Software Instability Analysis Based on Afferent and Efferent Coupling Measures

Danilo B. Santos*, Antônio M. P. Resende, Eudes C. Lima, André P. Freire

Research Group on Software Engineering, PQES, Department of Computer Science, Federal University of Lavras (UFLA), PO Box 3037, 37200-000, Lavras, MG, Brazil.

* Corresponding author. Email: danilobatista@posgrad.ufla.br, tonio@dcc.ufla.br, eudes@posgrad.ufla.br, apfreire@dcc.ufla.br
Manuscript submitted July 30, 2016; accepted October 31, 2016.
doi: 10.17706/jsw.12.1.19-34

Abstract: Software instability measures indicate the necessity to modify a software module (class, package, subsystem, etc) due to changes in other related software entities. If there is low instability, then there is evidence the analyzed entity has little dependence on others and the project has a good maintainability. Otherwise, there is evidence that the analyzed entity is sensitive to changes occurred in other entities. In the latter case, software reconstruction could be necessary and the maintainability becomes harder because of dependencies. Consequently, the higher the value of instability in an entity the more vulnerable it is to unexpected changes, even if the entity does not suffer direct changes in its code. This article adopts the instability definition of Martin [1] that depends on the afferent (Ca) and efferent (Ce) coupling metrics. It presents a Systematic Literature Review (SLR) of Martin's instability looking for reference values published in scientific articles and practiced in the open source market. Furthermore, this article analyzes the Martin's instability equation and the evolution of Ca, Ce and instability through new releases of 107 software. Authors applied a systematic literature review (SLR), and observed that there is a shortage of reference values in scientific articles. They performed a statistical analysis of instability measures in 107 free software products, involving three different versions of each, totaling 321 product versions. It was not possible determine or suggest a reference value to Ca, Ce and instability measures due to the high variation of those measures. It was observed that 48% of software products had high instability equal to 1, the maximum value allowed, and the instability average obtained was 0.7. Based on results of this paper, we conclude that software architects and engineers should concentrate more efforts to produce low instability software since first version, because the most of software keep the instability level through the releases. More analysis is necessary to confirm this behavior about software instability through releases.

Key words: Software instability analysis, afferent coupling, efferent coupling.

1. Introduction

Software engineers recommend the use of software measures in order to monitor projects, discover non-conformities and point out risks like low modularity in software projects since the early stages of project development.

There are several measures applicable to software projects and products. This article focuses on the Martin's instability measure, which indicates the necessity of performing modifications in an entity due to updates occurred in other software entities. Thus, if an entity has a high value of instability, then there is a high risk of undesired changes could affect the analyzed entity's behavior, due to changes in other system

entities. The reverse is also true. Thus, a low instability value means there is a small risk of change in the analyzed entity's behavior, due to changes in other system entities. For this reason, the instability is highly dependent on the existing coupling level among entities. Martin [1] instability measure definition is related to dependencies among entities, for instance, dependencies (coupling) among software packages. Chindamber and Kemerer [2] define coupling as "...any evidence of method of one object using methods or instance variables of another object constitutes coupling". Software coupling measures are important because they aim to analyze the relationship between two software entities, which can directly influence the instability, considering the definition of Martin [1].

The Martin's instability could be used to evaluate the software modularity because it analyzes the dependencies among packages. The ISO/IEC/IEEE 24765:2010 states the modularity is the "degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components". Similarly, we can use cohesion and coupling to evaluate the dependencies and modularity as well.

Some authors argue that software packages should present low coupling and high cohesion, in order to improve quality in the software [3]-[5]. As maintenance affects directly the coupling and cohesion of software [6], it could affect the software instability and, consequently, the final software quality.

However, the task of assessing and diagnosing software has been limited by the lack of reference values for software measures [7]. This fact is a consequence of the broad spectrum for software measures with an absence of reference values and without validation, important information to help in the decision-making [8].

This paper analyzed Martin's software instability by two different methods. The first analysis involved the conduction of a systematic literature review, to identify the reference values of Martin's instability proposed by academia. The second method consisted of a statistical analysis of three versions of 107 free software products, in order to identify the value of the instability practiced in the free software market. The Martin's instability is based on the afferent and efferent coupling measures.

The first method was important because if there were no reference values for measure instability as pointed by Tempero *et al*. [7], then the use of measures becomes limited. It is essential to establish reference values in order to provide references more robust to evaluate and diagnose whether software is improving, worsening or stabilizing. This article concludes there is a lack of reference values for Martin's instability measure.

The second activity allowed us to understand what had happened with software instability after applying statistical analysis to three versions of 107 open source software written in Java, released and in use.

Section 2 presents a definition of measures of afferent and efferent coupling and instability. Following, section 3, presents the conduction of a Systematic Literature Review (SLR), which aimed to collect afferent and efferent coupling values reported in scientific articles. Section 4 presents a mathematical analysis of instability measures defined as a function and a discussion of their maximum and minimum values. Section 5 presents a statistical analysis of open source software projects, in order to identify the values of instability prevailing in the open source market. Finally, Section 6 presents conclusions and future work.

2. Afferent and Efferent Coupling, and Instability

The measure of instability, proposed by Martin [1], has the opposite concept of stability presented by the ISO/IEC 25000 [9]. Martin's instability depends on coupling among software entities and stability of ISO/IEC 25000 depends on lacking of coupling. Consequently, the values of stability and instability are correlated inversely. However, both have the same goal, which is indicating the potential effects that an analyzed entity may suffer due to the changes made in other entities. For instance, an entity could be a

software class, package, subsystem, etc, used by other entities. Therefore, if a class A use a class B, then the class B updates can affect the class A.

If an entity has low instability, then there is evidence that the analyzed entity has little dependence on others. Otherwise, there is evidence that the analyzed entity is highly dependent on other entities.

Martin's Instability (I) is an indirect measure, which depends on afferent coupling (Ca) and efferent (Ce) coupling [1] measures, determined by the formula:

$$I = (Ce/(Ce + Ca))$$

Martin [1] defined Ca and Ce as follows. Ca counts the number of classes following the rules: a) classes counted must depend on the analyzed entity; b) classes counted must be outside of the entity analyzed; and c) each class is counted just once. Ce counts the number of classes following the rules: a) classes counted must be inside the entity analyzed; b) classes counted must depend on other classes located outside of the entity analyzed; c) each class is counted just once.

In order to calculate the value of Ca for Package_1 (Fig. 1), a software engineer should count the number of classes out of the package that have dependencies incident on it. For example, the package Package_1 has classes called A and B. The sets that define the classes dependent on A and B are {C, E, F} and {C, D}, respectively. The union of these two sets is the set of classes that depend upon Package_1. In this case, the union set is given by {C, D, E, F}. Therefore, a software engineer concludes the Ca value for Package_1 is equal to 4, the total elements in this union set {C, D, E, F}.

It is noteworthy that class C (Fig. 1) has more than one dependency relationship focusing on Package_1, as seen in the two dependency sets. However, the software engineer counts all dependencies only once, even a class that has more than one incident dependency relationship on the same package, like in the case of class C.

Similarly, the counting procedure is applied for the remaining packages, resulting in zero for Package_2 and Package_3.

To calculate the value of Ce, the software engineer must count the amount of classes dependent on other packages inside the analyzed package. For example, Package_2 has two classes C and D (Fig. 1). First, define the sets of classes that depend on classes C and D. In this case, the sets are {A, B} and {B} respectively. Considering Package_2, the value of Ce is equal to 2. It means there are two classes inside Package_2, having external dependencies.

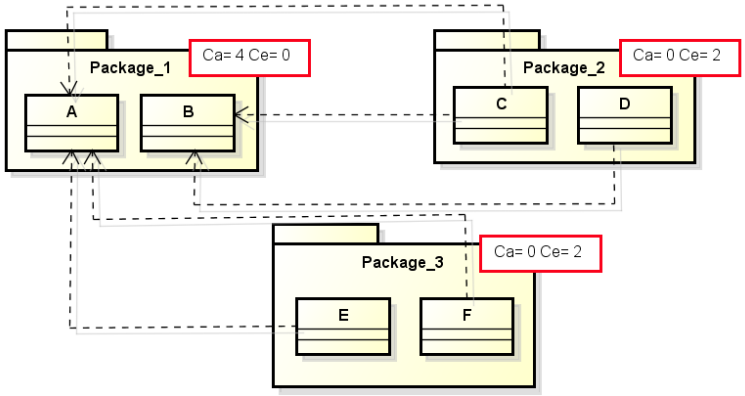

Fig. 1. Example of afferent couplings and efferent.

A software engineer can calculate the Ca and Ce of the entire system, presented in Fig. 1, adding every Ca and Ce of each package in the system. In this case, the value of the Ca and Ce are both 4.

3. Systematic Literature Review (SLR)

This article applies the concepts of SLR, presented in accordance with three research papers [10], [11], [12]. The SLR protocol proposed was defined according to the recommendations provided by Biolchini [12].

3.1. Planning

The planning involved five main topics called "questions of the SLR", being the language of articles, database set, inclusion and exclusion criteria, and search string. The questions of the SLR were: **i**) what are the reference values suggested in the literature for the software measures afferent and efferent coupling? **ii**) what are the methods of calculation of such values for these measures? Articles to be included in the SRL must have been written in English.

The database set used included the most relevant sources for papers in Software Engineering, being the IEEE Xplore, Scopus, Springer, Ei Compendex, Science Direct and the ACM Library. The inclusion and exclusion criteria were: **i**) being part of a conference proceedings or peer-reviewed journal; **ii**) to propose or validate at least one of the measures selected for this particular work; **iii**) propose benchmarks or provide measurement methods for Ca and Ce measures; and **iv**) have unrestricted access to its content by the authors of this work. The search string was:

(("afferent coupling") OR ("efferent coupling")) AND (metric OR metrics) AND ((range OR ranges) OR (interval OR intervals) OR (measure OR measures OR measuring OR measurement) OR (threshold OR thresholds) OR (("reference value") OR ("reference values")) OR (limits OR limit))

3.2. Execution and Results

The initial search started in August 2014. The second column of Table shows the number of papers found as result of the use of the search string in the selected databases. The JabRef[1] tool supported the organization, sorting and selection of articles during the primary selection execution.

In the primary selection, the articles were selected after examining the title, keywords, abstract, and then applying the exclusion criteria explained in Section 3.1 The third column of Table presents the number of papers selected in this step. Columns 4, 5 and 6 of Table present the number of papers included after the secondary selection. This phase included a thorough screening after examining the title, keywords, abstract, introduction, results and conclusion. Also, articles were classified at that stage as irrelevant, repetitive and incomplete were excluded from the study.

At the end of the conduction of the primary and secondary selections, four articles were selected and thoroughly examined. One article cited by the four articles drew attention from the researchers and received the classification of "extra article".

Table 1. Results of the Systematic Review

Bases	Initial Search	Primary selection	Secondary Selection			Results of primary studies	Included
			Irrelevant	Repeated	Incomplete		
IEEE	77	7	6	0	0	1	1
Science	27	4	2	0	0	2	
EI Compendex	57	4	2	1	0	1	
Scopus	3	1	0	1	0	0	
Springer	26	2	2	0	0	0	
ACM	40	3	3	0	0	0	
Total	230	21	15	2	0	4	1

[1] http://jabref.sourceforge.net/

Those extra articles received the same analysis process applied to primary and secondary selections. Regarding the databases listed in the SLR planning, none of them indexed the extra article found because it was not published in an indexed magazine or event. If the extra article was indexed, we had found it applying the same search string. This fact explains the absence of the extra article in the SLR. This conclusion was possible after seeking the article cited in the databases used without success. Finally, the last column in Table 1 shows one extra article added.

The SLR found 230 papers initially. IEEE contained 33.47% of the papers, ScienceDirect contained 11.73%, EI Compendex contained 24.78%, Scopus contained 1.30%, Springer contained 11.30%, and ACM contained 17.39%.

Table presents the selected articles detailing their titles, publication year, and the database in which they were found. The article classified as "not indexed" represents the extra article found following leads from citations in the articles examined in the regular search.

Table 2. Selected Articles by Systematic Review

#	Title	Year	Database
1	OO Design Quality Metrics An Analysis of Dependencies. [1]	1994	not indexed
2	Exploring the Relationships between Design Metrics and Package understandability A Case Study. [14]	2010	IEEE Xplore
3	Empirical comparison of three metrics suites for fault prediction in packages of object-oriented systems: A case study of Eclipse. [15]	2011	Science
4	Investigation of Aspect-Oriented Metrics for Stability Assessment: A Case Study. [16]	2011	EI Compendex
5	Identifying thresholds for object-oriented software metrics. [17]	2012	Science

Table summarizes results obtained after analyzing the selected articles. The reference values and the methods used to define the reference values are answered for Ca and Ce measures. The identifier of article is in first column, indicating the data source.

There are some threats to the validity of the SLR based on ignoring important articles. It could have happened if: a) they used another nomenclature for Ca and Ce but the same sense given by Martin; b) they did not appear in any database used; and c) they contained inappropriate description of the title, abstract, keywords, or conclusions. Two people led the selection process independently. However, despite all the efforts in being as thorough as possible, a relevant article might have been removed incorrectly.

4. Analysis of Instability

A behavioral analysis of the function instability was performed regarding Martin's definition and formula, described in Section 2. The values Ca and Ce only varied from 0 to 10 in order to calculate the value of the instability and present the curve. However, the values of Ca and Ce are generalizable in a range from 0 to N. The analysis intended to demonstrate how the instability varies according to Ca and Ce variations. Martin [1] presented two categories of software entities called "independent" and "responsible". If an entity is independent, it has no dependencies on other parts of the system. If an entity is responsible, then it has several other entities that depend on it. An entity has to be independent and responsible to be considered the most stable.

Table 3. Summarization of SLR Results

Articles	Reference Values		Calculation Method	
	Ca	Ce	Ca	Ce
[1]	Not Present	Not Present	Ca=Σ Adependencies, where Adependencies are the dependencies of classes external of category that depend on the	Ce=Σ Edependencies where Edependencies are the dependencies of the category on external classes to the category.

			category.	
[13]	Not Present	Not Present	Not Present	Not Present
[14]	Not Present	Not Present	Not Present	Not Present
[15]	Not Present	Not Present	Not Present	Not Present
[16]	Size (#classes) / Intervals Good / fair /poor ≤100 [0;1] / [2;20] / >20 101–1000 [0;1] / [2;20] / >20 >1000 [0;1] / [2;20] />15	Not Present	Not Present	Not Present

Fig. 2 shows the result of the analytical description of the mathematical functions. On the horizontal axis are the values of the Ca measure. The depth axis has values of the Ce measure, and the vertical axis has instability values given by the formula. The reader should note that Ca increases the value from the right to the left due to the rotation of the map curves for better visualization.

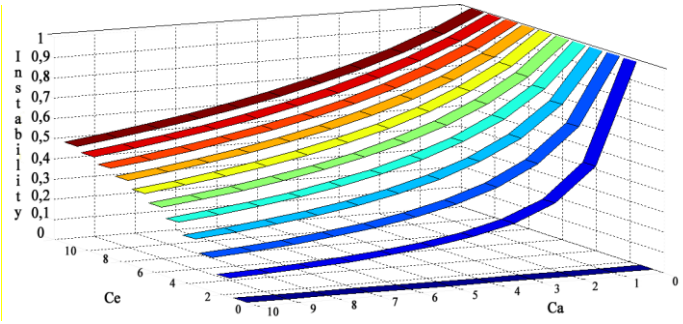

Fig. 2. Behavioral analysis of instability based on Ca and Ce.

The following statements about Martin's instability can be written looking at the graphic and the instability formula: **i**) is zero when the Ce value is equal to 0, regardless of the value presented by Ca; **ii**) decreases when the Ce value is constant, and the Ca values increases; **iii**) decreases when the Ca value is constant, and the Ce values decreases; **iv**) increases when the Ce value is constant, and the Ca values decreases; and **v**) increases when the Ca value is constant and the Ce value increases.

Fig. 2 shows that the higher the amount of Ce in comparison to the same Ca value, the greater the instability of the analyzed entity. This also shows that the higher the amount of Ca in comparison to the same Ce value, the lower the instability of a software.

4.1. Minimum and Maximum Values for Ca and Ce

In order to understand better the instability function, Table 4 presents the general minimum and maximum values for Ca and Ce of a software package in relation to other existing packages in any analyzed system. Those values were determined according to the concepts defined by Martin [1] and cited in Section 2. The acronym NC is the total amount of existing classes in the system minus the amount of classes present in the package (entity) analyzed.

The Ca measure has the minimum value 0 when a package consists of classes from which no other classes' packages depend on them. The maximum value is equal to the NC when all classes in other packages of a system depend on the classes of the reporting package.

The Ce has the minimum value of 0 when analyzed package classes do not depend on any other external class in the analyzed package. The CN denotes the maximum Ce value. That occurs when the set of all classes in the analyzed package depends on all the system classes, except for the classes present in the analyzed package itself.

5. Market Pratices of Open Source Software

This section presents the results of statistical analyses applied to 107 open source software projects in order to identify the values of Ca, Ce and instability measures practiced in the market. This analysis includes descriptive statistics, means test and an exploratory analysis of the responsive measures. The software measures were gathered automatically through a tool.

Ten (10) software measurement tools were listed as candidate tools to collect the measurements of software. The criteria for the selection of tools were: **i**) the results should be conveyed per package; **ii**) the tool should perform analysis on the Java source code; **iii**) the tool should have released the newest stable version after 2011; and **iv**) the tool should measure Ca and Ce in accordance with the definition of Martin [1]. Table shows the selection results of the measurement tool analysis. CodePro was the tool selected.

Table 4. Results of Measuring Tools Selection

Tools	Result of Selection	Reason
AnalysT4j	Rejected	Support & Development Discontinued
CCCC	Rejected	Support & Development Discontinued
CodePro	ACCEPTED	Met all the selection criteria
Ckjm	Rejected	Do not perform analysis on the packages
DependecyFinder	Rejected	Does not perform analysis java files
JDepend	Rejected	Inadequate measure of the measures
Metrics1.3.8	Rejected	Does not perform analysis java files
NDepend	Rejected	Does not perform analysis java files
Refactor IT	Rejected	Only projects with less than 50 classes
SD Metrics	Rejected	Does not perform analysis java files

In Table the column titled "Tools" displays the name of the metric tools. The column entitled "Results of Selection" shows the acceptance of a tool (presented all the selection criteria) or rejection (did not meet at least one of the selection criteria). Finally, the column entitled "Reason" justifies the main reason for rejecting the tool.

The selected dataset used to analyze market practices consisted of 107 open source software projects in three versions each, totalizing 321 software projects versions. Those projects are from the SourceForge[2] repository. The criteria applied to select those software projects were: **i**) software projects written in Java language; **ii**) availability of source code; **iii**) existence of three versions available; and **iv**) the versions released after 2010.

The first version (V1) of each software project should be the first version released after 2010. The last version (V3) of each software project should be the last version released in source forge repository. In this case, the software projects where gathered in 2014. The second version (V2) of each software project should be the intermediate number version between the first and last version. Three software have one version before 2010, in order to complete the 321 software, because there were just two version between 2010 and 2014. So, three software did not respect the last software projects selection criteria.

5.1. Statistical Analysis

This section presents the results of statistical analysis techniques about Ca, Ce and Instability measures, their behavior and current market practices in open source projects.

The statistical techniques applied are descriptive analysis, comparison of means, frequency distribution and a specific method to evaluate the measure evolutions along the versions. The R-Software and its interface package R-Studio supported the analysis [13].

[2] http://sourceforge.net/

5.2. Descriptive Statistics

Table shows the results of descriptive statistic analysis about open source software market for the 107 software projects, regarding three versions of each software, totaling 321 instances. The columns in Table, entitled "Ca", "Ce" and "Instability", shows the descriptive statistics of software measures for three software versions (V1, V2, V3). Each row presents a statistical measure.

Table shows that the measures have an asymmetric distribution, because the mean, median, and mode are not the same, considering the same measure in the same version. We observed a high standard deviation for the measures, which reveals high dispersion of values from the mean.

Table 6. Descriptive Statistics of Ca measures, Ce, and Instability

Statistical Measures		Ca			Ce			Instability		
		V1	V2	V3	V1	V2	V3	V1	V2	V3
Mean		36.3	49.0	73.7	140.6	188.7	198.8	0.7	0.7	0.7
Std. Error		11.2	12.4	16.5	18.7	22.2	22.7	0.0	0.0	0.0
Median		0.0	0.0	0.0	57.8	100.0	126.0	0.8	1.0	0.7
Mode		0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0
Std. Deviation		115.8	128.3	171.2	193.4	229.5	235.3	0.4	0.4	0.4
Minimum		0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Maximum		804	843	916	881	950	963	1.0	1.0	1.0
Quartiles	25%	0.0	0.0	0.0	1.0	1.4	1.4	0.5	0.6	0.6
	50%	0.8	0.0	0.0	57.8	100.0	126.0	0.8	1.0	0.7
	75%	15.0	31.0	62.0	192.3	316.0	302.0	1.0	1.0	1.0

The dispersion reflects different architectures and designs implemented in each software project. Software projects with similar size have a different number of packages affecting the cohesion and coupling of packages. Thus, the measures of Ca, Ce, and the instability change as well.

If we consider high instability the values greater than 0.75 or the 25% highest values of instability, then at least 50% of version 1 and 2 have high instability. We highlight that the first quartile is enough to presents instability greater than or equal 0.5.

We used the normality test called Kolmogorov-Smirnov, with 5% significance to test the following hypotheses: (i) H0 - data follow a normal distribution; (ii) H1 - data do not follow a normal distribution. The results showed the measures of Ca, Ce and instability do not have a normal distribution because the significance is smaller than the p-value of 0.05. The high values for standard deviation compared to mean values of Ca and Ce means the values of Ca and Ce are spread out too much. That avoids the possibility to get a reference value or the most common range for Ca and Ce measures.

5.3. Comparison of Means

The nonparametric Kruskal-Wallis [17] was the means comparison test applied, given the data did not follow a normal distribution. This test requires independence of the data and can be applied only between treatments (different software projects) because they are independent. The Kruskal-Wallis could not be applied to compare means of different versions of the same software project, considering that the last version of a software product is an evolution of the previous ones, and there is reuse of code, packages, interfaces, databases, and others. Therefore, a version X of a software influences the version X + 1. That reuse causes dependency within treatments and Kruskal-Wallis only works for independent data.

Respecting the mandatory rule of independence data, Kruskal-Wallis is applied to analyze different software projects. The objective of Kruskal-Wallis test was to verify if the measures Ca, Ce, and I differ significantly from the same version of each software project at with 5% significance level. The hypotheses

for this test were: **i**) H0 - the measure is statistically equal in the 107 software projects; **ii**) H1 - at least one software project has a significantly different value for the measured among 107 software projects. Table displays in its columns the version of a software project, the value of the Kruskal-Wallis chi-square, the degrees of freedom of the test, and the p-value, respectively.

The result indicates there are no significant differences of instability between the software in three versions because the values presented by p-value are greater than the specified significance (0.05). Therefore, the test failed to reject H0, which states that the variation of instability between the software of a same version was not significantly different. Similarly, the same can be said for Ca and Ce.

Table 7. Test Medium Kruskal-Wallis

Versions	Kruskal-Wallis Chi-square	d.f.	p-value I	p-value AC	p-value Ce
V1	106	106	0.4817	0.4817	0.4817
V2	106	106	0.4817	0.4817	0.4817
V3	106	106	0.4817	0.4817	0.4817

Consequently, software projects with high/low coupling values hold almost the same value statistically throughout the versions. Based on the Kruskal-Wallis analysis, we can state that the mean variations of Ca, Ce and I among versions seen in Table are not caused by the version variation (new releases), but due to other factor(s) not identified. That implies the software industry and software engineers should apply extra efforts to provide a good coupling measure to software projects since the first version. Note that new releases did not change the coupling values significantly.

5.4. Frequency Distributions of Market Practices

The frequency distributions showed that 48% of software produced currently has instability equal to 1, the maximum allowed value for the instability. Therefore, 48% of the software available in the open source market can be considered highly unstable according to the definition of Martin [1]. That means a high coupling among entities of a software project.

The bar relating to the value 1 and 48% of relative frequency in Fig. 3 has been removed to facilitate the visualization of other data. The graph shows the bars from the second value most frequent, which was approximately 4%. Different instability values of one (1) concentrated around 0.57 value mostly.

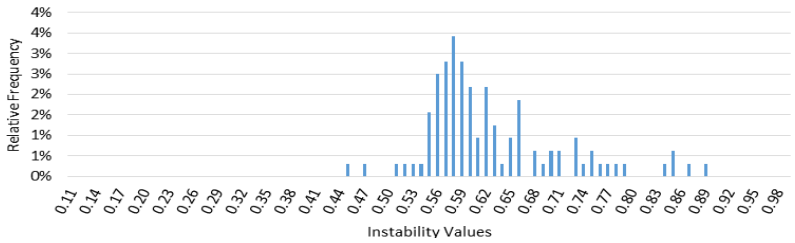

Fig. 3. Frequency of Instability in the 107 software projects in their 3 versions.

Fig. 4 shows the relative frequency values of Ce after analyzing 107 software projects in their three versions. The value of one (1) is the most frequent in the open source market, currently with a relative frequency of 21%. The bar corresponding to the amount of Ce equal to 1 has been omitted to facilitate viewing of the remaining bars in the graph. The value of the second most frequent, among others, can be seen in the graph, and its value is below 1.40%. Most of the Ce values are between 0.60% and 1% approximately, with Ce values rather scattered.

Fig. 5 shows the relative frequency of Ca values after analyzing 107 software projects in three versions

each one. Note the value zero (0) is the most practiced on the market with 69% relative frequency. The bar corresponding to the amount of Ca equal to 0 has been omitted to facilitate viewing of the remaining bars in the graph. The second Ca value most frequent, among others, was less than 1%.

Regarding the results presented in Fig. 3, Fig. 4 and Fig. 5, and there is NOT significant difference between versions of I, Ca and Ce (shown in Comparison of Means section), the Instability, Ca and Ce hold similar values throughout new releases. Consequently, that is an incentive to software industries and software engineers apply more efforts to get better values in the first version.

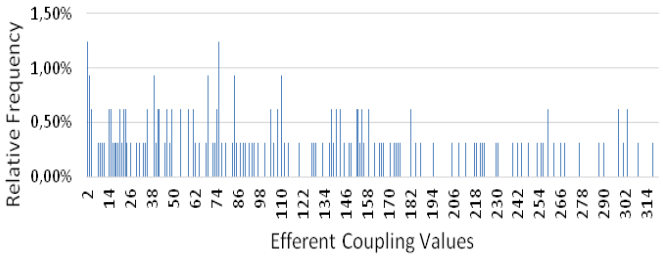

Fig. 4. Frequency of Ce values in the 107 software.

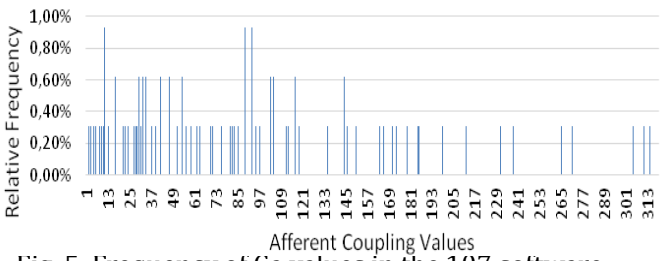

Fig. 5. Frequency of Ca values in the 107 software .

5.5. Exploratory Analysis of I, Ca and Ce evolution among Versions

An exploratory analysis aimed to answer the question: What is the variation of instability, Ca and Ce between versions of the same software?

The first step was to calculate the instability value of a version minus the instability value from a later version. The authors applied that calculus for all possible combinations of versions, for instance, calculating the instability of version 2 minus the instability of version 1 for each software. The version variations regarded are version 1 to version 2 (V1→V2), version 2 to version 3 (V2→V3) and version 1 to version 3 (V1→V3).

The second step consisted of classifying the instability variation as positive, negative, or zero. If the instability increased from one version to another, the variation was positive. If the instability decreased, the change was negative. Finally, the variation cases with value zero were named null.

The third step consisted of calculating the frequency distribution of positive, negative and null variations, classifying them into class intervals. The authors decided to create ten classes. Regarding only instability, each class represented a variation range of 20% of 1, because the highest value of instability is 1. Thus, the class entitled "]0%;-20%]" represents all software that has changed from 0 to -0.2. A software project that had instability with value 0.8 in version 1 and had instability with value 0.7 in version 2 had a -0.1 instability variation, and it belongs to the class "]0%;-20%]". That means the instability was reduced from version 1 to version 2 in -0.1.

Different from instability, Ca and Ce were not restricted in a well-defined range. In order to normalize the variation and determine the size of each class interval, the lowest value was subtracted from the highest value presented by the measures and the result was divided by 10 (number of class intervals), determining the amplitude of each interval.

Table shows the frequency of these class intervals for instability. The first column presents the class intervals, followed by the frequencies. Regarding the instability, the value zero had a high frequency. Thus, the “zero” became a special interval class, as a highlighted line in Table, making it easier to comprehend the behavior of the measure Ca, Ce and I. The class interval zero showed its high incidence, which could interfere with the frequency analysis of the intervals.

Fig. 6 and Fig. 7 present the analysis results for the instability measure. Fig. 6 shows the frequency of positive, null and negative variations between versions for instability. The first set of bars entitled "V1→ V2"

indicates 52% of software did not change the value of instability from version 1 to version 2 after analysis of 107 software projects. Similarly, positive and negative variations occurred in 23% and 24% of 107 software projects, respectively.

Note the null variation (Fig. 7) was equal or greater than 50% of software projects in all three cases of the variation. Therefore, the instability did not change in more than half of the cases when the software version had changed. This indicates a low variability of architecture and design software, meaning that the first instability value of software is kept along new versions in at least 50% of projects.

Table 8. Relative Frequency Variation Instability among Versions

Percentage Range	Relative Frequency		
	V→V2	V2→V3	V1→V3
]-80%; -100%]	7%	6%	6%
]-60%; -80%]	0%	0%	0%
]-40%; -60%]	2%	3%	2%
]-20%; -40%]	3%	1%	4%
]-0%; -20%[	13%	16%	16%
[0%]	52%	59%	50%
]0%; 20%]	7%	9%	7%
] 20%; 40%]	4%	2%	5%
] 40%; 60%]	1%	1%	2%
] 60%; 80%]	2%	0%	2%
] 80%; 100%]	9%	4%	6%
TOTAL	100%	100%	100%

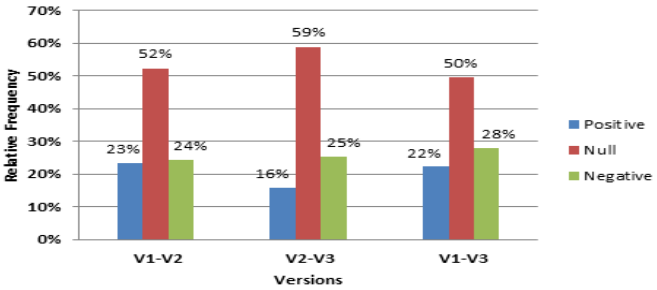

Fig. 6. Instability variation analysis between versions.

Regarding previous analyses about frequency distributions of open source market practices, the instability 1, higher possible level, appeared in 48% of software projects. Thus, most software projects presented had high instability according to Martin's definition, and the high instability is kept high due to the high null variation. In addition, there is a balance between positive and negative variations. Both of them had a value around 25% of software projects in most cases of the three bar sets.

The set null bars involving changes from version 1 to version 3 presents 50% of software as null variation. That also implies 78% of software had the instability decreased or null, and 72% of software did not change or had instability increased.

Fig. 7 shows the variation of the relative frequency instability as shown in Table. The class interval entitled "[0%]" represents a zero variation in the instability value, and its frequency was higher than 50% in all versions of comparisons.

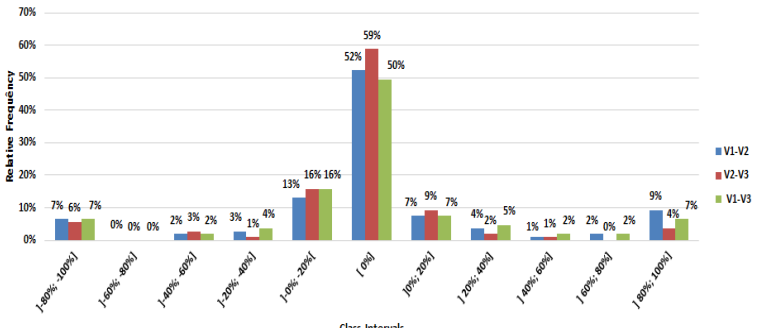

Fig. 7. Relative Frequency Distribution of Instability Variations.

The higher positive variation is 1 (one) and occurs when a version of software has instability 0 (zero) and

another recent version has instability of 1. The higher negative variation is -1 and occurs in the opposite way. The class interval "]-0%; -20%]" represents value variations of instability from a less than 0 up to -0.2. The class interval "]0%; 20%]" represents value variations of instability from a greater than 0 up to 0.2. The other classes are similar varying 0.2 between each class.

In the Fig. 7, the analysis presents a concentration at zero value and first left and right interval class entitled "[0%; 20%]" and "[-0%; 20%]". Therefore, there was a low variation of instability in most software projects, even in the 107 software projects that presented a high instability value. That indicates that the first version of the software product had high instability and it held a high value even releasing new versions.

The same analysis method was applied to measure Ca, shown in Table, in Fig. 8 and Fig. 9. Likewise, for instability, the measure Ca had a large percentage of cases with no changes between versions in most of the software projects (Fig. 8). However, the percentage of cases in which the Ca variation was positive varies greater than the percentage of negative variation (Fig. 8). This behavior shows a tendency in increasing the Ca with the evolution of software.

Fig. 9 confirms that the statement where the bars located on the right side of "[0%]" are taller than the bars located on the left side of "[0%]". In addition, there was a positive variation in the most of the cases. The left side of Fig. 9 shows mostly 0% of variation, indicating the Ca did not decrease its value.

The same analysis method was applied to measure Ce showed in Table, in Fig. 10 and Fig. 11. The percentage of positive changes in Ce is substantially greater than the negative and null variations. Fig. 10 shows positive variations of 66%, 59%, and 67% while other bars show values from 15% to 23%. Consequently, the values indicate an increase in the number of dependencies on services provided by entities outside the entity analyzed, increasing the instability.

Table 8. Relative Frequency of Ca Change between Versions

Percentage Range	Relative Frequency		
	V1→V2	V2→V3	V1→V3
]-80%; -100%]	1%	1%	1%
]-60%; -80%]	0%	0%	0%
]-40%; -60%]	0%	0%	0%
]-20%; -40%]	0%	0%	0%
[0%]	66%	65%	62%
]0%; 20%]	17%	10%	17%
] 20%; 40%]	12%	1%	3%
] 40%; 60%]	12%	1%	2%
] 60%; 80%]	1%	1%	0%
] 80%; 100%]	0%	1%	3%

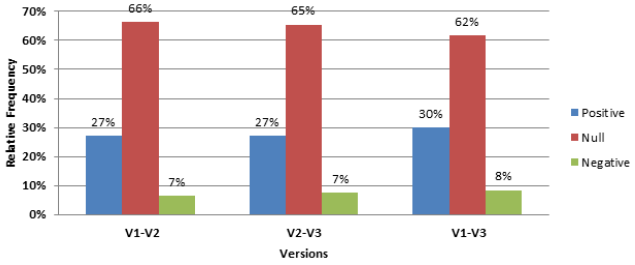

Fig. 8. Ca variation analysis between versions.

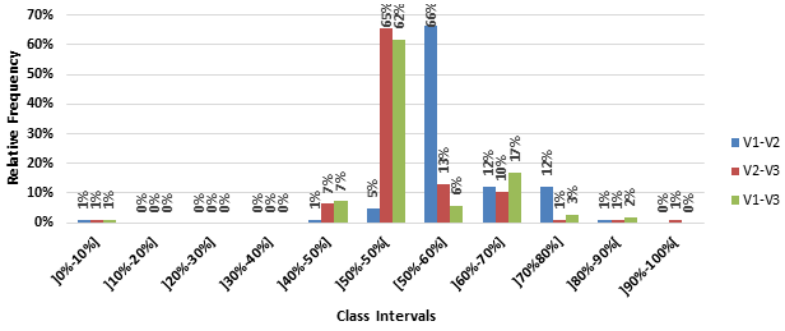

Fig. 9. Relative frequency distribution of ca variations.

Despite the high increase of Ce between versions, this variation showed low amplitude. The interval [0%] representing null variation was 17%, 18% and 15% of software (Fig. 11). The interval entitled "[0%; 20%]" presents 50%, 10% and 30% of software varying positively. The interval entitled "[20%; 40%]" presents 7%, 42% and 25% of software varying positively. There is little software in the other variations. Therefore, a variation of Ce is concentrated on the positive side, indicating higher values for new versions released.

Table 9. Relative Frequency of Ce Variation between Versions

Interval Percentage	Relative Frequency		
	V1→V2	V2→V3	V1→V3
]-80%; -100%]	3%	4%	2%
]-60%; -80%]	2%	2%	0%
]-40%; -60%]	3%	1%	5%
]-20%; -40%]	7%	1%	5%
]-0%; -20%[	52%	26%	36%
[0%]	17%	18%	15%
]0%; 20%]	7%	42%	25%
] 20%; 40%]	3%	1%	6%
] 40%; 60%]	4%	3%	2%
] 60%; 80%]	1%	1%	3%
] 80%; 100%]	1%	2%	2%

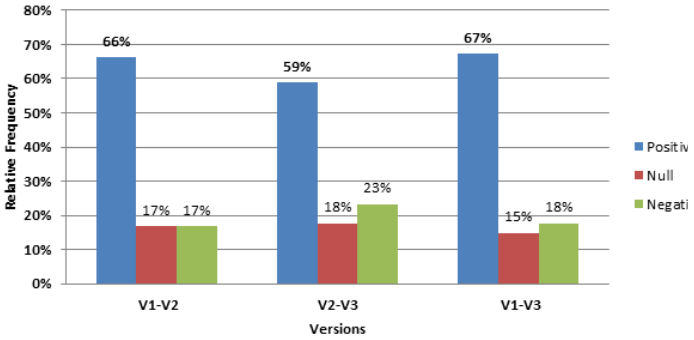

Fig. 10. Ce variation analysis between versions.

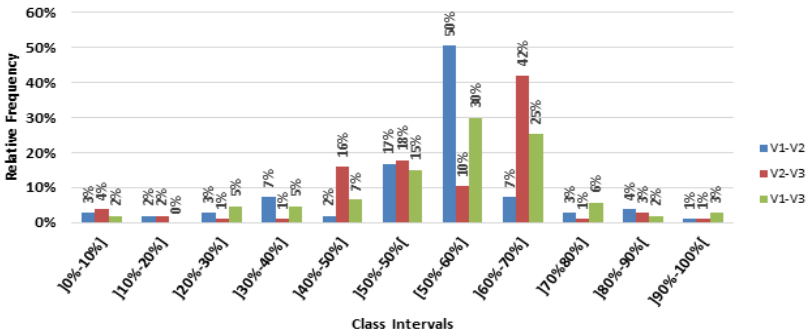

Fig. 11. Relative frequency distribution of ce variations.

5.6. Additional Discussion

The variation of instability average among software versions was almost null for more than 50% of software projects analyzed, regarding variation from version 1 to version 2, from version 2 to version 3, and from version 1 to version 3.

Regarding that 48% of software projects had instability equal 1, we can state that software projects have high instability and that instability is kept through the time. A sentence could become a hypothesis to be confirmed in future is: “Software projects achieve high instability and nothing is done to revert the instability”. Other sentence could be analyzed is: “The instability of major software projects doesn’t get variation thought the time.” Those sentences, if proved, could point out the lack of software architectural attention. So, software projects get high instability, in some release and keep it high.

Considering the instability reflects package dependencies, and as bigger is the dependencies as bigger is the instability, we could be facing a conclusive fact that efferent coupling (Ce) increase through the time more than afferent coupling. We can see, in Fig. 10, that the major Ce variations were positive when occurred. We can see, in Fig. 8, that the major Ca variations were negative when occurred. In this case, if Ce increase and Ca decrease through the time then instability must increase according Martin’s instability

formula.

6. Conclusion

The purpose of this research was to analyze the instability (I) based on afferent coupling (Ca) and efferent coupling (Ce) measures. First, there was a SLR to identify benchmarks, and Ca and Ce calculation methods. Subsequently, the authors performed an analysis of the Ca, Ce and instability values practiced by the software market of open source.

Based upon the Systematic Literature Review (SLR), the authors concluded that, there is a shortage of work defining the calculation method or proposing reference values for Ca, Ce and Martin's Instability [1]. Only one article covered the Ca calculation method and Ce. That means only 0.31% of the articles obtained in the initial search contains the desired content. However, all papers analyzed had used or mentioned Martin's definition of the measures, giving robustness in relation to its definition.

The analysis of market practices indicated that the instability of software tends not to vary or varies very little between versions of software. Statistical analysis was applied to 107 software in the three different versions and 48% presented the highest instability according to Martin's definition. Therefore, the instability was high and holds high throughout the new releases.

The Ce measure increased its value according to the evolution of software versions, indicating a smooth increasing of external dependencies of the package analyzed.

The 69% of Ca values was zero, indicating a high dependency from other packages to the package analyzed. So, the software presented a high coupling. A variation of Ca has a higher impact on instability value than a variation of Ce. In order for Ce to generate significant changes in the value of instability, the amplitude of its variation should be higher compared with Ca.

The main contributions of this research are: **i**) the literature review and identification of reference values for Ca and Ce proposed in papers; **ii**) the identification of market practices for Ca, Ce and instability metric values; **iii**) statistical analysis of data from market practices, involving descriptive statistics, comparison means, frequency distributions, evolution of Instability, Ca and Ce among versions.

Based on results of this paper, we conclude that software architects and engineers should concentrate more efforts to produce software containing low instability since first version, because the most of software keep the instability level through the new releases. More analysis is necessary to confirm this behavior about instability of software through time.

As future works, the authors state to be necessary to lead: **a**) more analysis to confirm the behavior of software instability through time; **b**) a correlation analysis between Ca, Ce and I compared to the frequency of errors and cost of errors in software; and **c**) a comparison of software instability among different instability measures. It is possible to check the differences values among them and what those differences actually represents, clearing differences and similarities.

Acknowledgements

The authors gratefully acknowledge CNPQ (Conselho Nacional de Desenvolvimento Científico e Tecnológico) and FAPEMIG (Fundação de Amparo a Pesquisa do Estado de Minas Gerais) for its financial support for Post-Doc and research projects, allowing the investigations.

References

[1] Martin, R. (1994, October). OO design quality metrics. An analysis of dependencies, 12. Retrieved October 27, 2014 from: http://www.cin.ufpe.br/~alt/mestrado/oodmetrc.pdf

[2] Chindamber S., & Kemerer C. F. (1991). Towards a metrics suite for object-oriented design. *Proceedings*

of the Conference on Object-oriented Programming Systems, Languages, and Applications.

[3] Al Dallal, J. (2013). Object-oriented class maintainability prediction using internal quality attributes. *Journal Information and Software Technology*. *55*, 2028-2048.

[4] Bavota, G., Lucia, A., Marcus, A., & Oliveto, R. (2013). Using structural and semantic measures to improve software modularization. *Journal Empirical Software Engineering. 18(5)*, 901-932.

[5] Chen, Z., Zhou, Y., Xu, B., Zhao, J., & Yang, H. (2002). A Novel approach to measuring class cohesion based on dependence analysis. *Proceedings of International Conference on IEEE*.

[6] Lanza M., & Marinescu, R. (2006). *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Springer Business Media.

[7] Tempero, H., Baxter, G., Noble, J., & Frean, M. (2006). Understanding the shape of java software. *ACM Sigplan Notices*.

[8] Fenton, N., & Neil, M. (2000). Software metric: Roadmap. *Proceedings of the Conference on the Future of Software Engineering Future Software Enginering*.

[9] ISO/IEC 25000. (2014). Systems and software engineering. Systems and software Quality Requirements and Evaluation (SQuaRE).

[10] Linde, K., & Willich, S. (2003). How objective are systematic reviews? Differences between reviews on complementary medicine. *Journal of the Royal Society of Medicine, 96(3)*, (156-157).

[11] Justus, R. (2009, June). A guide to writing the dissertation literature review. Pratical Assessment Research & Evaluation, 14(13). Retrieved June 13, 2014 from http://lemass.net/capstone/files/A%20Guide%20to%20Writing%20the%20Dissertation%20Literature%20Review.pdf

[12] Biolchhini, J. E. A. (2005). Systematic review in software engineering. *System Engineering and Computer Science Department COPPE/UFRJ*.

[13] Elish, M. O. (2010). Exploring the relationships between design metrics and package understandability: A case study. *Proceedings of the International Conference on Program Comprehension* (pp. 144-147). Braga, Portugal.

[14] Muhammed, A. M., Elish, M. O., & Al-Yafei, A. H. (2011). Empirical comparison of three metrics suites for fault prediction in packages of object-oriented systems: A case study of Eclipse. *Journal Advances in Engineering Software, 42(10)*, 852-859.

[15] Elish, M. O., Mohammad, A. M., & Al-Khiaty, M. (2011). Investigation of aspect-oriented metrics for stability assessment: A case study. *Journal of Software*, *6(12)*, 2508-2514.

[16] Ferreira, K. A. M., Mariza, A. S. B., Roberto, S. B., Luiz, F. O. M., & Heitor, C. A. (2012). Identifying thresholds for object-oriented software metrics. *Journal of System and Software, 85(2)*, 244-257.

[17] Kruskall, W. L., & Wallis, D. H. (1952). Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260), 583-621.

Danilo Batista dos Santos was born in 1992. He got a B.Sc. in information systems in 2014 from Federal University of Lavras (UFLA) at Brazil. He has experience in computer science, with emphasis on software engineering. He is acting on the following fields of study: software metric, measurement tools and software refactoring. Currently he is attending master in computer science from UFLA.

Antonio Maria P. de Resende was born in 1975. He holds a B.Sc. in informatic got in 1995 and a MSc and a DSc from Instituto Tecnologico de Aeronáutica (ITA-Brazil). He is a professor and former head of the Computer Science Department of Universidade Federal de Lavras (UFLA), Brazil. He is author of two books on aspect oriented programming and selecting software components. Currently, he coordinates the

Software Engineering Research Group at UFLA. His recent research focuses on object- and aspect-oriented software metrics, with a focus on reference values, statical analysis and quality protocols. He is also investigating diagnosis of problems in software systems using metrics, and using the results to prescribe improvements.

Eudes de Castro Lima received his B. Sc in information systems in 2011 and a MSc in computer science in 2014, both from Federal University of Lavras (UFLA) in Brazil. He has experience in computer science, with emphasis on software engineering, acting on the following topics: software metric, reference valures and measurement tools. He is currently a researcher at the Software Engineering Group (PQEs/UFLA).

André Pimenta Freire has a PhD in computer science from the University of York, UK (2013), an MSc in computer science from the University of São Paulo, São Carlos (2008) and a BSc in computer science from the University of São Paulo, São Carlos (2005). His research interests include human-computer interaction and empirical software engineering, especially focusing the development of user-centred techniques for design and evaluation of interactive systems.