining package change-history

Mining package change-history can be described as a process of investigating coupling among the packages as per their past co-changed or co-evolved pattern. Generally, package-coupling is measured through design (UML) diagrams or source code [4, 6]. In present development scenario, during the evolution of the software system change-logs are stored in the repositories. Whenever any change or modification is committed; change-log is stored and reflects set of components that are changed together in that commit. These change-logs can be mined to extract change-reports related to packages of the software system. So for this purpose, change-logs are required to be filtered or preprocessed. The change-logs of large size are unnecessary because these are due to some environment change or the result of some long pending commits etc. These change-logs do not reflect change-coupling or co-evolution information about packages. After filtering, shortlisted change-logs are referred as Package-Change-Reports (PChR). For example, consider below mentioned sample change-log of a project *System-X* having four packages (P1, P2, P3 and P4):

```
Change-log-1
M/trunk/System-X/src/P1/C1.java
M/trunk/System-X/src/P3/C2.java
M/trunk/System-X/src/P1/C2.java
M/trunk/System-X/src/P4/C4.java
M/trunk/System-X/src/P3/C2.java
```

From above change-log-1, it can be seen that the classes of packages P1, P3 and P4 are committed or changed together. It indicates that packages P1, P3 and P4 are co-evolved; it means they are change-coupled. So, the package-change-report (PChR-1) is mined as under. Each package will be considered only one time in PChR.

```
PChR-1
System-X/src/P1
System-X/src/P3
System-X/src/P4
```

So, in this way change-logs are mined to form the package-change-reports. These change-reports are prerequisite data for our history-based package-changeability measures. In section 3, we describe how these change-reports are utilized for the computation of change-coupling among the packages.

The rest of the paper is organized as follows. Section 2 describes the related work. Section 3 discusses package-changeability measures and framework of computation. Section 4 describes the findings obtained from a case study. Section 5 concludes this paper.

2. Related Work

For object-oriented systems, most of the coupling metrics have been defined for class level and package level on the basis of design and source code[4,6-11]. Apart from this, history-based software quality analysis through data mining techniques is also emerging[1,2,5,12-14].

Nowadays, software development activities are efficiently recorded in the software-repositories[1,2]. Various tools and applications are available to aid the developers or maintainers to log the evolution of the software system. The svnsearch[15], clearcase[16], bugzila[17] etc. are examples of such utilities. The information stored in repositories can be mined to analyze past development and to improve future software development as well as maintenance. In present research scenario, data mining techniques are frequently used to investigate the software quality as well as to assist developers in various software engineering activities like testing, change-prediction, bug-prediction etc.[12-14,18,19]. Predicting changeability or change impact analysis is an important activity while maintaining the software. It helps to properly predict the amount of work to be done for accommodating the changes efficiently. Vanya et al. investigated co-evolving entities and discussed how interactive visualizations can support the process of analyzing the structural issues[5]. Zimmerman et al. also mined version history to guide software changes. They developed a methodology for change prediction[1]. Ying et al. and Robbes et al also analyzed change-data available in version histories[14-18]. Kagdi et al. and Beyer et al. explored past development data and used clustering techniques to predict co-change pattern[19,20].

3. Package-Changeability Measures

Investigation of change-coupling is an important technique to measure the relationship among packages of an object-oriented system. Two packages are considered as coupled if any kind of link or relationship exists between them. The change-coupling can be an important factor for establishing the dependency or coupling among the packages. In object-oriented system each package or sub-package is having its own importance and responsibility. Each package contributes towards the overall functionality of the software system. So, in this work, we are considering a package or sub-package as an individual component. In this section, firstly we describe the formation of Package-Proximity Matrix from package-change-reports. Secondly, a set of Package-Changeability measures are defined and demonstrated. Thirdly, a framework to compute these measures is also presented.

3.1. Package-proximity matrix

Package-Proximity Matrix (PPM) is constructed by mining change-history of the software system. As we described in section 1, the package-change-reports are mined from the available change-history. These change-reports tell about packages that are changed-together. Suppose two packages P_a and P_b are changed together then the weight PPM[P_a,P_b] is set to 1 and if both P_a and P_b are never changed together in past then PPM[P_a,P_b] is set to 0. Here, we are not considering the extent of co-change among two packages. Our target is to know whether two packages are co-changed or not. So, weights PPM[P_a,P_b]'s are of boolean nature. Value of each weight PPM[P_i,P_j] shows whether two packages are co-changed or co-evolved in the past or not as per package-change-reports. Table 1 shows Package-Proximity Matrix (PPM) for example package-change-report described in section 1.

Table 1. Package-proximity matrix for system-X

Packages	P1	P2	P3	P4
P1	1	0	1	1
P2	0	1	0	0
P3	1	0	1	1
P4	1	0	1	1

For example, PPM[P1,P3] is equal to 1. It means they are co-changed in past. PPM[P1,P2] is equal to 0, it shows they are not co-changed in past.

In next section, we define package-changeability measures.

3.2. Package-changeability measures

a) Package change-proximity

Two packages P_a and P_b are said to be change-proximate (i.e. *PChProx*(P_a,P_b)) iff there exists a link between them. The link between P_a and P_b can be seen from the Package Proximity Matrix (PPM). So, *PChProx*(P_a,P_b) can be computed as equation 1:

$$\begin{aligned} PChProx(P_a,P_b) &= 1 \quad iff\ PPM[P_a, P_b]=1 \\ otherwise &= 0 \end{aligned} \tag{1}$$

It indicates, whether both packages P_a and P_b are change-coupled or not as per the available change-history of the packages.

b) Package change-coupling index

Package Change-coupling Index– *PChCI*(P_a) measures the extent of change-coupling of a package P_a. For each package P_a, *PChCI*(P_a) can be measured as per equation 2:

$$PChCI(P_a) = \frac{\sum_{a \neq b, b=1}^{N} PChProx(P_a, P_b)}{N} * 100 \tag{2}$$

Here, *PChProx*(P_a,P_b) is change-proximity between package P_a and P_b and N is total number of packages in the software system.

We know that in object-oriented environment a package or sub-package is basically a collection of related classes. It is very much necessary that these classes should collaborate to fulfil the objectives of the software system. Sometimes classes of a package P_a are required to be coupled with some classes of other package P_b. This reflects the change-coupling between packages P_a and P_b to fulfil the required and common functionality. The excessive coupling should not be allowed as it is the result of bad design. So, developers have to decide to what extent packages are allowed to be coupled. For this purpose, two package change-coupling-thresholds θ_1 and θ_2 can be defined or set by development team to categorize the changeability of packages.

Hence, A package is supposed to be less change-coupled if *PChCI*(P_a) is negligible(less than θ_1) and critically change-coupled if *PChCI*(P_a) is above θ_2. Moreover, the changeability of packages is also ranked accordingly as *GOOD, MODRERATE* and *BAD* according to the following rules.

if $PChCI(P_a) \leq \theta_1$	*then*	*Package-Changeability*(P_a) = *GOOD*
if $PChCI(P_a) > \theta_1$ *and* $\leq \theta_2$	*then*	*Package-Changeability*(P_a) = *MODERATE*
if $PChCI(P_a) > \theta_2$	*then*	*Package-Changeability*(P_a) = *BAD*

Packages under the *GOOD* category of changeability are very less change-coupled and they can be easily changeable. Packages under the *MODERATE* category of changeability are change-coupled but not much. These packages will require attention during changes to them because change to them can produce ripple effect. Further, packages under the *BAD* category of changeability are highly change-coupled and during changes to them these packages will require extra attention because probability of change-ripple effect is very high.

For example project i.e. *System-X*, suppose developer assumed $\theta_1=10\%$ and $\theta_2=30\%$. Then package change–coupling indices for all four packages are computed as *PChCI(P1)=75%, PChCI(P2)=0%, PChCI(P3)=75%* and *PChCI(P4)=75%.* Further, changeability of packages P1, P3 and P4 are ranked as BAD. It means changes to these packages require extra attention. It also shows that these packages are highly change-coupled. The packages should be more cohesive so they need to be restructured due to their bad design. The changeability of package P2 is ranked as *GOOD*. It shows that it is not change-coupled and can be changed easily in future.

In next sub-section, a framework for the computation of package-changeability measures is discussed.

3.3. Framework of package-changeability measurement

- *Input:*

For any software system:-
Package Change-Reports, Set of packages P_j's and Number of packages-N.

- *Output:*

Package-Proximity Matrix-*PPM*, Package Change-Proximity-*PChProx(P_a,P_b)*, Change-coupling index-*PChCI (P_a)* and *Package-Changeability(P_a).*

- *Steps:*

a) Formation of *Package-Proximity Matrix (PPM)* as per the available Change-reports of packages according to the procedure described in section 3.1.
b) Computation of *Package Change-Proximity-PChProx(P_a ,P_b)* of each pair of packages as defined in section 3.2.
c) Measurement of Package *Change-Coupling Index-PChCI(P_a)* and *Package-Changeability(P_a)* of each package as defined in section 3.2.

In next subsection, a case study and findings from the results are discussed.

4. Case Study and Result Findings

A case study has been performed by extracting the change-logs for the Egit project from SVNSearch, a subversion web based utility[21]. The proposed package-changeability measures have been computed for packages of EGit project. The available change-logs are preprocessed and package-change-reports are formed. After that, package-changeability measures are computed from the shortlisted set of package-change-reports, as described in section 3. Here, we discussed some findings drawn from our results. We assumed $\theta_1=10\%$ and $\theta_2=30\%$.

- The packages *clone* and *components* are changed together most of the times.
- The packages *{clone, components* and *internal}* and packages *{core* and *op}* are more change-coupled.
- The results show that change-coupling indices of packages *push* and *dialog* are on the lower side, which depict that these packages are very less change-coupled and any changes to these packages can be done easily.
- Further, the package *fetch* is least change-coupled with other packages. So, the changeability of this package is *GOOD*. The packages *{internal* and *repository}* and *{component* and *repository}* are more change-coupled and their changeability is also ranked as *MODERATE.*
- Most of the packages of Egit project come under *GOOD* package-changeability category.
- Some packages *core, ui, op* etc. are under *MODERATE* category which indicates that they may be providing some common services to other packages. No package is found under *BAD* package-changeability category.

5. Conclusion

In changeability analysis the change-coupling among the components need to be measured for properly maintaining the software system. Now days, during the development of complex systems; huge amount of software-engineering data are stored in the software-repositories. Software evolution data and change-logs are also important part of it. Data mining techniques can be applied manually, semi-automatically or automatically to extract the hidden facts about the quality of the software system. The extracted facts or knowledge can be used for reengineering, comprehension, change-impact analysis, fault propagation, reusability etc. In this paper, firstly we mined the package-change-reports from the available change-history. Secondly, package-proximity matrix has been constructed to know the change-coupling or dependency between the packages. Thirdly, the package-changeability measures are defined and a framework has been given for the computation of these measures. Further to analyze package-changeability measures; a case study has been done by extracting the change-logs of the project *Egit* from a web-based subversion utility i.e. svnsearch. On the basis of findings drawn from case study, we can say that mining past change-coupling or co-evolution pattern will help to effectively implement the subsequent changes. In future, we are planning to develop a change-history based reverse-engineering framework for the complex and legacy systems.

References

1. Zimmerman T, Weissgerber P, Diehl S, Zeller A. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, vol. 31(6), 2005, p. 429-445.
2. Zimmermann T, Weißgerber P. Preprocessing CVS data for fine-grained analysis. In *Proc. the Int. Workshop on Mining Software Repositories (MSR)*, Edinburgh, United Kingdom, May 2004, p. 2-6.
3. Fowler M, Beck K, Brant J, Opdyke W and Roberts D. Refactoring: Improving the design of existing code," *Boston: Addison-Wesley*, 2000.
4. Gupta V, Chhabra J K. Package coupling measurement in object-oriented software," Journal of computer science and technology, 24(2), 2009, p. 273-283.
5. Vanya A, Premraj R and Vliet H V. Resolving unwanted couplings through interactive exploration of co-evolving software entities - an experience report. *Information & Software Technology,* 54(4), 2012, p. 347-359.
6. Chidamber S R, Kemerer C F. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6), 1994, p. 476-493.
7. Chidamber S R, Kemerer C F. Towards a metrics suite for object oriented design. In *Proc. the 6th ACM Conf. Object- Oriented Programming: Systems, Languages and Applications* (*OOPSLA*), Phoenix, AZ, Oct. 6-11, 1991, p.197-211.
8. Lee Y S, Liang B S, Wu S F and Wang F J. Measuring the coupling and cohesion of an object-oriented program based on information flow. In *Proc. International Conference on Software Quality*, Maribor, Slovenia, Nov. 6-9, 1995, p. 81-90.
9. Tagoug N. Object-oriented system decomposition quality. In *Proc. the 7th IEEE International Symposium on High Assurance Systems Engineering*, Tokyo, Japan, Oct. 23-25, 2002, p. 230-235.
10. Gui G, Scott P D. Coupling and cohesion measures for evaluation of component reusability. In *Proc. International Workshop on Mining Software Repositories*, Shanghai, China, May 22-23, 2006, p. 18-21.
11. Lozano A, Wermelinger M and Nuseibeh B. Evaluating the relation between changeability decay and the characteristics of clones and methods. In *Proc. of 4th International ERCIM Workshop on Software Evolution and Evolvability*, 2008.
12. Xie T, Acharya M, Thummalapenta S and Taneja K. Improving software reliability and productivity via mining program source code. In *IEEE International Symposium on Parallel and Distributed Processing,* 2008, p. 1-5.
13. Zhong S, Khoshgoftaa T M and Seliya N. Analyzing software measurement data with clustering techniques. *IEEE Intelligent Systems,* 09(2), 2004, p. 20-27.
14. Ying A, Murphy G, Ng R, and Chu-Carroll M. Predicting source code changes by mining change history. *IEEE Transactions on Software Engineering*, 30(9), 2004, p. 574-586.
15. http://svnsearch.org/
16. http://www-03.ibm.com/software/products/en/clearcase
17. http://www.bugzilla.org/
18. Robbes R, Lanza M, and Pollet D. A benchmark for change prediction. *Faculty of Informatics, Universit Della Svizzerra Italiana, Lugano, Switzerland, Tech. Rep. 06*, 2008.
19. Beyer D and Noack A. Clustering software artifacts based on frequent common changes. In *Proc. of the 13th International Workshop on Program Comprehension*, 2005
20. Kagdi H. Mining software repositories to support software evolution. *Ph.D. dissertation, Kent State University,* 2008.
21. http://svnsearch.org/svnsearch/repos/EGIT/search