# Evolution Impact on Architecture Stability in Open-Source Projects

**Abstract**

*Software systems usually evolve constantly, which requires constant development and maintenance. Subsequently, the architecture of these systems tends to degrade with time. Therefore, stability is a key measure for evaluating an architecture. 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.*

## 1 Introduction

Software evolution is the vigorous activities of software systems while they are improved and maintained over their lifespans (Lehman, 1980; Godfrey & German, 2013; Alenezi & Almustafa, 2015). 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. 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 (ISO/IEC/IEEE, 2011) 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 (ISO/IEC, 2011). According to this standard, stability is defined as the degree to which the software product can avoid unexpected effects from modifications of the software. (ISO/IEC, 2011).

Abundant research studies addressed the software evolution of open-source systems, with more than one hundred research papers referenced in recent systematic literature reviews

(Breivold, Chauhan, & Babar, 2010; Syeed, Hammouda, & Syatä, 2013). 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 (Martin, 2003):

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

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 (Martin, 2003) 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.

Empirical studies frequently rely on data sets. Since open source software projects are often developed by volunteers with no formalized development methods (Gyimothy, Ferenc, & Siket, 2005), their process and resource data are often incomplete and unreliable. Consequently, to conduct and validate the empirical work of open source projects, product metrics only are exploited. By using open-source systems data, researchers will be able to assess and examine different hypotheses about the applicability of different software engineering methods. Open-source systems have been the source for most empirical studies since all their development data are publicly available.

The remainder of this paper is organized as follows. Section 2 presents some background about architecture stability evolution. Section 3 presents the used methodology. The experimental evaluation is given in Section 4. Some threats to validity are presented in Section 5. Section 6 discusses related work. Conclusions of the research are presented in Section 7.

## 2 Architecture Stability Evolution

Software requirements are normally volatile which entails the fact that are likely change and evolve over time. The changes are inevitable since they reflect stakeholders needs and the environment in which the software system operates.

When the business requires the system to be used for a long-term, the system must evolve to respond to business changes and future requirements. That system should also create a revenue for future prospects. In order to sustain that stability of the software becomes an essential objective of the system, especially, the architecture stability.

Architectural stability is a quality, which means to what degree a software architecture is flexible enough to tolerate evolutionary changes in business requirements, environment, and needs while maintaining a stable architecture.

When maintaining and evolving a software systems, substantial effort is dedicated toward the architecture as a key artifact. Failing to accommodate the change ultimately leads to the degradation of the usefulness of the system. Henceforth, there is a persistent necessity for flexible software architectures. A software architecture that lacks flexibility might cause disruptive changes for the requirements to be accommodated. These changes may break, disturb, and change the architectural structure, topology, style, or even the underlying architectural infrastructure, which will incur expensive and difficult changes as requirements evolve (Bahsoon & Emmerich, 2003).

To develop an architecture that is stable in the presence of changes and flexible enough to be modified and revised to the changing requirements is one of the essential challenges in software engineering. A stable architecture is able to add value to the company as the system evolve to accommodate new changes. By evaluating and valuing the flexibility of a software architecture, we aim at guiding architects a useful method to reason about the stability of their architecture (Bahsoon & Emmerich, 2004).

Stability is an architectural quality with strategic significance and long-term strategic and operational benefits. Architectural stability may result in benefits of strategic significance (instantiate from the architecture new market products; the flexibility to respond to competitive changing market; and the ability to accommodate new services). It may furthermore provide long-term operational benefits (reduced maintenance cost).

The degree of reuse of software modules is subject to different characteristics of software artifacts. One of these main characteristics is the stability of the software architecture. If the architecture remains stable while evolving, it means that the architecture contribute to implement the main functionality of the software system and can also be considered good candidate for being reused in software systems implementing similar functionality (Aversano, Molfetta, & Tortorella, 2013).

Evaluating the architecture stability targets understanding the impact of evolution on the software architecture. Software architecture is an important concern in the object-oriented community, since the software architecture comprises information that ease the communication between developers and help in developing better quality software. The literature (Jazayeri, 2002; Kpodjedo, Ricca, Galinier, & Antoniol, 2009) suggests that stability as a key measure for evaluating an architecture and, thus, several approaches were proposed to analyze the evolution of software architectures (Kimelman, Kimelman, Mandelin, & Yellin, 2010; Hassaine, Guéhéneuc, Hamel, & Antoniol, 2012). Ongoing research on software architecture has considered the architectural stability problem as an open research challenge and difficult to handle (Bahsoon & Emmerich, 2006).

| 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**

## 3 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 $\Delta I$ is obtained as:

$$\Delta I = \frac{1}{K}\sum_p (I_{v-1} - I_v)_p \tag{2}$$

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).

$$\text{ASI(v)} = \frac{1}{N}\sum_p I_p \tag{3}$$

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 $\Delta I$ 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 $\Delta I$ 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 $\Delta I$ given in (2) and the ASI for the current release *but* computed only for the *newly* added packages. This would be expressed as:

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

Where $\Delta I$ 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 $\Delta I$ 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).

## 4 Experimental Evaluation

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

**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 |

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

**Table 3. jEdit Average Metrics for each Version**

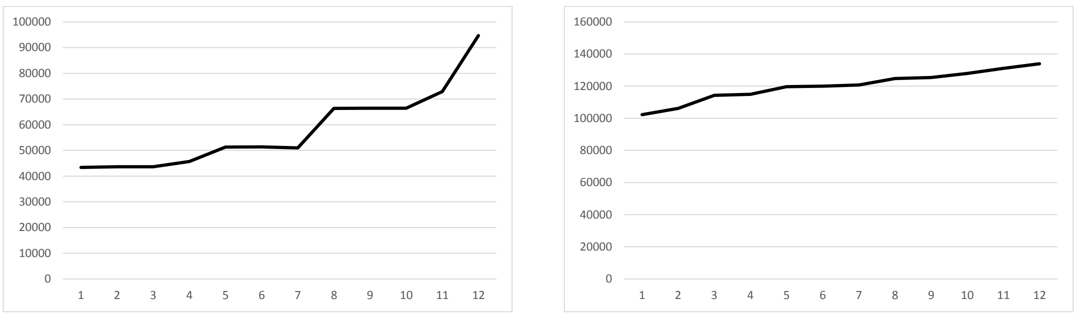
| Metric | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LOC | 43382 | 43655 | 43658 | 45706 | 51325 | 51376 | 50951 | 66347 | 66391 | 66398 | 72870 | 94656 |
| Packages | 11 | 11 | 11 | 11 | 12 | 12 | 12 | 14 | 14 | 14 | 15 | 16 |
| Ce | 4.1 | 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 |

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. Figure 1 shows the growth in the lines of code (LOC) in the selected systems. The figures show a steady but stabilizing growth of these systems. 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. The average instability ($I$) also increases which shows that the packages of the jEdit systems are susceptible to change.



(a) jEdit (b) PDFBox

**Figure 1. The growth in LOC of the selected systems.**

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. The average instability (I) decreased which shows that the packages of the PDFBox systems are less susceptible to change after the major architectural restructuring.

Figure 2 shows the architectural stability evolution of the selected systems. For each system, we report ASI, The instability change $\Delta I$ 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 2(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 $\Delta I$ based on ASI only, depicted in Figures 2(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 2(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.

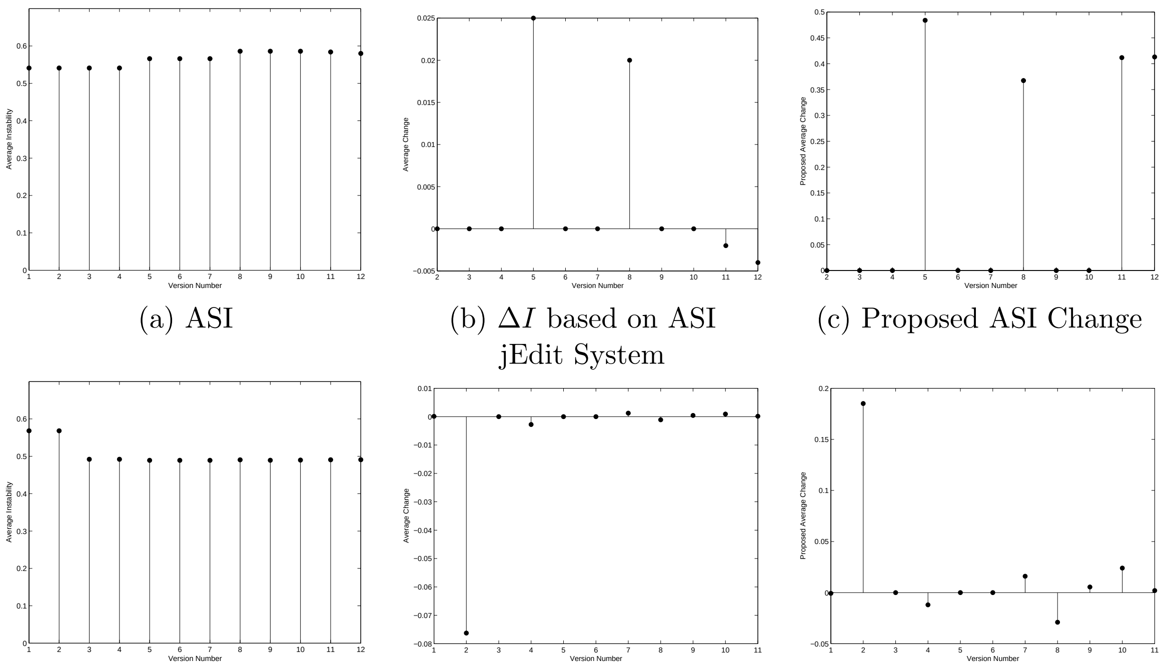
The strength of the proposed method in computing the instability can be clearly seen in case of the PDFBox system. The system started by 31 packages. In the third release number of packages was reduced to 26. The computed stability change based on averaging alone Figure 2(e) shows that stability improves 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 $\Delta I$ 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 2(f).

## 5 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.

## 6 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



(a) ASI (b) $\Delta I$ based on ASI (c) Proposed ASI Change
jEdit System

(d) ASI (e) $\Delta I$ based on ASI (f) Proposed ASI Change
PDFBox System

**Figure 2. Demonstration of the Proposed method for computing Instability change where figures (left to right) in each row represent: Aggregate System Instability (ASI), $\Delta I$ based on ASI, Proposed ASI Change.**

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 of the first versions of the software evolution laws. Koch established several dissimilarities in the evolution of open-source software projects of different sizes (Koch, 2007). 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 (D'Ambros & Lanza, 2006) 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. (Ducasse, Lanza, & Ponisio, 2005) 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 re-engineering and development of object-oriented systems, to recognize and investigate both sets of classes and packages. Wilhelm and Diehl (Wilhelm & Diehl, 2005) used Martin's (Martin, 2003) and size metrics to build a tool that helps to control package dependencies. Lungu et al. (Lungu, Lanza, & Nierstrasz, 2014) developed a tool called Softwarenaut that recover several architecture views from the source code.

Capiluppi and Boldyreff (Capiluppi & Boldyreff, 2007) 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 (Martin, 2003). They showed that low instability modules (i.e. stable modules) are good candidates to be turned into independent, external modules.

Mens et al. (Mens & Demeyer, 2001) 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. (Lee, Yang, & Chang, 2007) 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.

Several researchers proposed several metrics to measure the stability of a software architecture. Martin (Martin, 1997) proposed a metric to measure the stability based on structural dependencies. He related the stability with the dependencies between packages of the system. However, he did not mention how this metric is able to assess the evolution impact on the software architecture.

Bansiya (Bansiya, 2000) proposed a method to assess the stability of the architecture by several object oriented metrics. These class-level metrics are compared between releases to determine how changes in the structural characteristics of the different releases. The fact that these metrics are at class level, makes this approach not suitable for large systems where higher-level structures are used. The class-level is considered to be too finely grained to be used as an organizational unit for large systems.

Using the Bansiya (Bansiya, 2000) approach, AlShayeb (Alshayeb, 2011) assessed the impact of software refactoring on the stability of the architecture and recommended to avoid refactoring methods that affect the class hierarchy. However, AlShayeb did not consider the package as the basic unit of software architecture; instead focused on the class level. Therefore, the study was limited to the effect of refactoring methods at fine-grain level (methods and class).

Tonu et al.(Tonu, Ashkan, & Tahvildari, 2006) have used coupling, cohesion, and complexity as factors to assess the architectural stability. They adopted a metric-based approach to assess the architecture stability utilizing retrospective and predictive analyses. They were able to recognize where the architecture of the systems becomes stable or not.

Aversano et al. (Aversano et al., 2013) used two metrics based on a previous work (Olague, Etzkorn, Li, & Cox, 2006) and showed using an empirical study to assess how the evolution impacts the software architecture stability. Even though, these metrics provide useful insight on how much change is done to the architecture from one release to another one, one metric finds the change in terms of number of packages and the other in terms of number of the interactions among packages. However, their values are not normalized which make them not suitable to be compared with a chosen threshold.

The literature reveals that the available architectural stability measures have some limitations. For instance, the metrics proposed by (Bansiya, 2000) and the methodology presented by (Alshayeb, 2011) consider the method/class level which is fine-grain level. Bansiya work is suitable only when having cost and economic as his backdrop (Bansiya, 2000).

In this work, we proposed an easy to follow approach that enables software engineers to investigate the architectural stability between software releases. This approach will enable them to locate software packages that are susceptible to changes more that others. Our proposed approach is 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.

## 7 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.

## References

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

Alshayeb, M. (2011). The impact of refactoring on class and architecture stability. *Journal of Research and Practice in Information Technology*, *43*(4), 269.

Aversano, L., Molfetta, M., & Tortorella, M. (2013). Evaluating architecture stability of software projects. In *20th working conference on reverse engineering (wcre)* (pp. 417–424).

Bahsoon, R., & Emmerich, W. (2003). Evaluating software architectures: Development stability and evolution.

Bahsoon, R., & Emmerich, W. (2004). Evaluating architectural stability with real options theory. In *Proceedings. 20th ieee international conference on software maintenance* (pp. 443–447).

Bahsoon, R., & Emmerich, W. (2006). Architectural stability and middleware: an architecture centric evolution perspective. In *Proceedings of the 2nd international ecoop workshop on architecture-centric evolution (ace'06).*

Bansiya, J. (2000). Evaluating framework architecture structural stability. *ACM Computing Surveys (CSUR)*, *32*(1es), 18.

Breivold, H. P., Chauhan, M. A., & Babar, M. A. (2010). A systematic review of studies of open source software evolution. In *17th asia pacific software engineering conference (apsec), 2010* (pp. 356–365).

Capiluppi, A., & Boldyreff, C. (2007). 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.* (pp. 9–9).

D'Ambros, M., & Lanza, M. (2006). Reverse engineering with logical coupling. In *13th working conference on reverse engineering, 2006. wcre'06.* (pp. 189–198).

Ducasse, S., Lanza, M., & Ponisio, L. (2005). Butterflies: A visual approach to characterize packages. In *11th ieee international symposium on software metrics* (pp. 10–pp).

Godfrey, M. W., & German, D. M. (2013). On the evolution of lehman's laws. *Journal of Software: Evolution and Process*.

Gyimothy, T., Ferenc, R., & Siket, I. (2005). Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software Engineering,*, *31*(10), 897–910.

Hassaine, S., Guéhéneuc, Y.-G., Hamel, S., & Antoniol, G. (2012). Advise: Architectural decay in software evolution. In *16th european conference on software maintenance and reengineering (csmr)* (pp. 267–276).

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

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

Jazayeri, M. (2002). On architectural stability and evolution. In *Proceedings of the 7th ada-europe international conference on reliable software technologies* (pp. 13–23).

Kimelman, D., Kimelman, M., Mandelin, D., & Yellin, D. M. (2010). Bayesian approaches to matching architectural diagrams. *Software Engineering, IEEE Transactions on*, *36*(2), 248–274.

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

Kpodjedo, S., Ricca, F., Galinier, P., & Antoniol, G. (2009). Recovering the evolution stable part using an ecgm algorithm: Is there a tunnel in mozilla? In *Software maintenance and reengineering, 2009. csmr'09. 13th european conference on* (pp. 179–188).

Lee, Y., Yang, J., & Chang, K. H. (2007). Metrics and evolution in open source software. In *Seventh international conference on quality software, 2007. qsic'07.* (pp. 191–197).

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

Lungu, M., Lanza, M., & Nierstrasz, O. (2014). Evolutionary and collaborative software architecture recovery with softwarenaut. *Science of Computer Programming*, *79*, 204–223.

Martin, R. C. (1997). Stability article. *Engineering Notebook, The C++ Report*.

Martin, R. C. (2003). *Agile software development: Principles, patterns, and practices.* Upper Saddle River, NJ, USA: Prentice Hall PTR.

Mens, T., & Demeyer, S. (2001). Future trends in software evolution metrics. In *Proceedings of the 4th international workshop on principles of software evolution* (pp. 83–86).

Olague, H. M., Etzkorn, L. H., Li, W., & Cox, G. (2006). Assessing design instability in iterative (agile) object-oriented projects. *Journal of Software Maintenance and Evolution: Research and Practice*, *18*(4), 237–266.

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

Tonu, S. A., Ashkan, A., & Tahvildari, L. (2006). Evaluating architectural stability using a metric-based approach. In *Proceedings of the 10th european conference on software maintenance and reengineering* (pp. 10–pp).

Wilhelm, M., & Diehl, S. (2005). Dependencyviewer-a tool for visualizing package design quality metrics. In *Vissoft* (pp. 125–126).