Architectural Stability Evolution in Open-Source Systems

Mamdouh Alenezi
College of Computer & Information Sciences
Prince Sultan University
Riyadh 11586, Saudi Arabia
malenezi@psu.edu.sa

Fakhry Khellah
College of Computer & Information Sciences
Prince Sultan University
Riyadh 11586, Saudi Arabia
fkhellah@psu.edu.sa

ABSTRACT

Open-source software systems are becoming progressively vital these days. Since open-source softwares are usually developed in a different management style, the quality of their architectures needs to be studied. ISO/IEC SQuaRe quality standard characterized stability as one of the sub-characteristics of maintainability. Unstable software architecture could cause the software to require high maintenance cost and effort. Almost all stability related studies target the package level. To our knowledge, there has been no proposed work in literature that addresses the stability at the system architecture level.

In this work, we propose a simple, yet efficient, technique that is based on carefully aggregating the package level stability in order to measure the change in the architecture level stability as the architecture evolution happens. The proposed method can be used to further study the cause behind the positive or negative architecture stability changes.

CCS Concepts

•**Software and its engineering** → **Software system structures; Software evolution;** •**General and reference** → *Empirical studies; Measurement;*

Keywords

Software Quality; Software Evolution; Stability; Software Metrics; Open Source; Package Structure.

1. INTRODUCTION

Software evolution is the vigorous activities of software systems while they are improved and maintained over their lifespans [11, 6, 1]. 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.

ICEMIS '15, September 24-26, 2015, Istanbul, Turkey
© 2015 ACM. ISBN 978-1-4503-3418-1/15/09…$15.00
DOI: http://dx.doi.org/10.1145/2832987.2833014

A major characteristic of software evolution is architecture evolution. While a specific system is evolving, its architecture is affected. In opposition, having a plan for how an architecture should evolve is a powerful mechanism to plan and guide software evolution.

Software Architecture is defined in the IEEE standards [8] as "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution". One desired quality of the software architecture is stability. Stability is one of the maintainability characteristics of the ISO/IEC SQuaRe quality standard [7]. According to this standard, stability is defined as the degree to which the software product can avoid unexpected effects from modifications of the software. [7].

Abundant research studies addressed the software evolution of open-source systems, with more than one hundred research papers referenced in recent systematic literature reviews [2, 15]. Although there are abundant research studies that investigated the evolution of these softwares, very little effort targeted the architecture in general and no work addressed the architectural stability of these systems. Almost all stability related studies target the package level stability by using various metrics. In this work, we concentrate on the most frequently used metric for assessing the package stability which is defined as [13]:

$$\text{Instability}I = \frac{C_e}{(C_e + C_a)} \quad (1)$$

The I metric for a package is defined as the ratio of efferent coupling C_e to total coupling $C_e + C_a$ for the package. The C_a denotes the number of other packages that depend upon classes within the package (*fan-in*). It measures the incoming dependencies. The C_e denotes the number of other packages that the classes in the package depend upon (*fan-out*). It measures and counts the outgoing dependencies. The I metric indicates how a flexible a package is to change. The metric ranges from zero to one, one indicates a completely unstable package whereas zero indicates a completely stable package. Martin [13] suggests that some packages are easier to change than others. Easy to change packages should depend on less easy to change packages. Depending on packages make a package less stable, as changes affecting depended-upon packages propagate to the depending package. Having other packages depend on a certain package make that package more stable, as more effort is needed to merge changes with all dependent packages. The I metric shows how easy a package is to change.

Due to system evolution, the overall system stability is affected. This is due to mainly changes in the currently existing packages and the instability incurred due to the added new packages. It is very important for system architect to monitor the overall system instability due to various maintenance task such as adding and removing packages, hence, changing the packages dependency relationships. However, calculating an aggregate value that shows the overall system stability due to evolution has not been addressed in literature.

The main contribution of this work is two-fold. First, proposing a simple, yet efficient, technique that is based on carefully aggregating the package level stability in order to measure the change in the architecture level stability as the architecture evolution happens. Second, empirically applying the proposed method on two open source systems.

The remainder of this paper is organized as follows. Section 2 presents the used methodology. The experimental evaluation is given in Section 3. Some threats to validity are presented in Section 4. Section 5 discusses related work. Conclusions of the research are presented in Section 6.

2. APPROACH

In order to monitor the stability change of a given system as new releases are developed, the stabilities of two consecutive releases should be compared. This would give more insight and clarify if the change is positive or negative and if the current system instability is dominated by the effect of the newly added packages or due to changes in the currently existing packages or to both.

Suppose that the instabilities of two consecutive system releases $v-1$ and v are computed using (1) and given by I_{v-1} and I_v, then the average amount of change in the system stability ΔI is obtained as:

$$\Delta I = \frac{1}{K} \sum_{p} (I_{v-1} - I_v)_p \quad (2)$$

Where p is the package and (v) is the release and K is the number of *common packages* in the two consecutive releases. A straight forward approach to compute the Aggregate System Instability (ASI) of a certain release (v) of a system can be done by averaging over all release(v) packages instabilities (I) computed using (1).

$$\text{ASI(v)} = \frac{1}{N} \sum_{p} I_p \quad (3)$$

Where N is the number of packages in release (v), and I_p is the instability for package p.

However, due to the average mixing effect, computing the system stability change ΔI using ASI(v) plainly as given in Equation 3 is misleading and will lead to incorrect interpretations and, hence, decisions. To illustrate the drawback of using the average to compute the overall system stability, consider the toy system given in Table 1. The system has two releases where the first release has two packages P_1,P_2 and a new package P_3 was added in the second release. The table shows the computed instabilities for all the packages for the two consecutive releases. In the second release there has been improvement in the instabilities of packages P_1 and P_2. However, the instability for the newly added package P_3 is high (0.7) which would naturally cancels the obtained improvement. By looking at the ASI values for the two releases, one would see that the second release has better stability ($I = 0.63$) in comparison to the previous release ($I = 0.85$). In addition, the instability change ΔI is (-0.22) as computed using (2). One would incorrectly conclude that there has been improvement in the system stability in the second release as the change is negative. This incorrect conclusion is due to the mixing effect of the averaging used to compute ASI for both releases.

Our approach to reflect the instability change due to evolution is based on expressing the aggregate system instability change for a certain release v as being composed of the average of two main components: the change due to updates in the common packages of the two consecutive releases ΔI given in (2) and the ASI for the current release *but* computed only for the *newly* added packages. This would be expressed as:

$$\text{Aggregate System Instability change (v)} = \frac{(\Delta I + ASI(v))}{2} \quad (4)$$

Where ΔI is the aggregate overall change of instability for all common packages in the two consecutive releases (i.e., $v-1$ and v) that have a change in their instability metric due to the evolution, and ASI(v) is computed as in 3, where N would be the number of newly added packages only. The range of the aggregate system instability change is between -1 and 1.

To clarify the approach, assume that the lists of packages of two consecutive releases $(v-1)$, and (v) of a certain system are $\{P_1, P_2, \ldots, P_n\}$, and $\{P_1, P_2, \ldots, P_n, P_{n+1}, P_{n+2}\}$, respectively. Assume that in release (v), two new packages were added: P_{n+1}, P_{n+2}. Let us also assume that instability changes happen in only L common packages, (i.e., the computed change $(I_{v-1} - I_v)_p \neq 0$). Then, in order to compute the aggregate system instability change for release (v), as given in (4), we start by computing ΔI using (2) with $K = L$. ASI(v) is then computed using (3) with $N = 2$, since only two new packages are added to release (v).

To illustrate how such approach correctly reflects the stability change, we refer back to the toy system given in Table 1. The corrected stability change is computed using (4) as 0.23. This tells us that there has been reduction in the stability in the second release not improvement. This does make more sense as the computed instability for the newly added package P_3 (0.7) exceeds the improvement happened to the two existing packages: P_1 and P_2, (i.e., -0.5 in total).

3. EXPERIMENTAL EVALUATION

Table 2: Selected Software Systems

System	Versions	LOC	Packages
jEdit	3.0.0-4.2.0	43382-94656	11-16
PDFBox	1.5.0-1.8.7	102242-133959	26-31

In order to evaluate the proposed approach, we select open-source applications implemented in Java. To select the systems for the empirical analysis, three selection criteria have been used.

- The selected systems had to be well-known systems that are very widely used.

Package	I_1	I_2
P_1	0.7	0.5
P_2	1	0.7
P_3(new release 2)only	-	0.7
ASI	**0.85**	**0.63**
ASI Based Instability Change		-0.22
Proposed Instability Change		0.23

Table 1: Toy System Example

- The systems had to be sizable, so the systems can be realistic and have multi-developers.
- The systems had to be actively maintained.

Characteristics of the selected software systems are listed in Table 2. jEdit6 is a medium-sized, text editor. It focuses on providing different features for developers, including macro scripting, syntax highlighting, and a comprehensive plug-in environment. We have collected 12 versions from jEdit which represent 4 years of evolution and development. Table 3 shows the evolution behavior of the jEdit system. The size of jEdit started at 43382 LOC and stopped at 94656 LOC. The number of packages started at 11 and stopped at 16. This continuous increase of the number of package shows that new features are actually reflected in increasing the number of packages. It also shows that no restructuring has happened in these releases. Usually restructuring results in decreasing the number of packages. The C_e metric average shows that even though there is increase in the number of packages, the responsibilities of these packages are increasing.

PDFBox is a Java open-source tool that can ease working with PDF files. Through the tool, one can create, manipulate, extract content in PDF documents. We have collected 12 versions from PDFBox which represent 3 years of evolution and development. Table 4 shows the evolution behavior of the PDFBox system. The size of PDFBox started at 102242 LOC and stopped at 133959 LOC. The number of packages started at 31 and stopped at 26 which means that system has restructured to reduce the number of packages. The C_e metric average shows that even though there is increase in the number of packages, the responsibilities of these packages are increasing.

Figure 1 shows the architectural stability evolution of the selected systems. For each system, we report ASI, The instability change ΔI based on ASI only, and the proposed aggregate system instability change. The goal to show how these measures actually reveal the difference in stability between releases. For both systems, looking at Figures 1(a,d), it is established that average values of instabilities do not give any insight about the effect of evolution on the system stability.

In addition, the computed ΔI based on ASI only, depicted in Figures 1(b,e), incorrectly reflect the actual change in the instabilities. To establish the validity of the above, we will correlate the computed change in instability to the actual changes in the instability due to evolution. In case of jEdit system, the correct change in the stability for the fifth releases, as computed by the proposed method Figure 1(c), is (0.49). On the other hand, the computed change based on averaging alone is almost negligible (0.025). To validate the correctness of the obtained insatiability based on the proposed method, we investigate the actual change happened to the system in the fifth release in comparison with the previous release. In the fifth release new package with $I = 1$ was added. The computed instability change based on ASI alone (0.025) does not reflect such added instability due to the new package.

The strength of the proposed method in computing the instability can be clearly seen in case of the PDFBox system. The system started by 31 packages. In the third release number of packages was reduced to 26. The computed stability change based on averaging alone Figure 1(e) shows that stability improves in the third release (the computed change is negative (-0.075)) then stabilizes. This actually is incorrect as in the third release new package was added with $I = 0.33$. In addition, the average ΔI for the remaining packages is positive (0.04)). This means that there is degradation in the instability due to evolution not improvement as mistakenly computed by averaging alone. This supports the correctness of the computed instability (0.19) based on the proposed method Figure 1(f).

4. THREATS TO VALIDITY

Threat to validity is very common in empirical studies. The validity of the results obtained in this work is constrained by a number of aspects. We have conducted the study on only two Java-based open-source systems which limits the generalization of the results. We can generalize the results to same-size Java-based systems that are open-source. We have selected two systems with different sizes in order to see the effect of the size on our findings. These two systems were studied before in the literature. Extracting the data from the source code was another threat to the validity. We have developed our own R script to collect these metrics from the source code. In developing this script, we have validated the results in parts of these two systems manually to make sure that we are getting the right results. We have studied the system level architecture on our study. Since there is no clear idea about the relationship between the behavior observed at the system level and the behavior observed at the subsystem level.

5. RELATED WORK

The case of open-source software is an instance of an emerging domain that has been disregarded in the description of the evolution laws. On the time when these laws were developed, software systems were developed by co-located teams that use face-to-face communication. In open source systems, software usually developed by distributed teams, and communicating just through electronic mechanisms. This new type of development is very different from the accepted software engineering practices during the times

Table 3: jEdit Average Metrics for each Version

Metric	1	2	3	4	5	6	7	8	9	10	11	12
LOC	43382	43655	43658	45706	51325	51376	50951	66347	66391	66398	72870	94656
Packages	11	11	11	11	12	12	12	14	14	14	15	16
Ce	4.1	4.1	4.1	4.1	4.15	4.15	4.15	4.65	4.65	4.65	4.8	5.19
I	0.541	0.541	0.541	0.541	0.566	0.566	0.566	0.586	0.586	0.586	0.584	0.580

Table 4: PDFBox Average Metrics for each Version

Metric	1	2	3	4	5	6	7	8	9	10	11	12
LOC	102242	106185	114282	114895	119682	119955	120726	124762	125327	127904	131068	133959
Packages	31	31	26	26	26	26	26	26	26	26	26	26
Ce	6	6.129	6.307	6.307	6.5	6.5	6.5	6.538	6.5	6.538	6.576	6.615
I	0.568	0.568	0.492	0.492	0.489	0.489	0.489	0.490	0.489	0.490	0.490	0.490

of the first versions of the software evolution laws. Koch established several dissimilarities in the evolution of open-source software projects of different sizes [9]. He found that small open-source software projects fulfill to some extent some of these laws. However, large software projects do not follow them at all since these projects have unbalanced workload among participants and a large number of participants.

D'Ambros and Lanza [4] proposed evolution radar to understand the package coupling. They used a visualization mechanism that visualizes classes for architecture recovery using two measures: package as group criterion and invocation number for the distance. They used the package level visualization to enable software engineers to visualize the software as connected packages. Ducasse et al. [5] introduced a visualization technique, which can be utilized to understand, analyze, and visualize the relationships of packages. They argued that relationships between packages and their contained classes are key aspects in the decomposition of an application. They suggested that it is necessary, for the reengineering and development of object-oriented systems, to recognize and investigate both sets of classes and packages. Wilhelm and Diehl [16] used Martin's [13] and size metrics to build a tool that helps to control package dependencies. Lungu et al. [12] developed a tool called Softwarenaut that recover several architecture views from the source code.

Capiluppi and Boldyreff [3] proposed a coupling-based approach to to indicate potentially reusable parts of projects, which could be reused as independent projects. Their approach was based on the instability metric [13]. They showed that low instability modules (i.e. stable modules) are good candidates to be turned into independent, external modules.

Mens et al. [14] provided guidelines in which how metrics have been, and can be, used to analyze software evolution. They contended that metrics can provide a great support to study software evolution. In order to support a reflective study, several metrics can be utilized to comprehend and appreciate the evolution quality of a software system by examining its successive releases. More specifically, metrics can be utilized to measure if the quality of a software has improved or degraded between two releases. Lee et al. [10] argued that software metrics can be utilized to evaluate and judge the quality of evolution of open source systems. The authors examined the evolution of an open source software system with regards to size, coupling and cohesion, and change quality.

6. CONCLUSION

In this work, we proposed an efficient technique to study the software architecture stability evolution. The technique is based on carefully aggregating the package level stability to measure the change in the stability of the architecture as the evolution happens. The proposed technique can be used to further study the cause behind the positive or negative architecture stability changes. The new proposed approach clearly showed the variance in the instability between releases. Future research directions include correlating the results of the proposed metric to other instability related metrics. In addition, we will use the method as performance indicator to predict if the proposed changes on various architecture levels will have negative or positive impact on the overall system architecture. We will study, the usage of the proposed method to investigate the relationship between packages stability and number of faults in them.

7. REFERENCES

[1] M. Alenezi and K. Almustafa. Empirical analysis of the complexity evolution in open-source software systems. *International Journal of Hybrid Information Technology*, 8(2):257–266, 2015.

[2] H. P. Breivold, M. A. Chauhan, and M. A. Babar. A systematic review of studies of open source software evolution. In *17th Asia Pacific Software Engineering Conference (APSEC), 2010*, pages 356–365. IEEE, 2010.

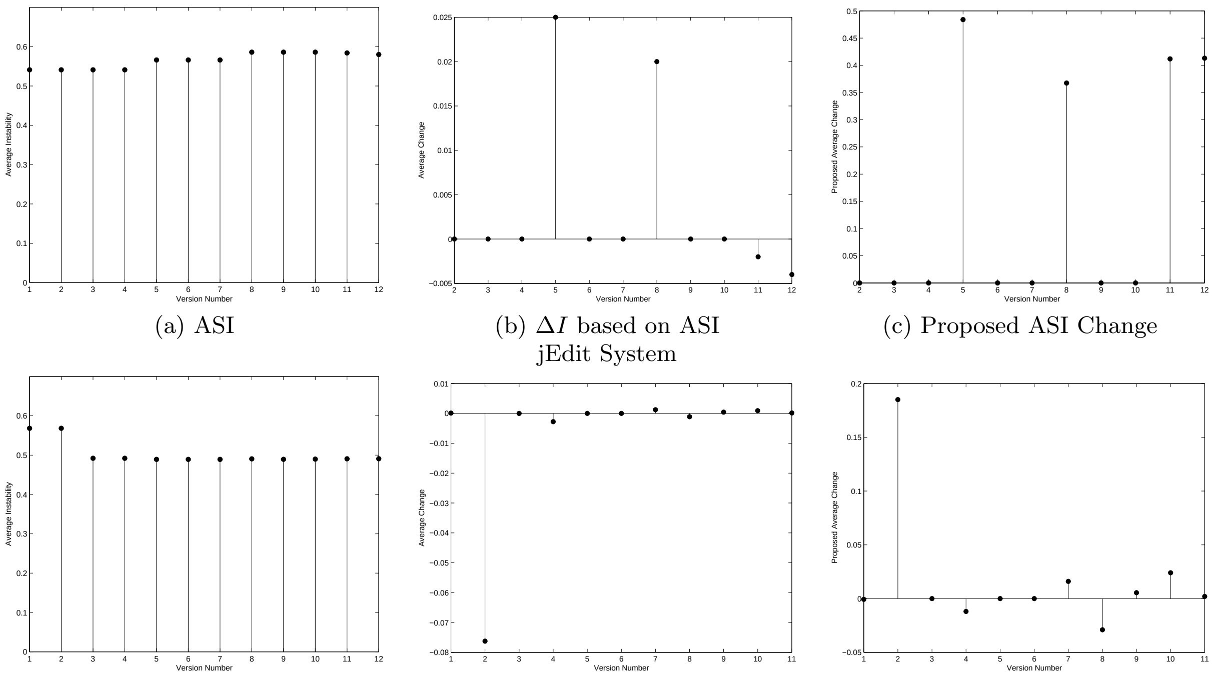
[3] A. Capiluppi and C. Boldyreff. Coupling patterns in the effective reuse of open source software. In *First International Workshop on Emerging Trends in FLOSS Research and Development, 2007. FLOSS'07.*, pages 9–9. IEEE, 2007.

[4] M. D'Ambros and M. Lanza. Reverse engineering with logical coupling. In *13th Working Conference on Reverse Engineering, 2006. WCRE'06.*, pages 189–198. IEEE, 2006.

[5] S. Ducasse, M. Lanza, and L. Ponisio. Butterflies: A visual approach to characterize packages. In *11th IEEE International Symposium on Software Metrics*, pages 10–pp. IEEE, 2005.

[6] M. W. Godfrey and D. M. German. On the evolution of lehman's laws. *Journal of Software: Evolution and Process*, 2013.

[7] ISO/IEC. Systems and software engineering - systems and software quality requirements and evaluation (square). *ISO/IEC 25010 - System and software quality models*, 2011.



(a) ASI (b) ΔI based on ASI (c) Proposed ASI Change

jEdit System

(d) ASI (e) ΔI based on ASI (f) Proposed ASI Change

PDFBox System

Figure 1: Demonstration of the Proposed method for computing Instability change where figures (left to right) in each row represent: Aggregate System Instability (ASI), ΔI based on ASI, Proposed ASI Change.

[8] ISO/IEC/IEEE. Systems and software engineering – architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pages 1 –46, 1 2011.

[9] S. Koch. Software evolution in open source projectsŮa large-scale investigation. *Journal of Software Maintenance and Evolution: Research and Practice*, 19(6):361–382, 2007.

[10] Y. Lee, J. Yang, and K. H. Chang. Metrics and evolution in open source software. In *Seventh International Conference on Quality Software, 2007. QSIC'07.*, pages 191–197. IEEE, 2007.

[11] M. M. Lehman. On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software*, 1:213–221, 1980.

[12] M. Lungu, M. Lanza, and O. Nierstrasz. Evolutionary and collaborative software architecture recovery with softwarenaut. *Science of Computer Programming*, 79:204–223, 2014.

[13] R. C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003.

[14] T. Mens and S. Demeyer. Future trends in software evolution metrics. In *Proceedings of the 4th international workshop on Principles of software evolution*, pages 83–86. ACM, 2001.

[15] M. Syeed, I. Hammouda, and T. Syatä. Evolution of open source software projects: A systematic literature review. *Journal of Software*, 8(11):2815–2829, 2013.

[16] M. Wilhelm and S. Diehl. Dependencyviewer-a tool for visualizing package design quality metrics. In *VISSOFT*, pages 125–126, 2005.