Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 82 (2016) 107 – 114

Procedia Computer Science

Symposium on Data Mining Applications, SDMA2016, 30 March 2016, Riyadh, Saudi Arabia

Defectiveness Evolution in Open Source Software Systems

Yasir Javed[a,b]*, Mamdouh Alenezi[a]

[a].*Prince Sultan University, Riyadh,KSA*
[b]*FIT, UNIMAS, SARAWAK, Malaysia*

Abstract

One of the essential objectives of the software engineering is to develop techniques and tools for high-quality software solutions that are stable and maintainable. Software managers and developers use several measures to measure and improve the quality of a software solution throughout the development process. These measures assess the quality of different software attributes, such as product size, cohesion, coupling, and complexity. Researchers and practitioners use software metrics to understand and improve software solutions and the processes used to develop them. Determining the relationship between software metrics aids in clarifying practical issues with regard to the relationship between the quality of internal and external software attributes. We conducted an empirical study on two open source systems (JEDIT and ANT) to study the defectiveness Evolution in Open Source Software Systems. The result reveals that a good designed software has lesser defects and have high cohesion. Moreover the study also revealed that defects are higher in initial versions and most corrected errors are from major classes in initial version. Removal of defects also reveals that a good software is consistently improved and feed backs are important part of open source systems.

Peer-review under responsibility of the Organizing Committee of SDMA2016
Keywords: Software Evolution; cohesion ; defectiveness evolution

1. Introduction

Software systems have become an essential component of any critical infrastructure. Software systems usually evolve constantly, which requires constant development and maintenance. As software evolves, its design and code qualities determine how costly is to develop and maintain that software. Software evolution is the vigorous activities

* Yasir Javed. Tel.: +966 11 494 8287; fax: +966 11 494 8317.
E-mail address: yjaved@psu.edu.sa

Peer-review under responsibility of the Organizing Committee of SDMA2016
doi:10.1016/j.procs.2016.04.015

of software systems while they are improved and maintained over their lifespan[10]. Software systems change and evolve throughout their life cycle to accommodate new features and to improve their quality. Software needs to evolve in order to survive for a lengthy period. The changes that software undergo lie within corrective, preventive, adaptive and perfective maintenance that lead to software evolution.

The availability of open source systems data allows us to explore different kinds of relationships. Internal characteristics and external characteristics can be investigated using several data mining techniques. The resulted insights will shed light on different decisions and endeavors while the system is being evolved over time. One of the main goals of software engineering research is to provide evidence to support and facilitate in making correct decisions during the development of the software[5]. Reaching these decisions always depends on how the data are analyzed and which information is extracted from the data during the analysis.

To understand how software quality changes as software evolve, we use both internal and external quality metrics as used by Neamtiu et al[3]. The attributes of software quality can be categorized into two main types: internal and external. Internal quality attributes can be measured using only the knowledge of the software artifacts, such as the source code, whereas the measurement of external quality attributes requires the knowledge of other factors, such as testability and maintainability. The attributes of software quality, such as defect density and failure rate, are external measures of the software product and its development process. External quality means how users' are perceiving and accepting the software. To quantify this, we use the defect density. Internal quality metrics assess internal quality, coupling, cohesion, and complexity. The more complex the software the more difficult to is to change/extend[3].

Software metrics[4] are measures utilized to evaluate the process or product quality. These metrics helps project managers to know about the progress of software and assess the quality of the various artifacts produced during development. The software analysts can check whether the requirements are verifiable or not. Software metrics are required to capture various software attributes at different phases of the software development. Software metrics can be utilized to adequately measure various phases of the software development life cycle. Software product metrics represent several aspects of the source code. They reveal a lot of design and complexity problems in the source code. Several empirical studies have established the notion that certain source code characteristics like code size, complexity, and coupling, programming language, and programming style hugely influence software maintenance efforts, costs, and effectiveness[1]. These characteristics include code size, complexity, and coupling, programming language, and programming style[2].

We conducted an empirical study on two open source systems Jedit and Ant. A theory of software evolution must be based on empirical results, verifiable and repeatable by the constant development at each phase whereas thinking (feedback) must be included from testers or end users[11].

2. Datasets of the Investigated Systems

We ran our empirical study on two open-source applications written in Java. We used several criteria to select the systems: 1) well-known systems that are used very widely; 2) sizable systems that yield realistic data; 2) actively maintained systems; 4) systems with publically available data, which is crucial in empirical studies. Apache Ant is a Java library and command-line tool whose mission is to drive processes described in build files as targets and extension points dependent upon each other. The main known usage of Ant is the build of Java applications. Ant supplies a number of built-in tasks allowing to compile, assemble, test and run Java applications. jEdit is a mature programmer's text editor supported by hundreds (including the time-developing plugins) of person-years of development. It is written in Java and runs on any operating system that supports Java, including Windows, Linux, Mac OS X, and BSD. The POI project consists of APIs that are used to manipulate various file formats based on Microsoft's OLE 2 Compound Document format, and the Office OpenXML format, which uses pure Java. Table 1 shows some characteristics about the dataset.

Table 1. Characteristics about the dataset

System	Version	LOC	Defect Density	# of Classes
ANT	1.3	37699	0.000749411	125
	1.4	54195	0.009223589	178
	1.5	87047	0.000519024	293
	1.6	113246	0.001395763	351

	1.7	208653	0.001620008	745
JEDIT	3.2	128883	0.006665815	272
	4	144803	0.0025645	306
	4.1	153087	0.003972284	312
	4.2	170683	0.000660032	367
	4.3	202363	0.000697531	492

3. Investigated Metrics

The investigated metrics are categorized as follows: coupling, cohesion, inheritance, and product size (*referring to number of classes)*. The metrics were derived from several suites of metrics. We focus on object-oriented metrics because they are accessible in the early stages of software development. The selected metrics of open source software systems are shown in Table 2 that are used to calculate coupling, cohesion, covariance and correlation for each system. Table 3 shows the descriptive statistics like Min, Max and Standard Deviation each calculated against defect density.

Table 2. An example of a table

Metric Name
Weighted methods per class (WMC)
Depth of Inheritance Tree (DIT)
Number of Children (NOC)
Coupling between object classes (CBO)
Response for a Class (RFC)
Lack of cohesion in methods (LCOM)
Lack of cohesion in methods (LCOM3)
Afferent couplings (Ca)
Efferent couplings (Ce)
Number of Public Methods (NPM)
Data Access Metric (DAM)
Measure of Aggregation (MOA)
Measure of Functional Abstraction (MFA)
Cohesion Among Methods of Class (CAM)
Inheritance Coupling (IC)
Coupling Between Methods (CBM)
Average Method Complexity (AMC)
McCabe's cyclomatic complexity (CC)
Lines of Code (LOC)

Table 3. Descriptive Statistics about the Dataset

System	**Min**	**Max**	**Std Deviation**
ANT	0	1	-0.241971707
Jedit	0	1	-0.184046838

4. Co-relation Analysis

One of the most difficult tasks to deliver in object-oriented design is to have a well-designed classes; classes that are easy to understand, easy to maintain and easy to reuse. Two main factors that influence the design of classes are coupling and cohesion. Coupling and cohesion are highly related. Bad cohesion usually leads to bad coupling because they have a highly interdependent influence[12].

Cohesion and coupling are equally important for software quality. We investigate the relationship between coupling and cohesion during the software evolutions. The objective of this step of our experiments is attempting to explain the relationship between coupling and cohesion. To test the hypothesis, if cohesion is correlated with coupling, we considered the two systems against each version and their overall combined effect for cohesion and coupling respectively.

Table 4. Covariance and Correlation between Coupling and Cohesion

System	Version	Covariance	Correlation
ANT	1.3	-0.178317575	-0.178317575
	1.4	-0.267708987	-0.267708987
	1.5	-0.226207766	-0.226207766
	1.6	-0.223399833	-0.223399833
	1.7	-0.245364893	-0.245364893
	Comb	0.067013728	-0.241971707
JEDIT	3.2	-0.086502465	-0.121641126
	4	-0.11483904	-0.143109866
	4.1	-0.131455534	-0.171383546
	4.2	-0.134324123	-0.17423805
	4.3	-0.20873689	-0.254397444
	comb	-0.143966153	-0.184046838

In the results shown in Table 4, we can see very interesting facts. The negative covariance and correlation between coupling and cohesion in Ant have increased over time. This tells us that the design of the system has improved over that period especially, after version 1.3. In case of jEdit, the covariance and correlation have a steady increase which also tells us that the design has improved while the system is evolving. Looking at these numbers shows an evident relationship between coupling and cohesion. If the negative correlation is increasing over time, this means that they are inversely proportional to each other and the design is being improved.

5. Defect Density Classification

Defect density is one of the most established measures of software quality[9]. Defect density consists of post-release defects per thousand lines of a delivered code[9]. This definition is used mainly among practitioners to calculate and evaluate the quality of their projects at a certain phase of development. Defect density is used to measure the quality of the software product. It indicates the improvements in the quality of the successive releases of certain software. The lower the number of defect densities, the better the software quality is.

We have classified the classes of Jedit and Ant into three categories in terms of defect density. We evaluate the classification based on Precision, Recall, F-measure, and Area under Curve (AUC) or ROC (Receiver Operating Characteristics) as[13] argues that AUC is the best measure to report the classification accuracy. Precision measures how many of the vulnerable instances returned by a model are actually vulnerable. The higher the precision is, the fewer false positives exist. Recall measures how many of the vulnerable instances are actually returned by a model. The higher the recall is, the fewer false negatives exist. F-Measure is the harmonic mean of Precision and Recall.

In Table 5, we show the classification results of defect density. We have combined all the dataset and classified the classes into three categories in terms of defect density. We then ran the classification of Random Forrest using 10-cross validation. The results show that using these metrics, it is very feasible to predict the defect density of these classes. In other words, these internal measures can help software engineers forecast the defectiveness of developed classes before deploying them.

Table 5. Classification Results

System	Precision	Recall	F-Measure	ROC Area
Jedit	0.726	0.785	0.75	0.756
ANT	0.981	0.981	0.981	0.999

Another experiment would be figuring out which ones of these metrics are more helpful in finding defective classes. It is very important to understand, which metrics are the most influential metrics in determining the defect density of a particular class. This will examine how well each metric can individually differentiate classes as defective or not. We have used the well-known Chi-Square (X2) feature selection algorithm. The Chi-Square (X2) test is used to examine independence of two events. The events, X and Y, are assumed to be independent if P(XY) = P(X)P(Y). In term selection, the two events are the occurrence of the term and the occurrence of the class.

Table 6 shows the results of the influential metrics. There are a lot of similarities between the two sets of metrics. In fact, RFC, LOC, and WMC appeared in both columns. These results can be justified very easily from two perspectives. The first perspective is that LOC is actually a component in defect density. The other perspective is that these metrics usually positively correlate with each other by looking at the literature. What can we learn from this experiment is that by looking at only these metrics, software engineers can decide about the defectiveness of the developed classes.

Table 6. The Influential Metrics using the Chi-Square (X2)

Rank	ANT	Jedit
1	RFC	LOC
2	LOC	RFC
3	WMC	CAM
4	Ce	WMC

6. Defect Density Evolutions

In this experiment, we show how defect density evolves in these two open source systems. Figure 1(a) shows the evolution of defect density in the Ant system. The defect density started very low in version 1.3. Then, there has been an increase of defect density in version 1.4. Later versions have seen a big improvement in terms of defect density. This is very common in software systems. Some software releases are impacted by a boom in the new delivered features which will be accompanied by a big number of defects. Restructuring and testing the software will usually result in improved defectiveness.

Figure 1(b) shows the evolution of defect density in the jEdit system. Form the figure; we can see that the defectiveness of jEdit has decreased overtime. This shows that jEdit is improving overtime and the design is improving and a lot of these defects have been resolved. jEdit shows a very clear pattern with regards to defect density.

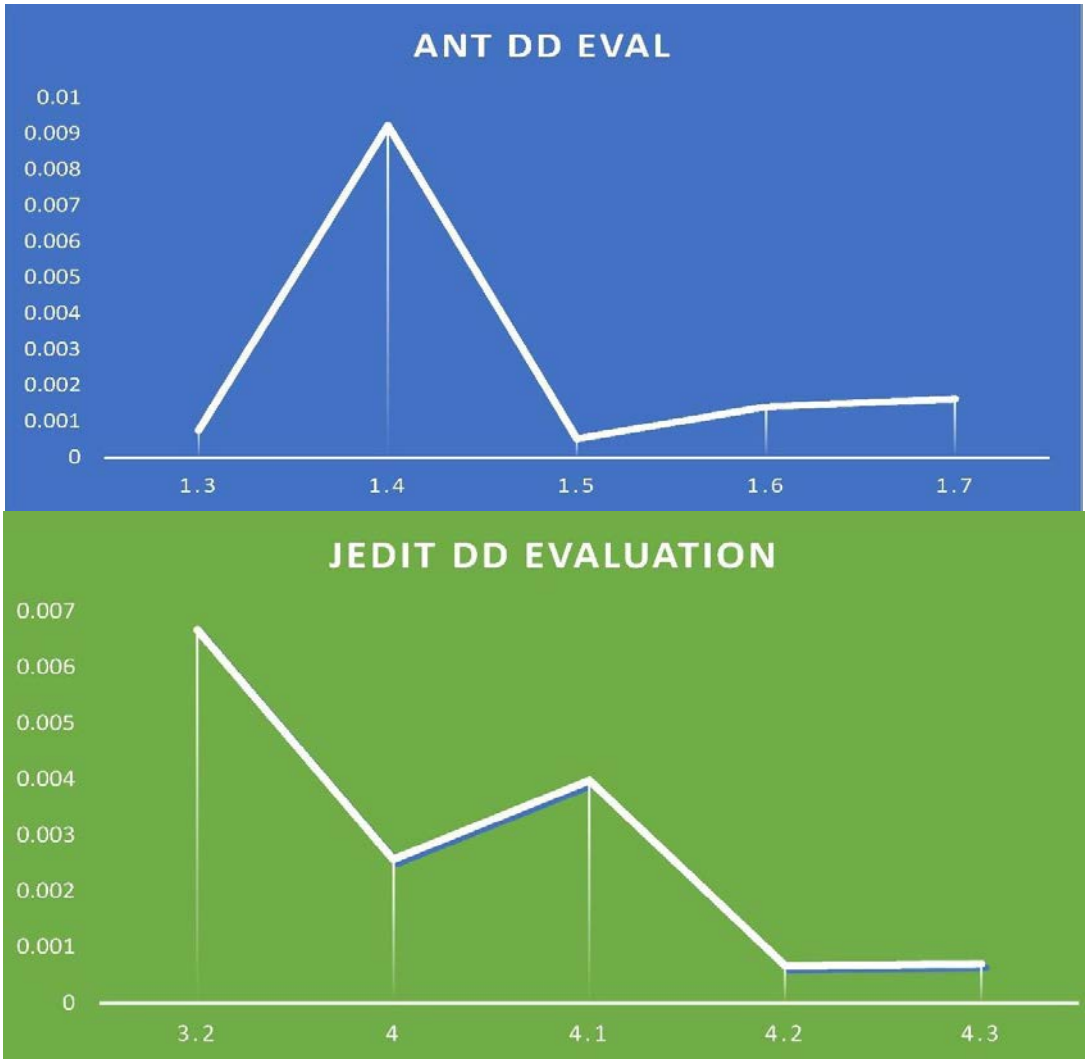

Fig. 1. (B) Defect Density Evolution in Ant; (G) Defect Density Evolution in jEdit.

Figure 2 shows the relationship between the defect density evolution and the covariance between coupling and cohesion. It is very clear from the two systems that when the covariance increases (good design) the defect density decreases. This is very intuitive since in software engineering it is very known that improving the software design will yield a more quality product (less defective).

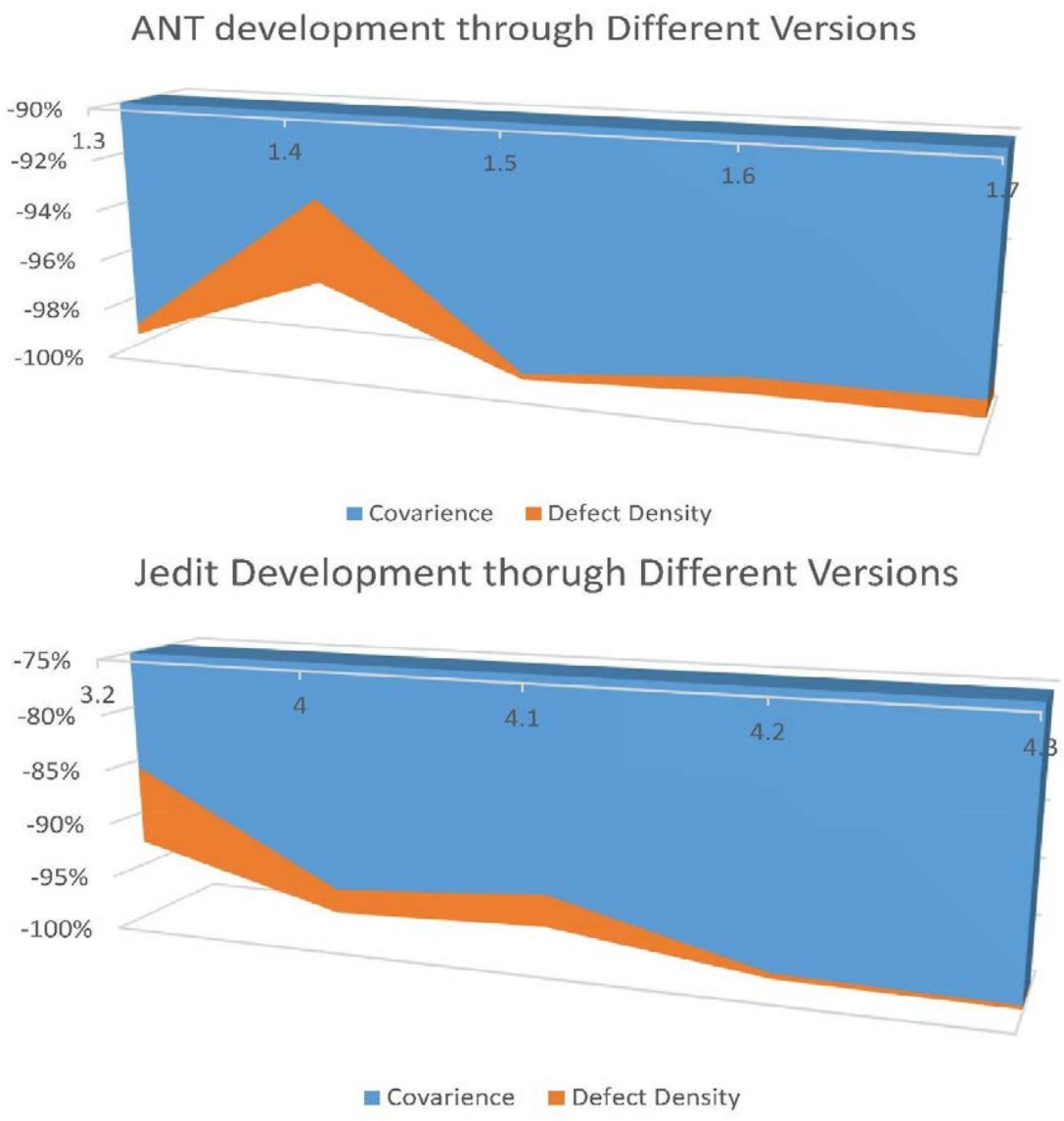

Fig. 2. The relationship between defect density and coupling and cohesion covariance

7. Related Works

There are a lot of studies that tried to find relationships between software metrics and quality characteristics of software systems. Jabangwe et al.14 conducted a systematic literature review in finding the empirical evidence on the link between object oriented metrics and external quality attributes. Their results suggested that complexity, cohesion, size and coupling measures have a better link with reliability and maintainability than inheritance measures. Neamtiu et al.15 applied Lehman's laws of evolution on several open source systems in order enhance our understanding of open source software evolution. Alenezi and Abunadi16 studied the quality of open source systems from product metrics perspective. They studied defect density in open source systems. They found that defect density

is relevant to different developers and different product sizes. Furthermore, they have found that open source project has shown to have low defect density and the larger the product the lower the defect density is.

8. Conclusion

Building software that is of high quality is an essential aim for software engineering practitioners. To measure quality of software, different metrics are used and are available especially in open source software projects. This study allows the selection of open-source software's to be made on basis of design and defects. Our study reveals that a good designed software have high cohesion and less number of defects. Our study also reveals that initial version of open source software projects have higher defects (as seen in Figure 1 and 2)and it decreases with new version that also shows that feedback on open source software projects is an important aspect for improving cohesion and decreasing defects. The study also reveals that development of open source software's is done modular in which major classes are corrected in initial version while minor are corrected in later versions.

References

1. Ware, M. P., F. George Wilkie, and Mary Shapcott. "The application of product measures in directing software maintenance activity." Journal of Software Maintenance and Evolution: Research and Practice 19.2 (2007): 133-154.
2. Jones, Capers. "Geriatric issues of aging software." CrossTalk 20.12 (2007): 4-8.
3. Neamtiu, Iulian, Guowu Xie, and Jianbo Chen. "Towards a better understanding of software evolution: an empirical study on open-source software." Journal of Software: Evolution and Process 25.3 (2013): 193-218.
4. Malhotra, Ruchika. Empirical Research in Software Engineering: Concepts, Analysis, and Applications. CRC Press, 2015.
5. Fenton, Norman, and James Bieman. Software metrics: a rigorous and practical approach. CRC Press, 2014.
6. Shah, Syed Muhammad Ali, and Maurizio Morisio. "Complexity Metrics Significance for Defects: An Empirical View." Proceedings of the 2012 International Conference on Information Technology and Software Engineering. Springer Berlin Heidelberg, 2013.
7. Alenezi, Mamdouh, and Kenneth Magel. "Empirical evaluation of a new coupling metric: Combining structural and semantic coupling,"." International Journal of Computers and Applications 36.1 (2014).
8. Bettenburg, Nicolas, Meiyappan Nagappan, and Ahmed E. Hassan. "Towards improving statistical modeling of software engineering data: think locally, act globally!." Empirical Software Engineering 20.2 (2015): 294-335.
9. Shah, Syed Muhammad Ali, Maurizio Morisio, and Marco Torchiano. "Software defect density variants: A proposal." Emerging Trends in Software Metrics (WETSoM), 2013 4th International Workshop on. IEEE, 2013.
10. Alenezi, Mamdouh, and Fakhry Khellah. "Evolution Impact on Architecture Stability in Open-Source Projects." International Journal of Cloud Applications and Computing (IJCAC) 5.4 (2015): 24-35.
11. Godfrey, Michael W., and Daniel M. German. "On the evolution of Lehman's Laws." Journal of Software: Evolution and Process 26.7 (2014): 613-619.
12. Larman, Craig. Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development. Pearson Education India, 2005.
13. Czibula, Gabriela, Zsuzsanna Marian, and Istvan Gergely Czibula. "Software defect prediction using relational association rule mining." Information Sciences 264 (2014): 260-278.
14. Jabangwe, R., Börstler, J., Šmite, D., & Wohlin, C. (2015). Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review. Empirical Software Engineering, 20(3), 640-693.
15. Neamtiu, I., Xie, G., & Chen, J. (2013). Towards a better understanding of software evolution: an empirical study on open-source software. Journal of Software: Evolution and Process, 25(3), 193-218.
16. Alenezi, M., & Abunadi, I. (2015). Quality of Open Source Systems from Product Metrics Perspective. International Journal of Computer Science Issues (IJCSI), 12(5), 143.