Similar Characteristics of Internal Software Quality Attributes for Object-Oriented Open-Source Software Projects

Mariana Santos, Rodrigo Amador, Paulo Henrique de Souza Bermejo, Heitor Costa

DCC - UFLA - Lavras - MG - Brazil

`mariana@bsi.ufla.br, toluenotnt@gmail.com, bermejo@dcc.ufla.br heitor@dcc.ufla.br`

Abstract. *Organizations are becoming increasingly concerned about software quality. In object-oriented (OO) systems, quality is characterized by measurements of internal quality attributes. An efficient and proper method to analyze software quality in the absence of fault-prone or defective data labels is cluster analysis. The aim of this paper is to find similarities among project structures by measuring characteristics of internal software quality. In a sample of 150 open-source software systems, we evaluated software using macro and micro categories. Results obtained using cluster analysis indicated that some domains such as Graphics, Games, and Development tend to have similarities in specialization, abstraction, stability, and complexity. These results exploit the ability of OO software metrics to find similar behavior across domains. The results provide an immediate view of the trends and characteristics of internal software quality of Java systems that need to be addressed so that software systems can continue to be maintainable.*

1. Introduction

Software quality assurance is a vital component of software development [Seliya; Khoshgoftaar, 2007]. Organizations around the world are growing increasingly concerned about software quality. However, software quality assurance activities are costly, taking up more than 50% of the project's budget [Hildburn; Towhidnejad, 2002; Hamid; Hasan, 2010; Kannojia; Singh, 2013]. In this context, several alternatives have been studied to find efficient methods to obtain information on software quality.

Software quality can be defined as the characteristics of products or services that satisfy the explicit and implicit needs of stakeholders [ISO/IEC 25010, 2011]. When the organization's target is source code quality or its complexity, quality is defined as internal quality. Source code quality has a large impact on software quality and is essential to software maintainability [Plosch *et al*., 2007; Baggen *et al*., 2012]. One way to predict software quality is to evaluate key software attributes through software metrics [Kayarvizhy; Kanmani, 2011]. When metrics are specific, they can represent aspects and characteristics inherent to a programming technology, such as inheritance (object-oriented) [Tian *et al*., 2008]. One of the reasons that software development has evolved using object-oriented (OO) technology is the belief that the OO code has high quality and maintainability [Briand *et al*., 1997]. The classes of an OO project are expected to have such characteristics because they are compilation units where the source code is developed.

Despite the existence of several theories about what constitutes good OO design, only empirical studies on real system structure can provide tangible answers regarding a project's quality [Briand *et al*., 2000]. The aim of this study is to identify similar characteristics among project structures, considering their different domains through software metrics. The underlying assumption is that software components with similar attributes will have similar quality characteristics [Seliya; Khoshgoftaar, 2007]. Cluster analysis has proven to be a suitable method to analyze software quality without fault-prone or defective data labels [Zhong *et al*., 2004]. This analysis is expected to aid stakeholders in identifying needs and problems related to specific quality attributes of each project's domains or categories [McMillan *et al*.; 2011; Souza; Maia, 2013]. Thus, this study intends to answer the following research question:

Do the software domains have structural similarities with each other in aspects such as modularity, abstraction, stability, complexity, and specialization?

The paper is organized as follows: A brief theoretical background is provided in Section 2. The methodology used to analyze the data is explained in Section 3. The analysis of results is discussed in Section 4. Related work is presented in Section 5. Threats to validity are shown in Section 6. Finally, the conclusion and suggestions for future work are provided in Section 7.

2. Background

In this section, software domains, metrics used to characterize OO software, and cluster analysis are discussed.

2.1. Software Domains

Software can be classified into different categories, and content is an important factor in determining their application or domain. Content is related to the meaning and form of the information input and output [McMillan *et al*., 2011; Souza; Maia, 2013]. The development of significant generic domains for software applications is a difficult task; as software complexity grows, specific domain characteristics become unclear [Pressman, 2009]. Software can be classified into eight main domains [Pressman, 2009] (Table 1).

Table 1 Software Categories

Domain	Description
System Software (SS)	Collection of software to be used by other software (e.g., compilers, editors, and IDEs)
Real-Time Software (RTS)	Software to monitor, analyze, and control real-world events at the time they occur
Business Software (BS)	Software for information management to facilitate business operations and decision making
Engineering and Scientific Software (ESS)	Software characterized by conventional numerical algorithms
Embedded Software (ES)	Software that is read only by memory systems and used to control products and systems for industrial and consumer markets
Personal Computer Software (PCS)	Software used in personal daily tasks (e.g., word processing and spreadsheets)
Web-Based Software (WBS)	Software accessible on web pages by a browser with executable instructions and data
Artificial Intelligence Software (AIS)	Software developed with non-numerical algorithms for solving complex problems that cannot be solved by computation or simple analysis

This arrangement presents a macro view of the concept of application domains, which can cause confusion. For example, software in the **BS** domain, such as management systems and inventory control, can be software for desktop computers.

Another classification of software has been suggested by Sourceforge, the most popular open-source software repository. Sourceforge has approximately 3.4 million users (developers) registered, almost 324,000 projects, and almost 4,722 commits/day. Sourceforge's suggested classification is detailed, and it considers content and information. It groups software into 10 domains: Audio and Video (AV), Business and Enterprise (BE), Development (D), Games (G), Communication (C), Home and Education (HE), Graphics (GPH), Science and Engineering (SE), Security and Utilities (SU), and Systems Administration (SA).

We used both classifications to identify areas that have similar characteristics in a general way and a more specific way. The **macro categories** are the first type of classification, and the **micro categories** are the second type of classification.

2.2. Software Metrics

Software metrics are used to categorize and quantify qualitative data of software or its internal and external specification value [ISO/IEC 25010, 2011]. Their main function is to help software professionals plan and predict software development and control quality and effort (quantified) for developing software. In general, software metrics can be classified into software product and software process metrics [Li, 2000]. In this paper, only software product metrics are addressed.

At the end of the 1980s, several studies concluded that traditional software metrics were not sufficient to analyze and characterize OO software quality [Hamid; Hasan, 2010]. New metrics have been proposed to evaluate the structural quality of OO code [Chidamber; Kemerer, 1991; Li; Henry, 1993; Chidamber; Kemerer, 1994; Lorenz; Kidd, 1994; Bieman; Kang, 1995; Hitz; Montazeri, 1995; Lee *et al*., 1995; Briand *et al*., 1997]. Such metrics are intended to provide a way to assess internal software quality.

The first set of metrics for OO software was proposed by Chidamber and Kemerer (CK) [Chidamber; Kemerer, 1991; Chidamber; Kemerer, 1994]. This set is composed of six metrics: *weighted method per class* (WMC), *coupling between objects* (CBO), *response for class* (RFC), *lack of cohesion of methods* (LCOM), *depth in inheritance tree* (DIT), and *number of children* (NOC). These metrics reflect some OO mechanisms such as inheritance, association, aggregation, and polymorphism, but not all [Prasad; Nagar, 2009]. Thus, new metrics were proposed to complement the CK suite. Since a large number of software metrics are available in the literature, the CK suite and metrics proposed by McCabe (1976) (Cyclomatic Complexity), Li and Henry (1993), Lorenz and Kidd (1994), Bieman and Kang (1995), and Martin and Martin (2006) were used in this study.

2.3. Cluster Analysis

Several studies have used clustering to predict and identify software characteristics [Yang *et al*., 2006; Shanthin; Chandrasekaran, 2012]. This multivariate technique consists of finding groups that have similar elements and that are different from other groups of elements according to some similarity measures [Jiawei; Micheline, 2011]. This technique has four phases [Hair *et al*.; 2009, Muhammad *et al*., 2012] (Table 2).

Cluster analysis was performed using Weka 3.6.10, an analytical tool developed by the University of Waikato. Statistical tests were conducted using four clustering algorithms: i) K-means with the Euclidean similarity function (KM-E) [MacQueen, 1967], ii) K-means with the Manhattan similarity function (KM-M) [Vaidya; Clifton, 2003], iii) Expectation-Maximization (EM) clustering algorithm [Zhang *et al*., 1999], and iv) hierarchical clustering. These algorithms generated their own clusters using the same dataset. The produced results were then interpreted.

Table 2 Phases of Cluster Analysis

Phase	Description
Selection of entities	The first step is to select entities to be grouped. In this study, 150 software systems were used and separated into macro and micro categories.
Selection of grouping attributes	Characteristics/attributes of entities that can be grouped are identified based on their similarity. In this case, the goal is to capture structural similarities/differences in source code using metrics as parameters that characterize internal quality factors.
Selection of clustering algorithm	As the initial research goal is to find similarities among software domains, clustering algorithms are used, particularly K-means (most common algorithm) and hierarchical clustering.
Data interpretation	A clustering solution that produces the lowest possible classification error is found. If the initial solution is likely to be a subject of optimization, the number of clusters should be reconsidered and the whole process should be repeated.

3. Methodology

In this section, the study design, data collection procedures, study variables, and hypotheses are described.

3.1. Study Design

The sample used was formed by open source Java software selected from two of the most popular repositories on the web: Github (https://github.com/) and Sourceforge (http://sourceforge.net/). The selection criteria were software with more than 50 stars (marked as a favorite by users) for Github and an assessment of over 3.5 stars for Sourceforge. These criteria were essential to eliminate “toy projects,” which can be described as personal software without adequate standardization. We collected 150 valid software systems for analysis, with a total of 12,178,587 lines of code and 69,334 classes. These software systems were separated into macro and micro categories.

Software systems selected from Github were arranged into micro categories based on the description of the system, provided on its web page, since the repository does not categorize software by domains. For example, if the project description is an API for application development on the Android platform, then it belongs to the ***D*** domain. Software systems were classified into macro categories based on the description of each domain. The software systems were compatible with only six domains. The STR and SE domains were discarded from the analysis.

Internal quality characteristics are the dependent variable (variable to be explained [Hair *et al*., 2009]) in this study. The dependent variable is affected by changes in the independent variable [Hair *et al*., 2009]. Coupling, inheritance, complexity, cohesion, abstraction, and size metrics are the independent variables (Table 3) in this study. To extract measures from the source code, we used the tools listed in Table 4.

3.2. Hypotheses

In this study, we tested the hypotheses (Table 5) based on the definition of each measure to answer questions about the design and similarity between projects. When software domains are considered, some metrics can have a few differences or significant variations from project to project. The hypotheses formulated will help determine which metrics are relevant to a domain. Those that do not differentiate classes very well or fail to build well-defined clusters are unlikely to be useful predictors of similar characteristics in the study dataset. In positive cases, they should be in the same clusters.

Table 3 Metrics Used

#	Metric	Acronym	Type	Description	Reference
1	McCabe Cyclomatic Complexity	VG	Complexity	Number of linearly independent paths in the code	McCabe (1976)
2	Weighted methods per class	WMC	Complexity	Sum of the methods complexities defined in a class	Chidamber; Kemerer (1991; 1994)
3	Number of overridden methods	NOVM	Inheritance	Number of overridden methods of a class	Lorenz; Kidd (1994)
4	Number of children	NOC	Inheritance	Number of classes that directly inherit from a given class	Chidamber; Kemerer (1991; 1994)
5	Depth of inheritance tree	DIT	Inheritance	Size of the longest path from a class to the root in the project hierarchy	Chidamber; Kemerer (1991; 1994)
6	Specialization index	SIX	Inheritance	(NOVM*DIT)/number of methods	Lorenz; Kidd (1994)
7	Lack of cohesion of methods	LCOM	Cohesion	Number of pairs of methods in a class using no attribute in common	Chidamber; Kemerer (1991; 1994)
8	Tight class cohesion	TCC	Cohesion	Percentage of pairs of public methods of a class that are connected, i.e., directly or indirectly used as attributes.	Bieman; Kang (1995)
9	Afferent coupling	CA	Coupling	Number of packages in other classes that rely inside the package. It is an indicator of package responsibility	Martin; Martin (2006)
10	Efferent coupling	CE	Coupling	Number of other packages that the classes in the package depend	Martin; Martin (2006)
11	Instability	RMI	Coupling	Indicator of the resilience of the package changes. Defined as I = CE / (EC + CA)	Martin; Martin (2006)
12	Coupling between objects	CBO	Coupling	Number of classes that a class is referenced plus number of classes referencing the class.	Chidamber; Kemerer (1991; 1994)
13	Response for class	RFC	Coupling	Number of methods in the class (not including methods inherited) plus number of distinct methods called by methods of the class.	Chidamber; Kemerer (1991; 1994)
14	Message passing coupling	MPC	Coupling	Number of invocations in a class	Li; Henry (1993)
15	Data abstraction coupling	DAC	Coupling	Number of attributes in a class that has another class as type	Li; Henry (1993)
16	Abstractness	RMA	Abstraction	Ratio between the number of abstract classes (and interfaces) and the number of classes in the package.	Martin; Martin (2006)
17	Lines of code	LOC	Size	Number of lines in a class	-
18	Number of classes	NC	Size	Number of classes in a project	-
19	Number of methods	NOM	Size	Number of methods in a class	-
20	Number of attributes	NOA	Size	Number of attributes in a class	-

Table 4 Tools Used

Tools	Metrics
Eclipse Metrics	VG, WMC, NOVM, NOC, DIT, SIX, LCOM, CA, CE, RMI, RMA, NC, NOM, NOA
Vizz Maintenance	CBO, RFC, MPC, DAC, TCC, LOC

4. Results

In this section, we describe how analyses were performed and what algorithms were used to create the set of clusters and results, considering the algorithm with the lowest error rate. To perform the clustering analysis, we used KM-E, KM-M, EM, and hierarchical clustering. The number of clusters generated was four, since the results for these clusters produced the lowest error rate. Table 6 shows the error rate of the clustering algorithms for the proposed categories, considering all metrics in the dataset. The best result (lowest error rate) is highlighted in bold. KM-E, KM-M, and EM achieved the best performance based on the average error rate.

If a group of variables in a dataset can produce heterogeneous clusters, these variables are likely to measure similarity between the objects to be tested. Three algorithms had a similar average error. In this case, the algorithm chosen was the one with the lowest error rate in each analysis performed on all combinations of pairs of metrics. Results of the analyses will be shown using the algorithm that obtained the lowest error for each approach (macro and micro categories).

Table 5 Hypotheses

Hypothesis	Description
H-Coupling and cohesion (CA, CE, RFC, CBO, DAC, MPC, TCC, and LCOM)	**H_0:** Software domains that have classes with high coupling and low cohesion are more likely to have dissimilar characteristics to software domains that have classes with low coupling and high cohesion. **H_1:** Metrics selected are not capable of identifying characteristics among domains in relation to coupling and cohesion. This means that the behavior on coupling and cohesion are similar for all domains. The hypothesis is partially validated if a pair of metrics presents the expected behavior.
H-Complexity and inheritance (VG, WMC, NOC, and DIT)	**H_0:** Software domains that have a larger inheritance hierarchy and more complex classes are more likely to have dissimilar characteristics to software domains that have a smaller inheritance hierarchy and less complex classes. **H_1:** Metrics selected are not capable of identifying characteristics among domains in relation to complexity and inheritance. This means that the behavior on complexity and inheritance are similar for all domains. The hypothesis is partially validated if a pair of metrics presents the expected behavior.
H-Complexity and size (VG, WMC, LOC, NOM, NC, and NOA)	**H_0:** Software domains that have a larger size and more complex classes are more likely to have dissimilar characteristics to software domains that have less complex classes. **H_1:** Metrics selected are not capable of identifying characteristics among domains in relation to complexity and size. This means that the behavior on complexity and size are similar for all domains. The hypothesis is partially validated if a pair of metrics presents the expected behavior.
H-Depth and descendants (DIT, NOC, RMI, and RMA)	**H_0:** Software domains that have classes located deeper in the inheritance hierarchy (less abstract) are more likely to have dissimilar characteristics to software domains that have less deep classes in the inheritance hierarchy (more abstract). **H_1:** Metrics selected are not capable of identifying characteristics among domains in relation to inheritance and abstraction. This means that the behavior on inheritance and abstraction are similar for all domains. The hypothesis is partially validated if a pair of metrics presents the expected behavior.
H-Complexity and overriding (NOVM, SIX, WMC, and VG)	**H_0:** The more overriding methods are used, the more complex it is to understand or test the class. Software domains that have more classes with overridden methods are more likely to have dissimilar characteristics to software domains with fewer classes with overridden methods. **H_1:** Metrics selected are not capable of identifying characteristics among domains in relation to complexity and inheritance. This means that the behavior on complexity and inheritance are similar for all domains. The hypothesis is partially validated if a pair of metrics presents the behavior expected.
H-Abstraction and stability (RMI and RMA)	**H_0:** Software domains that have more stable and abstract classes are more likely to have dissimilar characteristics to software domains that have less abstract and unstable classes. **H_1:** Metrics selected are not capable of identifying characteristics among domains in relation to abstraction and coupling. This means that the behavior on abstraction and coupling are similar for all domains. Since these are just two metrics, there is no partial validation for this hypothesis.

Table 6 Error Rate (%) Obtained by the Algorithms

Database	KM-E	KM-M	EM	HC
Micro	**82%**	83%	**82%**	87%
Macro	60%	**59%**	60%	58%
Average error	**71%**	**71%**	**71%**	73%

4.1. Micro Categories

In this subsection, we describe the analysis results for micro categories according to the five defined research hypotheses. For the analysis of micro categories, the KM-M algorithm obtained the lowest error.

- **H1-Coupling and cohesion (CA, CE, RFC, CBO, DAC, MPC, TCC, and LCOM):** Using coupling and cohesion metrics as parameters for classifying clusters, none of the algorithms formed heterogeneous clusters. This indicates that these metrics are common and not relevant to finding similarities among projects.
- **H2-Complexity and inheritance (VG, WMC, NOC, and DIT):** Using complexity and inheritance metrics as parameters for classifying clusters, none of the algorithms formed heterogeneous clusters. This indicates that these metrics are common and not relevant to finding similarities among projects.
- **H3-Depth and descendants (DIT, NOC, RMI, and RMA):** Using inheritance,

abstraction, and coupling metrics as parameters for classifying clusters in pairs, only the pair DIT (Axis X) and RMI (Axis Y) showed good results with an error rate of 82.7%, where Axis X is composed of the interval values measured for DIT and Axis Y is composed of the interval values measured for RMI (Figure 1). This result partially validates the hypothesis. Table 7 shows the different characteristics for each cluster: The ***SE*** domain is predominant in Cluster 0 (dark blue), the ***SU*** and ***GPH*** domains are predominant in Cluster 1 (red), the ***D*** and ***C*** domains are predominant in Cluster 2 (green), and the ***G*** and ***SA*** domains are predominant in Cluster 3 (light blue). Cluster 2 has elements with medium values for RMI and the lowest values for DIT in relation to the other three clusters. The behavior of Cluster 1 in the inheritance attribute is inversely related to that of Cluster 2, with higher values for DIT. This result indicates that the ***SU*** and ***GPH*** domains tend to have few abstract classes (more stability) and few descendants. On the other hand, the ***D*** and ***C*** domains tend to have a higher average number of descendants. The ***SE*** and ***BE*** domains seem to have a more balanced relationship between abstraction and inheritance. An analysis of Cluster 4 shows that most of the objects have an RMI close to 1, meaning the systems in the ***G*** and ***SA*** domains have more stable classes than the other clusters do. The existence of more abstract classes and interfaces allows the growth of direct descendants. Thus, the ratio between the number of abstract classes in the package and the number of classes in the package tends to be higher.

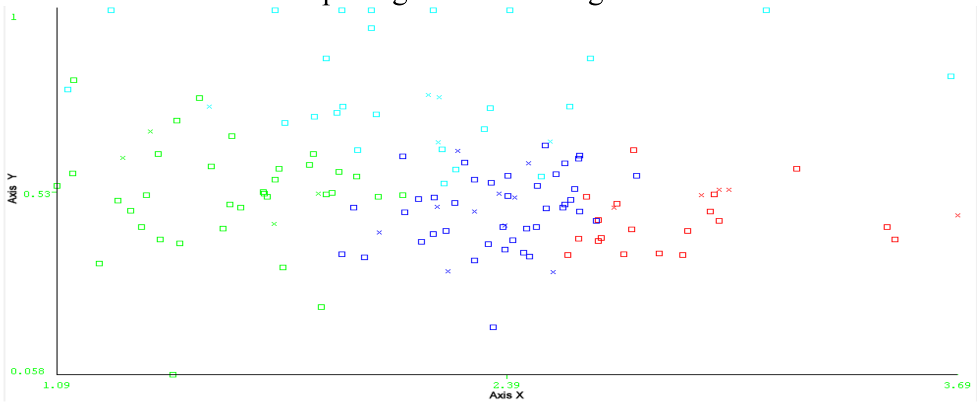

Figure 1 Four-Cluster Solution with DIT (Axis X) and RMI (Axis Y)

Table 7 Four-Cluster Solution Obtained with DIT and RMI

Cluster	G	AV	D	C	BE	GPH	HE	SE	SU	SA
0	4	4	4	3	8	6	6	10	4	2
1	1	3	1	3	3	4	3	-	5	3
2	5	4	6	5	2	4	3	2	4	5
3	5	4	4	4	2	1	3	3	2	5

- **H4-Complexity and size (VG, WMC, LOC, NOM, NC, and NOA):** Using complexity and size metrics as parameters for classifying clusters in pairs, WMC (Axis X) and LOC (Axis Y) showed good results with an error rate of 79.3%, where Axis X is composed of the interval values measured for WMC and Axis Y is composed of the interval values measured for LOC (Figure 2). This result partially validates the hypothesis. Table 8 shows the different characteristics of each cluster: The ***SE*** and ***GPH*** domains are predominant in Cluster 0 (dark blue), the ***SU*** domain is predominant in Cluster 1 (red), the ***BE*** domain is predominant in Cluster 2 (green), and the ***G***, ***D***, ***C***, and ***SA*** domains are predominant in Cluster 3 (light blue). Clusters 1 and 2 have elements with medium values for WMC and the lowest values for LOC; Cluster 1 has lower values than Cluster 2. This result indicates that systems in the ***SU*** and ***BE*** domains tend to have few lines of code, making the code less complex. Systems in the ***G***, ***D***, ***C***, and ***SA*** domains tend to have the lowest average for both metrics. On the other hand, systems in the ***GPH*** and ***SE*** domains seem to have more lines of code, but only some objects have higher values for WMC. The hypothesis that bigger classes have more complex software cannot be validated in this analysis.

- **H5-Complexity and overriding (NOVM, SIX, WMC, and VG):** Using complexity and inheritance metrics as parameters for classifying clusters, none of the algorithms formed heterogeneous clusters. This indicates that these metrics are common and not relevant to finding similarities among projects.

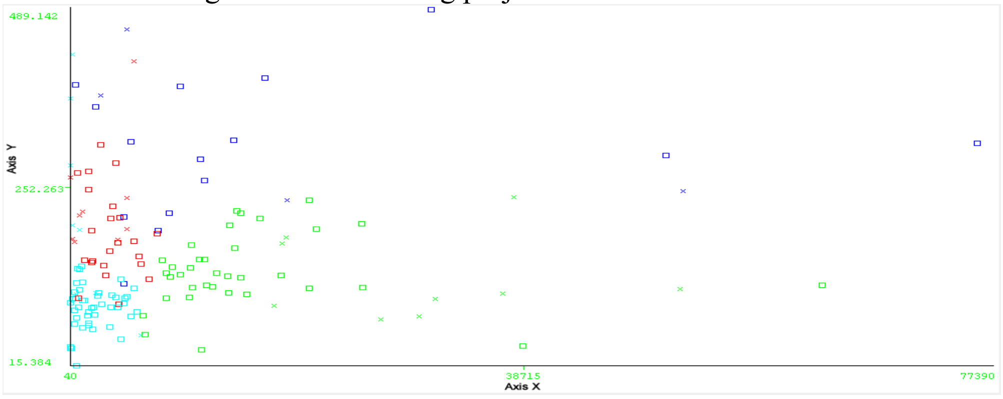

Figure 2 Four-Cluster Solution with WMC (Axis X) and LOC (Axis Y)

Table 8 Four-Cluster Solution Obtained with WMC and LOC

Cluster	G	AV	D	C	BE	GPH	HE	SE	SU	SA
0	1	3	2	1	2	4	2	4	-	-
1	1	3	2	4	2	2	4	3	9	3
2	4	6	3	2	9	3	6	4	4	3
3	9	3	8	8	2	6	3	4	2	9

- **H6-Abstraction and stability (RMI and RMA):** Using RMI and RMA as parameters for classifying clusters in pairs, the results (84%) supported this hypothesis. Figure 3 shows the cluster solution, where Axis X is composed of the interval values measured for RMA and Axis Y is composed of the interval values measured for RMI. Table 9 shows the different characteristics for each cluster: The ***G*** domain is predominant in Cluster 0 (dark blue), the ***HE*** and ***SE*** domains are predominant in Cluster 1 (red), and the ***SU***, ***SA***, ***D***, and ***BE*** domains are predominant in Cluster 2 (green). The ***G*** and ***SA*** domains tend to group elements in the same clusters. This behavior can be seen in Clusters 0 and 3. In Figure 3, Cluster 3 has high values for RMI and low values for RMA. Most of the elements of Clusters 0, 1, and 2 have average values for RMI, but only Cluster 0 has objects with values close to the medium value for RMA, based on metric results. This result indicates that the ***G*** domain tends to balance abstraction and stability better than other domains. Therefore, systems in the ***G*** domain, which appear in the ***SA*** domain in Cluster 3, have higher values for RMI. This arrangement indicates that classes of the systems in this domain tend to be very rigid, cannot be extended (not abstract), and can handle only small changes to remain stable. These indications are not ideal when software quality is the focus, because maintenance is difficult. If a line is drawn between the points [1, 1] (Figure 4), we can define the principal sequence. The points near this line are not abstract for their stability or unstable for their abstraction. This behavior is ideal for ensuring system quality and maintainability.

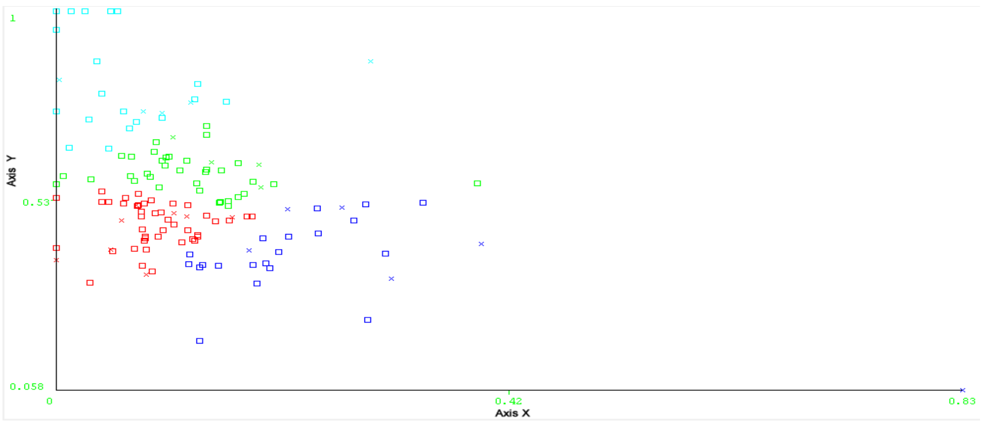

Figure 3 Four-Cluster Solution with RMA (Axis X) and RMI (Axis Y)

Table 9 Four-Cluster Solution Obtained with RMA and RMI

Cluster	G	AV	D	C	BE	GPH	HE	SE	SU	SA
0	6	3	3	3	4	4	1	1	2	1
1	1	5	3	6	5	7	8	8	6	4
2	3	3	5	4	5	3	4	3	5	5
3	5	4	4	2	1	1	2	3	2	5

4.2. Macro Categories

Results of the analysis using KM-E for macro categories are as follows:

- **H1-Coupling and cohesion (CA, CE, RFC, CBO, DAC, MPC, TCC, and LCOM):** Using coupling and cohesion metrics as parameters for classifying clusters, none of the algorithms formed heterogeneous clusters. This indicates that the values of these metrics are common and not relevant to identifying similarities among projects.

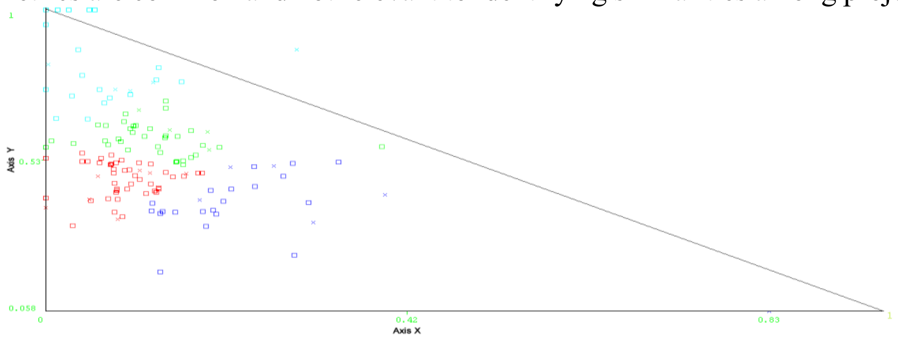

Figure 4 Dispersion of Elements in Principal Sequence for Micro-Category Analysis

- **H2-Complexity and inheritance (VG, WMC, NOC, and DIT):** Using complexity and inheritance metrics as parameters for classifying clusters, none of the algorithms formed heterogeneous clusters. This indicates that the values of these metrics are common and not relevant to identifying similarities among projects.
- **H3-Depth and descendants (DIT, NOC, RMI, and RMA):** Using inheritance, abstraction, and coupling metrics as parameters for classifying clusters in pairs, DIT (Axis X) and RMI (Axis Y) showed good results with an error rate of 68.5%, where Axis X is composed of the interval values measured for DIT and Axis Y is composed

of the interval values measured for RMI (Figure 5). This result partially validates the hypothesis. Table 10 shows the different characteristics for each cluster: The ***PCS*** domain is predominant in all clusters; the ***SS*** domain is predominant in Cluster 0 (dark blue), Cluster 1 (red), and Cluster 2 (green); and the ***ESS*** domain is predominant in Cluster 3 (light blue). Despite having a smaller error for the same analysis for micro categories, this solution is less heterogeneous (Figure 5). The existence of more abstract classes and interfaces allows the growth of direct descendants. Regarding RMI concept, the ratio between the number of abstract classes in the package and the number of classes in the package tends to be higher. Systems in the ***PCS*** domain (composed of the ***AV***, ***C***, and ***GPH*** domains) tend to have more descendants and average stability in comparison with other systems of the same domain and systems of different domains. The SS domain has similar behavior those micro-categories, and is mainly composed of the D domain.

- **H4-Complexity and size (VG, WMC, LOC, NOM, NC, and NOA):** Using complexity and size metrics as parameters for classifying clusters in pairs, WMC (Axis X) and LOC (Axis Y) showed good results with an error rate of 61.1%, where Axis X is composed of the interval values measured for WMC and Axis Y is composed of the interval values measured for LOC (Figure 6). This result partially validates the hypothesis. Table 11 shows the different characteristics for each cluster: The ***PCS*** domain is predominant in Cluster 0 (dark blue) and Cluster 3 (light blue), and the ***BS*** domain is predominant in Cluster 1 (red) and Cluster 2 (green). This solution is less heterogeneous than the same solution in micro categories (Figure 6). The ***PCS*** domain is composed of the ***AV***, ***C***, and ***GPH*** domains. Some systems (21 systems) in the ***PCS*** domain have higher complexity and a higher average number of lines of code than other domains. The ***BS*** domain has more lines of code than the other domains, so it is the most extensive.
- **H5-Complexity and overriding (NOVM, SIX, WMC, and VG):** Using complexity and inheritance metrics as parameters for classifying clusters, none of the algorithms formed heterogeneous clusters. This indicates that the values of these metrics are common and not relevant to identifying similarities among projects.

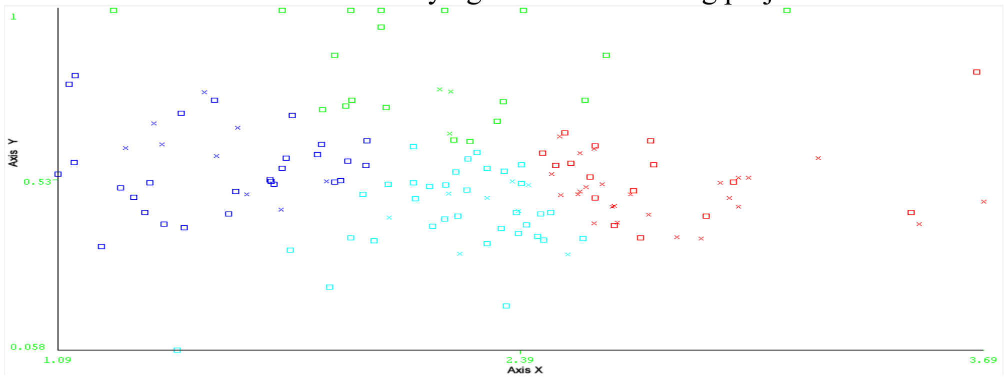

Figure 5 Four-Cluster Solution with DIT (Axis X) and RMI (Axis Y)

Table 10 Four-Cluster Solution Obtained with DIT and RMI

Cluster	SS	BS	ESS	PCS	WBS	AIS
0	11	2	3	18	1	4
1	7	6	1	25	2	2
2	7	1	3	10	-	3
3	4	6	8	18	1	6

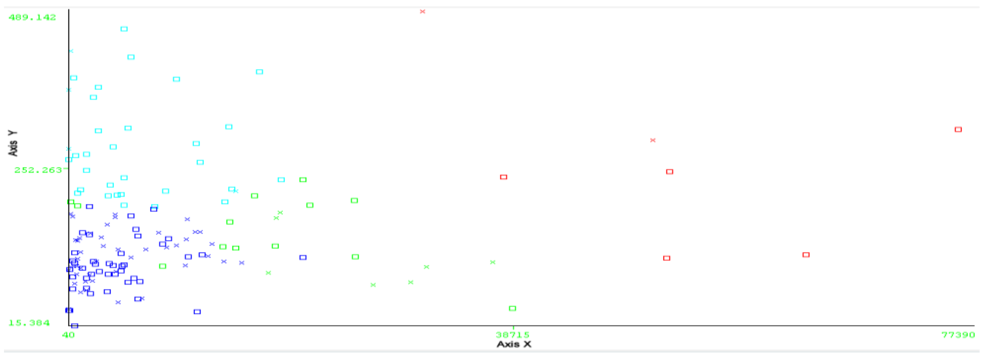

Figure 6 Four-Cluster Solution with WMC (Axis X) and LOC (Axis Y)

Table 11 Four-Cluster Solution Obtained with WMC and LOC

Cluster	SS	BS	ESS	PCS	WBS	AIS
0	24	3	7	45	2	7
1	1	3	2	1	-	-
2	1	7	2	4	2	4
3	3	2	4	21	-	4

- **H6-Abstraction and stability (RMI and RMA):** Using RMI and RMA as parameters for classifying clusters in pairs, the results (59.1%) supported this hypothesis, where Axis X is composed of the interval values measured for RMA and Axis Y is composed of the interval values measured for RMI. Table 12 shows the different characteristics for each cluster: The ***SS*** domain is predominant in Cluster 0 (red), and the ***PCS*** domain is predominant in Cluster 1 (red). Despite having a smaller error for the same analysis for micro categories, this solution is less heterogeneous. In Figure 7, Cluster 3 (light blue) has average values for RMA and RMI. Although Cluster 3 has average results, these results are higher than those of the other clusters. Some of the elements of Cluster 2 (green) have values near 0 for RMA, indicating that some systems in the ***PCS*** domain have more abstract classes. If a line is drawn between the points [1, 1] (Figure 8), we can define the principal sequence. The points near this line are not abstract for their stability or unstable for their abstraction. This behavior is ideal for ensuring system quality and maintainability.

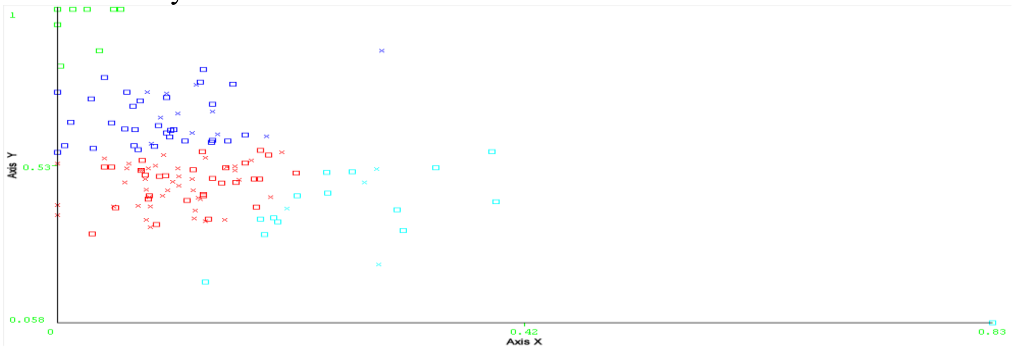

Figure 7 Four-Cluster Solution with RMA (Axis X) and RMI (Axis Y)

Table 12 Four-Cluster Solution with RMA and RMI

Cluster	SS	BS	ESS	PCS	WBS	AIS
0	15	3	6	12	2	8
1	10	8	8	45	2	3

2	2	-	-	8	-	-
3	2	4	1	9	-	4

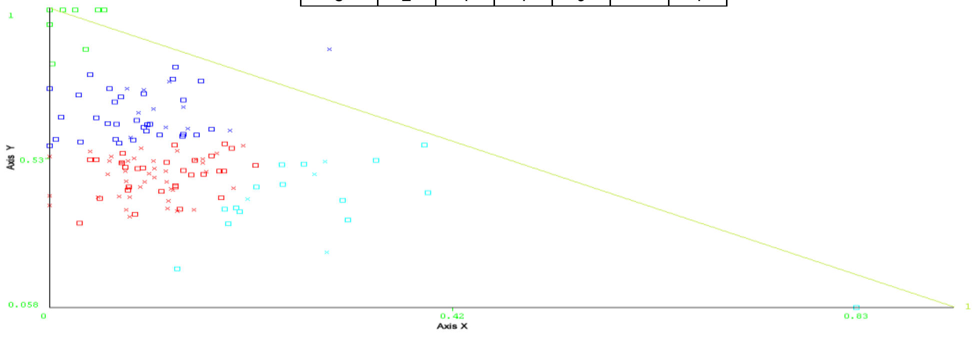

Figure 8 Dispersion of Elements in Principal Sequence for Macro-Category Analysis

4.3. Discussion

Regarding *H-Depth and descendants*, the results show that systems in the ***SU*** and ***GPH*** domains tend to have few abstract classes (more stability) and few descendants. The ***D*** and ***C*** domains tend to have a higher average number of descendants. For instance, systems in the ***SE*** and ***BE*** domains seem to have a more balanced relationship between abstraction and inheritance. Systems in the ***G*** and ***SA*** domains have more stable classes. However, this assumption may be valid partly because the metrics that produced heterogeneous clusters were the DIT x RMI combination. Thus, domains with classes located deeper in the inheritance hierarchy (less abstract), such as the ***D*** and ***C*** domains, are more likely to have dissimilar characteristics to software domains that have classes located less deep in the inheritance hierarchy (more abstract), such as the ***G***, ***SA***, ***BE***, and ***SE*** domain.

Regarding *H-Abstraction and stability* for micro categories, systems in the ***G*** and ***SA*** domains tend to have highly stable, concrete classes that are less prone to change, which may make the task of code maintenance of the systems difficult, directly affecting the internal quality. In addition, systems in the ***G*** domain tend to balance abstraction and stability better than the other domains. The idea is to design OO software to respect the Stable Abstractions Principle (SAP), in which a component is not too abstract or rigid to hinder changes. For macro categories, the result was partly similar, since part of the systems in the ***SA*** domain are classified as ***SS*** and ***PCS*** domains. The ***G*** domain composes the ***AIS*** domain. This hypothesis can be fully validated because RMI and RMA produced heterogeneous clusters for micro and macro categories. The behaviors analyzed in this study indicate that more efforts in these aspects are necessary so that systems can continue to be maintainable. However, the study has biases that we can consider threats to the validity of the results.

5. Related Work

The study of Dallal [2013] analyzed the relationship between internal quality attributes (size, cohesion, and coupling) and external quality attributes (maintainability in classes). Using statistical techniques, models were constructed using internal attributes (based on metric combinations) to predict maintainability in selected classes. The results indicate that developers can increase the maintainability (i.e., reduce maintenance efforts and costs) of their classes by reducing the size and coupling and increasing cohesion. However, the number of metrics and the data collected from software projects were limited. In addition, the study used three software systems from three different domains, making it difficult to generalize the study results.

Another study [Romano; Pinzger, 2011] aimed to investigate changes in Java interfaces and check if an interface has an impact on change in concrete and abstract classes. The authors found that the Service Interface Usage Cohesion (SIUC) metric (adapted to the OO concept) has a strong correlation with interface metrics than CK metrics. A third study [Malviya; Yadav, 2012] used the clustering technique for data mining projects to find OO sustainable systems (with high capacity for maintainability) using the K-means algorithm. The study covered 71 classes and 11 metrics including size metrics and the CK suite. Three clusters were found; the largest population of classes was grouped in the third cluster, which means the grouped classes were similar. However, the study was performed on one software system developed by Ada, so it may not represent the population of systems in the current IT industry. Authors of another study [Souza; Maia, 2013] proposed some reference values for a set of coupling metrics, considering software domains and evaluating 100 systems from the Sourcerer repository. The results showed that using reference values for systems from different domains may not be suitable to detect some characteristics.

This study differs from the aforementioned ones because it proposes a model that explains the similarity among domains in OO internal software quality. It provides a more immediate view of the trends and characteristics of internal Java software quality.

6. Threads to Validity

In terms of construct validity, which is the degree to which the variables under study can accurately measure the concepts they purport to measure, some of the metrics selected could have characteristics validated by statistical tests, such as WMC, DIT, LOC, RMI, and RMA. The results indicate that these metrics are relevant to analyzing the project similarity (considering domains) and assessing internal software quality. However, for other metrics, it is possible that the clustering technique is not sufficient to completely validate and detect characteristics inherent in OO Java software projects. Regarding internal validity, the results demonstrate empirical evidence and the practical significance of the project similarities, but they do not provide in-depth technical details of the projects. Such details would require qualitative analysis. For example, a code inspection in classes of the SU, GPH, D, and C domains and problems with inheritance, modularity, abstraction, and complexity could be shown qualitatively. As a threat to external validity, the study analyzes only OO software developed in Java.

7. Conclusion

The study aims to find similarities among structures of projects through metrics that characterize the internal software quality. Statistical tests were performed on a sample of 150 projects classified into macro and micro categories to identify which of these classifications provide better information on projects. Regarding the research question "*Do the software domains have structural similarities with each other in aspects such as modularity, abstraction, stability, complexity, and specialization,*" the results indicate that some specific domains tend to have similarities relating to four properties.

Systems in the ***SU*** and ***GPH*** domains have few descendants and few abstract classes. Software in the ***D*** and ***C*** domains can have similar characteristics in inheritance and abstraction, with a higher average number of descendants. Software in the ***SE*** and ***BE*** domains tend to make good use of inheritance. Software in the ***G*** and ***SA*** domains can have more stable classes and are harder to maintain due to the lack of flexibility in changes. Some software in the ***G*** and ***SA*** domains tend to have highly stable and concrete classes, which cannot be extended. Some software in the ***G*** domain tends to balance abstraction and stability better than the other domains.

This study contributes to the field of software engineering through quantitative techniques that provide observations of structural aspects of OO development, such as specialization, stability, abstraction, and complexity. In addition, the study identifies metrics such as WMC, DIT, LOC, RMI, and RMA that are relevant to the characterization of Java internal software quality. For software developers, the study shows that some domains such as ***GPH***, ***G***, and ***D*** tend to have the same characteristics and that more efforts in these aspects are necessary so that systems can continue to be maintainable. For future work, we suggest repeating the analyses on a larger sample of software, using other repositories of available projects and other metrics of OO software to obtain new results on characteristics that have not yet been explored.

References

Baggen, R.; Correia, J. P.; Schill, K.; Visser, J. (2012) Standardized Code Quality Benchmarking for Improving Software Maintainability. Software Quality Control 20, 2, pp. 287-307.

Bieman, J.; Kang, B. (1995) Cohesion and Reuse in an Object-Oriented System. In: ACM Symposium on Software Reusability. pp. 259-262

Briand, L. C.; Bunse, C.; Daly, J. W.; Differing, C. (1997) An Experimental Comparison of the Maintainability of Object-Oriented and Structured Design Documents. In: Empirical Software Engineering. pp. 291-312.

Briand, L. C.; Wüst, J.; Daly, J. W.; Porter, V. D. (2000) Exploring the Relationship Between Design Measures and Software Quality in Object-Oriented Systems. In: Journal of Systems and Software, vol. 51, no. 3, May 2000, pp. 245-273.

Chidamber, S.; Kemerer, C. (1991) Towards a Metrics Suite for Object-Oriented Design. In: Conference on Object-Oriented Programming: Systems, Languages and Applications. Published in SIGPLAN Notices 26 (11), pp. 197-211.

Chidamber, S.; Kemerer, C. (1994) A Metrics Suite for Object-Oriented Design. In: Transactions on Software Engineering 20 (6), pp. 476-493.

Dallal, J. A. (2013) Object-Oriented Class Maintainability Prediction Using Internal Quality Attributes. In: Inf. Software Technology 55, 11. pp. 2028-2048.

Hair, J. F., Black, W. C., Babin, B. J., Anderson, R. E., & Tatham, R. L. (2009) Multivariate Data Analysis, 6th edition, Bookman, 688p.

Hamid, N. F. I. A.; Hasan, M. K. (2010) Industrial-Based Object-Oriented Software Quality Measurement System and Its Importance. In: International Symposium in Information Technology, vol.3, no., pp.1332-1336.

Hitz, M.; Montazeri, B. (1995) Measuring Coupling and Cohesion in Object-Oriented Systems. In: International Symposium on Applied Corporate Computing.

ISO/IEC 25010 (2011) Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation - System and Software Quality Models.

Jiawei, H.; Micheline, K. (2011) Data Mining, Concepts and Techniques. Morgan Kaufmann Publishers. 744p.

Kayarvizhy, N.; Kanmani, S. (2011) Analysis of Quality of Object Oriented Systems Using Object Oriented Metrics. In: International Conf. on Electronics Computer Technology, pp.203-206.

Lee, Y.; Liang, B.; Wu, S.; Wang, F. (1995) Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow. In: International Conference on Software Quality.

Li, W. (2000) Software Product Metrics. In: Potentials. vol.18, no.5, pp. 24-27.

Li, W.; Henry, S. (1993) Object-Oriented Metrics that Predict Maintainability. In: Journal of Systems and Software 23 (2), pp. 111-122.

Lorenz, M.; Kidd, J. (1994) Object-Oriented Software Metrics. Prentice Hall Object-Oriented Series, Englewood Cliffs. 146p.

MacQueen, J. B. (1967) Some Methods for Classification and Analysis of Multivariate Observations. In: Symposium on Mathematical Statistics and Probability. pp.281-297.

Malviya, A. K.; Yadav, V. K. (2012) Maintenance Activities in Object Oriented Software Systems Using K-Means Clustering Technique: A Review. In: Sixth International Conference on Software Engineering, pp. 1-5.

Martin, R.C.; Martin, M. (2006) Agile Principles, Patterns, and Practices in C#. Prentice Hall. 768p.

McCabe, T. J. (1976) A Complexity Measure. In: Trans. on Sw. Engineering, no.4, pp.308-320.

McMillan, C.; Vasquez, M. L.; Poshyvanyk, D.; Grechanik, M. (2011) Categorizing Software Applications for Maintenance. In: International Conf. on Software Maintenance pp.343-352.

Plosch, R.; Gruber, H.; Hentschel, A.; Korner, C.; Pomberger, G.; Schiffer, S.; Saft, M.; Storck, S. (2007) The EMISQ Method - Expert Based Evaluation of Internal Software Quality. In: Software Engineering Workshop, pp. 99-108.

Pressman. R. S. (2009) Software Engineering. McGraw-Hill. 928p.

Romano, D.; Pinzger, M. (2011) Using Source Code Metrics to Predict Change-Prone Java Interfaces. In: International Conference on Software Maintenance, pp. 303-312.

Seliya, N.; Khoshgoftaar, T. M. (2007) Software Quality Analysis of Unlabeled Program Modules with Semi supervised Clustering. In: Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, vol.37, no.2, pp. 201-211.

Shanthin, A.; Chandrasekaran, R. M. (2012) Applying Machine Learning for Fault Prediction Using Software Metrics. In: International Journal of Advanced Research in Computer Science and Software Engineering

Singh, B.; Kannojia, S. P. (2013) A Review on Software Quality Models. In: Intern. Conference on Communication Systems and Network Technologies, pp. 801-806

Souza, L. B. L. de; Maia, M. de A. (2013) Do Software Categories Impact Coupling Metrics? In: Working Conference on Mining Software Repositories. pp. 217-220.

Tian, Y.; Chen, C.; Zhang, C. (2008) AODE for Source Code Metrics for Improved Software Maintainability. In: Int. Conf. on Semantics, Knowledge and Grid.pp.330-335.

Vaidya, J.; Clifton, C. (2003) Privacy-Preserving K-Means Clustering Over Vertically Partitioned Data. In: Int. Conf. on Knowledge Discovery and Data Mining. pp.206-215.

Yang, B.; Zheng, X.; Guo, P. (2006). Software Metrics Data Clustering for Quality Prediction. In: Computational Intelligence. pp. 959-964.

Zhang, B.; Hsu, M.; Dayal, U. (1999) K-Harmonic Means - A Data Clustering Algorithm. Hewlett-Packard Labs Technical Report HPL-1999-124

Zhong, S.; Khoshgoftaar, T. M.; Seliya, N. (2004) Analyzing Software Measurement Data with Clustering Techniques. In: Intelligence Systems. vol. 19, no. 2, pp. 20-27.