Samuel António Miquirice Domingos

STUDY ON THE RELATIONSHIPS BETWEEN COHESION AND COUPLING METRICS ON FAULT PREDICTION IN OBJECT ORIENTED SYSTEMS

Florianópolis
August 13, 2018

Samuel António Miquirice Domingos

STUDY ON THE RELATIONSHIPS BETWEEN COHESION AND COUPLING METRICS ON FAULT PREDICTION IN OBJECT ORIENTED SYSTEMS

Master's dissertation submitted to the Graduate Program in Computer Science at the Federal University of Santa Catarina as part of the requirements for obtaining the Master's degree in Computer Science. Advisor: Prof. Dr. Raul Sidnei Wazlawick

Florianópolis

2018

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Domingos, Samuel António Miquirice
Study on the relationships between cohesion and coupling metrics on fault prediction in object oriented systems / Samuel António Miquirice Domingos ; orientador, Raul Sidnei Wazlawick, 2018.
147 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós Graduação em Ciência da Computação, Florianópolis, 2018.

Inclui referências.

1. Ciência da Computação. 2. Métrica de Software. 3. Qualidade de Software. 4. Engenharia de Software. I. Wazlawick, Raul Sidnei. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

Samuel António Miquirice Domingos

STUDY ON THE RELATIONSHIPS BETWEEN COHESION AND COUPLING METRICS ON FAULT PREDICTION IN OBJECT ORIENTED SYSTEMS

This dissertation is recommended in partial fulfillment of the requirements for the degree of "Master in Computer Science", which has been approved in its present form by the Graduate Program in Computer Science.

Florianópolis, August 13th 2018.

Prof. Dr. José Luís Almada Güntzel
Graduate Program Coordinator
Federal University of Santa Catarina

Dissertation Committee:

Prof. Dr. Raul Sidnei Wazlawick
Advisor
Federal University of Santa Catarina

Prof. Dr. Marcelo Soares Pimenta
Federal University of Rio Grande do Sul
(Videoconferencing)

Prof. Dr. Cristiano Tolfo
Federal University of Pampa
(Videoconferencing)

Profa. Dra. Patrícia Vilain
Federal University of Santa Catarina

This work is dedicated to my God and my parents.

ACKNOWLEDGEMENTS

The author would like to thank Prof. Raul Sidnei Wazlawick for his advising, constant contributions and suggestions throughout the course, including the realization of the present work. The author also thanks the support of the Graduate Program in Computer Science (PPGCC-UFSC) for the trust deposited and in particular the Scholarship Institute (IBE) of Mozambique for the allocation of the international scholarship (from Mozambique to Brazil).

RESUMO ESTENDIDO

Contextualização: Existem diversas métricas propostas na literatura para medir a coesão e acoplamento em sistemas orientados a objetos e anualmente vão surgindo novas métricas. Algumas dessas métricas são úteis na predição de sistemas propensos a falhas. Porém, identificar tais métricas não é uma tarefa trivial. *Objetivo:* O presente trabalho faz o agrupamento das métricas de coesão e acoplamento existentes na literatura, com as respectivas ferramentas para automatizar o cálculo dos valores correspondentes a cada métrica. Este trabalho também faz uma análise cuidadosa de aplicação das métricas de coesão e acoplamento na predição de falhas, para permitir a identificação de métricas úteis. Espera-se que os resultados obtidos possam facilitar o planejamento e a elaboração de um novo modelo para predição de sistemas propensos a falhas. Finalmente, este trabalho procura encontrar a relação entre métricas de coesão e acoplamento, analisando estatisticamente os resultados das experiências. *Método:* Foram realizados dois mapeamentos sistemáticos da literaturae uma análise estatística. *Resultado:* Os resultados obtidos no primeiro mapeamento sistemático(95 estudos selecionados), apresentaram 95 métricas de coesão e 118 métricas de acoplamento, criando um agrupamento de 213 métricas de coesão e acoplamento. Dentre as 213 métricas agrupadas, 65 métricas de coesão e 81 métricas de acoplamento (totalizando 146) possuem ferramentas para automatizar o cálculo do valor correspondente a cada métrica. No total, foram encontrados 64 ferramentas, onde cada uma calcula mais de uma métrica de coesão ou acoplamento. Os resultados obtidos no segundo mapeamento sistemático(24 estudos selecionados), apontam para LCOM (métrica de coesão), CBO, e RFC (métricas de acoplamento) como sendo as mais úteis na predição de falhas, enquanto PROMISE Repository foi o sistema mais usado (36.4% dos estudos) na coleta de dados. A maioria dos projetos encontrados nos estudos de predição de falhas (94.6% de estudos) foram escrito em linguagem Java. CKJM é a ferramenta mais usada (37.5% dos estudos) para extrair as métricas de projetos em java. O estudo empírico mostrou que as métricas de coesão e acoplamento são inversamente correlacionadas. *Conclusão:* Algumas métricas de coesão e acoplamento (LCOM, CBO, and RFC) são úteis na predição de falhas. Porém, existem outras métricas que são úteis na predição de falhas (são pouco usadas) e não tem ferramentas para automatizar o cálculo do valor correspondente.

Palavras-chave: métrica de coesão, métrica de acoplamento, predição

de falha, propenso a falha, propensão a falha, sistemas orientados a objetos, ferramenta de engenharia de software.

ABSTRACT

Contextualization: There are several metrics proposed in the literature to measure cohesion and coupling in Object-Oriented Systems (OOS) and several new metrics continue to appear annually. Some of those metrics are useful for detecting fault-prone systems. However, assessing the utility of such metrics is not a trivial task. *Objective:* The present work makes the grouping of the cohesion and coupling metrics existing in the literature, with the respective tools to automate the calculation of the corresponding values. This work also makes a careful analysis of the application of cohesion and coupling metrics in fault prediction, to allow the identification of useful metrics. This result is expected to ease the planning and elaboration of a new prediction model for fault prone systems. Finally, this work attempts to find mutual relationships between cohesion and coupling metrics by statistically analyzing the results of experiments. *Method:* A systematic literature mapping (SLM) and a statistical analysis was conducted. *Result:* The results obtained in the first SLM (95 selected studies) presented 95 cohesion metrics and 118 coupling metrics, thus resulting in a set of 213 metrics of cohesion and coupling. Among the 213 metrics found, 65 cohesion metrics and 81 coupling metrics (totaling 146) have tools to automate the calculation of the corresponding value. In total, 64 tools were found, where each one calculates more than one metric of cohesion or coupling. The results obtained in the second SLM (24 selected studies) pointed to LCOM (cohesion metric), CBO, and RFC (coupling metrics) as the most useful in fault prediction, while the PROMISE Repository was the most used system in data collection(36.4% of studies). The project written in Java is the most used in studies of fault prediction (94.6% of studies). CKJM is the tool mostly used to extract metrics in Java projects(37.5% of studies). The empirical study shows that cohesion and coupling metrics are inversely correlated. *Conclusion:* Some cohesion and coupling metrics (LCOM, CBO, and RFC) are useful in fault prediction. However, there are other useful metrics in fault prediction that are less used and doesn't have tools to automate the calculation of the corresponding value.

Keywords: cohesion metric, coupling metric, fault prediction, fault prone, fault proneness, object-oriented systems, software engineering tool.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Object-oriented systems (OOS) are developed considering their internal qualities such as cohesion and coupling, as much as their external qualities as is the case of fault-proneness.

Cohesion and coupling are regarded as the fundamental characteristics of the internal quality of object-oriented systems. According to Etzkorn et al. (2004), cohesion is the degree to which the elements of a class or object belong together.

Cohesion of a class, according to Dallal (2015), can be defined as the degree of relationship between the members of the class(attributes and methods). The cohesion of a module can also be defined as the degree of relationship between the components of the module.

The coupling of a class, according to Dallal (2015), can be defined as the degree of its relationship to other classes, while the coupling of a module can be defined as the degree of its relationship to other modules (BRIAND; DALY; WUST, 1999).

Cohesion and coupling in software are measured by different metrics proposed in the literature.

From a literature review, several metrics to measure cohesion and coupling in OOS were found. However, it is difficult to quantify all existing metrics and the respective tools to automate their calculation. Without a carefully study, it is also difficult to know which metrics are useful in predicting faults in OOS.

The concepts of error, fault, and failure can be easily confused as if they were the same thing. However, in software engineering, there is a difference between these concepts.

According to Sanyal and Singh (2014), *fault* is a defect that can result in software failure, and *failure* refers to the misuse of software actions in the expected results, i.e. failure can be seen when a certain software produces unexpected results, while *error* is what occurs when the software requirements are not in compliance. The error is caused by a mistake of a human being, especially when specifying the requirements of the software.

Fault proneness, according to (BRIAND; WÜST; LOUNIS, 2001), can be defined as the likelihood of detecting a fault in a class. For this it can be used the classification technique called *logistic regression*, which is based on the likelihood of event prediction.

The present work makes a careful analysis of the application of the cohesion and coupling metrics in the prediction of faults, aiming to gather enough information to allow the identification of useful metrics in the prediction of faults.

1.1 DELIMITATION OF THE PROBLEM

From a literature review, there are several metrics proposed to measure cohesion and coupling in OOS and several new metrics continue to appear annually. Some of those metrics are useful for detecting fault-prone systems.

According to many authors (BASILI; BRIAND; MELO, 1996) (BRIAND; WÜST; LOUNIS, 2001) (EMAM; MELO; MACHADO, 2001) (YU; SYSTA; MULLER, 2002) (GYIMOTHY; FERENC; SIKET, 2005) (OLAGUE et al., 2007) (AGGARWAL et al., 2007) (ENGLISH et al., 2009) (ELISH; AL-YAFEI; AL-MULHEM, 2011) (RATHORE; GUPTA, 2012), cohesion and coupling metrics can be used to predict faults in classes and OOS.

However, it is difficult to know which metrics are useful in predicting faults in OOS. It is also difficult to know specifically where to find the existing metrics and the respective tools to automate their calculation. Without that knowledge, it could be difficult to avoid the emergence of new metrics proposals that do the same thing as some metrics already existing in the literature.

Studying the relations between internal quality attributes of software helps explaining some practical issues and results of the relation between internal (for instance, cohesion and coupling) and external quality attributes (for instance, fault prediction). This study may help software engineers and practitioners to figure out the quality factors that must be considered and focused on during the quality assessment process (DALLAL, 2015).

1.2 OBJECTIVES

1.2.1 General Objective

The general objective of this work is to study on the relationships between cohesion and coupling metrics on fault prediction in object oriented systems.

1.2.2 Specific Objectives

In order to achieve the general objective of this work, some specific objectives have been drawn, such as:

- To group the cohesion and coupling metrics existing in the literature, with the respective tools to automate the calculation of the corresponding values.

- Make a careful analysis of the application of cohesion and coupling metrics in fault prediction, to allow the identification of useful metrics.

- Empirically investigate the relationships among several cohesion and coupling metrics in object-oriented systems.

- Find mutual relationships between cohesion and coupling metrics in fault prediction by comparison of obtained results.

- To perform a detailed analysis on the effect of cohesion and coupling metrics on fault prediction in object-oriented systems.

Although the cohesion and coupling metrics in OOS can be analyzed in different aspects, the focus of this research is to carefully target the different specific objectives mentioned above.

1.3 RELEVANCE

The knowledge about all existing metrics and the respective tools to automate their calculation, could avoid the emergence of new metrics proposals that do the same thing as some metrics already existing in the literature.

According to many authors (BASILI; BRIAND; MELO, 1996) (BRIAND; WÜST; LOUNIS, 2001) (EMAM; MELO; MACHADO, 2001) (YU; SYSTA; MULLER, 2002) (GYIMOTHY; FERENC; SIKET, 2005) (OLAGUE et al., 2007) (AGGARWAL et al., 2007) (ENGLISH et al., 2009) (ELISH; AL-YAFEI; AL-MULHEM, 2011) (RATHORE; GUPTA, 2012), it can be seen that the cohesion and coupling metrics are not used only to verify how much a OOS is cohesive or coupled, but they can also be used to predict faults in OOS.

The present work makes a careful analysis of the effect of cohesion and coupling metrics on fault prediction, aiming to gather enough information to allow the identification of useful metrics in fault prediction.

This study may help software engineers and practitioners to figure out the quality factors that must be considered and focused on during the quality assessment process (DALLAL, 2015).

The results produced in the present research are relevant because they could ease the planning and elaboration of new models for predicting fault-prone systems.

1.4 ORGANIZATION OF WORK

After the introduction (Chapter 1), the work is organized into eight other sections. Chapter 2 presents the literature review. Chapter 3 is dedicated to grouping cohesion and coupling metrics, and their respective calculation tools. Chapter 4 is dedicated exclusively to the application of cohesion and coupling metrics in fault prediction. Chapter 5 presents the relationship between cohesion and coupling metrics for OOS.

The analysis and interpretation of results obtained are presented in Chapter 6. The comparison of obtained results with related works is done in Chapter 7. In Chapter 8 the limitations are presented, and Chapter 9 presents the conclusions and future work. Finally, the bibliographic references and appendix are presented.

2 LITERATURE REVIEW

This chapter presents several studies found in the literature during the first and second phases of the SLM presented in Chapters 3 and 4, with a focus on cohesion and coupling metrics. The descriptions of the metrics can be found in tables 17 and 18 (in appendix).

According to Etzkorn et al. (2004), cohesion is the degree to which the elements of a class or object belong together.

Cohesion of a class, according to Dallal (2015), can be defined as the degree of relationship between the members of the class(attributes and methods). The cohesion of a module can also be defined as the degree of relationship between the components of the module.

The coupling of a class, according to Dallal (2015), can be defined as the degree of its relationship to other classes, while the coupling of a module can be defined as the degree of its relationship to other modules (BRIAND; DALY; WUST, 1999).

In the research of Etzkorn et al. (2004), 10 cohesion metrics and their respective tools (3 tools) for C++ language were identified.

In another research (RAGAB; AMMAR, 2010), four cohesion metrics and 10 coupling metrics were surveyed, including their respective tools (4 tools) for Java and C++ languages. Kayarvizhy (2016), selected 9 tools and 6 metrics of cohesion and coupling.

The work with more metrics analyzed until now was (BRIAND; WÜST; LOUNIS, 2001) with 10 cohesion metrics and 28 coupling metrics, followed by (GEETIKA; SINGH, 2014) with 34 coupling metrics only, and (AGGARWAL et al., 2009) with 7 cohesion metrics and 21 coupling metrics.

In (NUÑEZ-VARELA et al., 2017), an systematic review of the literature (SRL) with data from 2010 to 2015, among 11 tools found, *Understand* occupied the first place and *CKJM* the second.

According to Radjenović et al. (2013), from 1991 to 2011, the most commonly used languages were: C++ (35%), followed by Java (34%) and C (16%). According to Isong and Ekabua (2015), from January 1995 to December 2012, about 54% of applications were developed in C++, and only 43% in Java. The results obtained in (SANTOS et al., 2016), indicate that from 2004 to 2013, Java was the most used language (55%) followed by C++ (21%). According to Nuñez-Varela et al. (2017), between 2010 and 2015 the Java language was the most used (64.7%), followed by AspectJ (12.6%), C++ (10.1%), C (5%), C#(2.5%), Jak (1.7%), Ada (0.8%), Cobol

(0.4%), Javascript(0.4%), Pharo (0.4%), PHP (0.4%), Python (0.4%), and Ruby (0.4%).

Nuñez-Varela et al. (2017) pointed to the *PROMISE repository of University of Ottawa* as the most used repository, and the *PROMISE* repository in second place. It diverges from Malhotra (2015) and Isong and Ekabua (2015) that point to *NASA* in the first place and *PROMISE* in second.

Table 1, presents the different related works found during the literature review.

Table 1: Literature Review

Authors	**Cohesion metrics**	**Coupling metrics**	**The useful Cohesion metrics for fault prediction**	**The useful Coupling metrics for fault prediction**	**Relationships: cohesion and coupling metrics**
Basili, Briand and Melo (1996)	LCOM	CBO and RFC	—	CBO and RFC	—
Briand, Wüst and Lounis (2001)	LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, Coh, Co, LCC, TCC, and ICH	CBO, CBO', RFC1, RFC2, ICP, IH-ICP, NIH-ICP, MPC, DAC, DAC', IFCAIC, ACAIC, OCAIC, FCAEC, DCAEC, OCAEC, IFCMIC, ACMIC, OCMIC, FCMEC, DCMEC, OCMEC, IFMMIC, AMMIC, OMMIC, FMMEC, DMMEC, and OMMEC	—	—	—
El Emam, Melo and Machado (2001)	—	ACAIC, ACMIC, DCAEC, DCMEC, OCAIC, OCAEC, OCMIC, and OCMEC	—	OCMEC	—
Kabaili, Keller and Lustman (2001)	LCC and LCOM	CBO, RFC, CBO_NA, CBO_IUB, and CBO_U	—	—	There is no correlation
Yu, Systa and Muller (2002)	LCOM	CBOin, CBOout, RFCin, RFCout, and Fan-in	—	CBOout and RFCout	—
Gyimothy, Ferenc and Siket (2005)	LCOM	CBO and RFC	—	CBO	—
Olague et al. (2007)	LCOM and CAM	CBO, RFC and DCC	LCOM	CBO and RFC	—
Aggarwal et al. (2007)	LCOM1 and LCOM2	CBO, CBO1, RFC, IFCAIC, ACAIC, OCAIC, FCAEC, DCAEC, OCAEC, IFCMIC, ACMIC, DCMIC, FCMEC, DCMEC, OCMEC, IFMMIC, AMMIC, OMMIC, FMMEC, DMMEC, and OMMEC	LCOM1 and LCOM2	CBO and CBO1	There are interrelationships between selected metrics

Badri, Badri and Gueye (2008)	DCDE, DCIE, DCD, and DCI	CBO	—	—	There is negative correlation
English et al. (2009)	—	CBO and RFC	—	CBO and RFC	—
Elish, Al-Yafei and Al-Mulhem (2011)	LCOM	CBO, RFC, Ca, Ce, and CF	—	Ca, Ce and CF	—
Rathore and Gupta (2012)	LCOM, LCOM3 and CAM	CBO, RFC, CA, CE and DAM	—	CBO, RFC, CA and CE	Moderate or correlated
Dallal (2013)	LCOM1, LCOM2, LCOM3, LCOM4, and LCOM5	MPC, RFC, DAC, DAC2, OCMEC, CBO_U, CBO_IUB, and CBO	—	—	There is negative correlation
Dallal (2015)	CBMC, ICBMC, OL, and PCCC	MPC, RFC, DAC, DAC2, OCMEC, CBO_U, CBO_IUB, and CBO	—	—	There is negative correlation
The present work (2018)	95 cohesion metrics	118 coupling metrics	LCOM, APIU, CAM, CU, DCO, Fcoh, HC, ILCO, IPSC, NHD, Overlap, PF, SBFC, SCC, and Tightness	CBO, RFC, Ce, MII, MPC, and NC	there is in fact a negative correlation

In the software engineering community, some researchers intuitively believe that high cohesion is related to low coupling and vice versa. However, the relationship between the cohesion and coupling metrics was not sufficiently studied to support this intuition. Research carried out in the analysis of the relationship between cohesion and coupling metrics did not cover a representative number of metrics to guarantee the generalization of results.

Researchers have proposed several metrics to find cohesion and coupling in object-oriented systems. However, few of them have proposed an analysis of the relationship between cohesion and coupling. Mateos et al. (2016) and Anabalon et al. (2017) reported the research they performed to analyze the correlation between cohesion and coupling metrics (CBO, RFC, and LCOM) with complexity metrics. On the other hand, Kumar, Misra and Rath (2017) analyzed the correlation between cohesion and coupling metrics (CBM, IC, CAM, LCOM3, CE, CA, LCOM, RFC, and CBO) with fault proneness.

In (DALLAL, 2015), (DALLAL, 2013), and (BADRI; BADRI; GUEYE, 2008), the authors analyzed the correlation between cohesion and coupling metrics, but conclusions were hard to generalize because a small number of metrics were used.

Badri, Badri and Gueye (2008) in their research use four cohesion metrics and only one coupling metric. Dallal (2013) in her

research used five cohesion metrics and eight coupling metrics, two years later, in a new research, Dallal (2015) used four cohesion metrics and eight coupling metrics.

The focus of the present work is on the analysis of the useful cohesion and coupling metrics in fault prediction, including the correlation between cohesion and coupling metrics.

3 GROUPING COHESION AND COUPLING METRICS

This work presents two systematic literature mappings (SLM). The first SLM is dedicated to grouping cohesion and coupling metrics and their respective calculation tools. The application of cohesion and coupling metrics in fault prediction was carefully analyzed in the second SLM (Chapter 4).

By the bibliographical revision, it can be said that there are several metrics to measure cohesion and coupling in OOS. However, it is difficult to know specifically where to find the all existing metrics and the respective tools to automate their calculation. That knowledge could avoid the emergence of new metrics proposals that do the same thing as some metric already existing in the literature.

The present chapter groups the cohesion and coupling metrics existing in the literature with their respective calculation tools. A systematic literature mapping (SLM) was carried out in which 95 relevant studies on cohesion and coupling metrics and their respective calculation tools were carefully analyzed.

3.1 METHODOLOGICAL PROCEDURES (1ST SLM)

In the present chapter, to answer the proposed research questions, a SLM was carried out following pre-defined procedures (Figure 1) that comprise three phases (adapted from the procedures presented by KITCHENHAM, 2004): planning, driving the SLM, and presenting the SLM results.

Figure 1 presents the phases of the SLM in order. The subsections that follow present more details about these phases.

3.1.1 Planning the Review

The planning phase went through two stages: the identification of the SLM need and the development of the SLM protocol.

3.1.1.1 Identification of the SLM need

Object-oriented systems are developed considering their internal qualities, such as cohesion and coupling.

Cohesion and coupling in a OOS are measured by different metrics proposed in the literature. Some of these metrics have calculation tools available. Unfortunately, finding them is not a trivial task. With the considerable number of cohesion and coupling met-

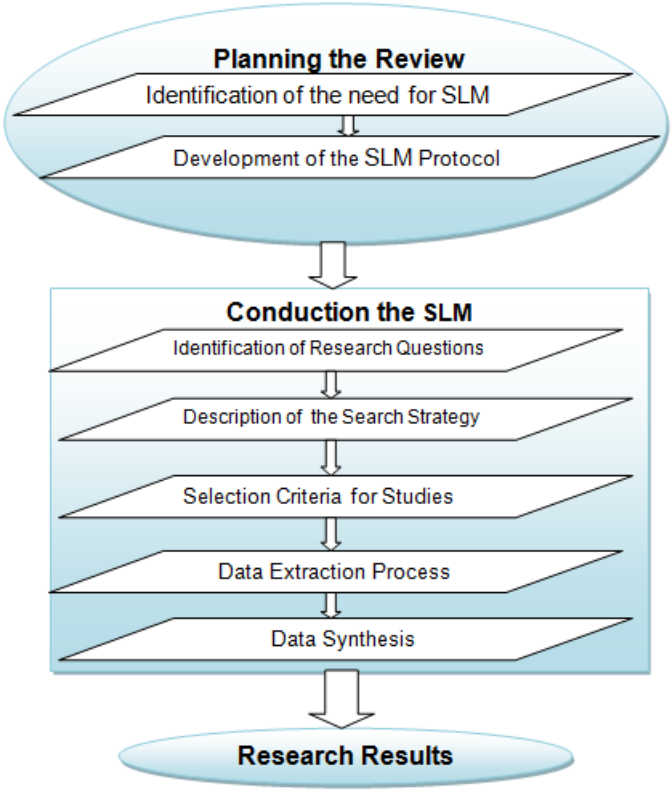

Figure 1: Process of systematic mapping

rics currently in the literature, it is important to carry out a study of the state of the art of the metrics gathering information for future research.

A search in six databases was carried out to find similar secondary studies on the subject. Studies have not been found with scope and period similar to those of the revision proposed in this first SLM.

3.1.1.2 Development of the SLM protocol

In the first step, the research questions to be answered in the SLM were formulated. In the second step, in addition to being given the description of the search strategy, the identification of the

search terms and the selection of sources to be searched for the identification of primary studies was also made.

The third step was marked by the determination of relevant studies based on research questions, including the determination of the inclusion and exclusion criteria for each primary study.

The fourth step was dedicated to the description of the data extraction process to collect the necessary information to respond to research questions, and in the last step the methods for data synthesis were elaborated.

The development of the SLM protocol has important steps (according to KITCHENHAM, 2004) that eased the development of the SLM, thus reducing the possibility of leaving the present work at the risk of achieving unwanted quality.

3.1.2 Conducting the SLM

The first SLM of this work was conducted with the following steps: a) identification of research questions; b) description of the search strategy; c) study selection criteria; d) evaluation of the quality of the study; e) data extraction process; f) data synthesis. The details of each step are presented in the form of subsections obeying the same sequence.

3.1.2.1 Identification of research questions

The research questions defined in this study are: **RQ1:** Are there tools available to automate the calculation of the values corresponding to all cohesion and coupling metrics in OOS? **RQ2:** What tools are most used to automate the calculation of the values corresponding to the cohesion and coupling metrics in OOS? **RQ3:** How many metrics of cohesion and coupling in OOS are there in the literature? **RQ4:** What is the most used programming language for metrics data collection? **RQ5:** Specifically, which metrics of cohesion or coupling are the most studied?

3.1.2.2 Description of the search strategy

For conducting the search, its scope was initially defined in terms of time, electronic databases, and search strategy. The definition of the scope helped in the reflection of the SLM objectives and the importance of carrying out a qualitative selection of the articles available in the databases. The terms of the scope are described below.

Time interval: The search included articles published in any period to meet the objectives of the research regarding obtaining all the metrics of cohesion and coupling for OOS.

Electronic databases: Six electronic databases were selected as primary sources for this research: ACM Digital Library, ProQuest, ScienceDirect, Scopus, Spring Link and World Scientific.

Scopus is a database that has connections to other databases (including IEEE Xplore Digital Library). The criteria for the choice of the six databases were based on their popularity in the literature (for the area of computing) and for the free access to the articles available in these databases that is provided by the Portal of Periodicals CAPES/MEC.

Search strategy: Automated searches were carried out in the electronic databases with the goal of finding articles relevant to the present work. In addition to the automatic searches, the manual searches were carried out during the data collection process, because it was perceived the need to include some quoted articles (in the selected articles) as good sources (article with a proposal of some metric) for obtaining the detailed description of some cohesion and coupling metrics, regardless of the tool indication to automate the calculation of the metric value. The searches were performed based on a string formed by keywords and logical operators "and" and "or", such as: "software metric", "cohesion metric", "coupling metric", "object oriented metric", "open source oo tool", "open source object oriented tool", "tool", "measuring", and "measure". The search string used in the first SLM was: (“software metric” OR “cohesion metric” OR “coupling metric” OR “object oriented metric”) AND (“open source oo tool” OR “open source object oriented tool” OR tool) AND (measuring OR measure).

3.1.2.3 Selection criteria for studies

After the search carried out in the six databases, followed the pre-process and finally the selection of the articles by the inclusion criteria (IC) and exclusion criteria (EC) presented below.

IC1: Publications in which the focus is directly or indirectly related to the cohesion or coupling metrics and that specify the tools used to automate the calculation of each metric.

IC2: Publications of journals, conferences, or events in computer science that present results, evaluations, or validations.

IC3: When finding articles with the same authors and similar researches, but at different events, only the most recent was

included.

IC4: Articles that propose and validate metrics of cohesion or coupling for object-oriented systems (OOS), regardless of the tool indication used to automate the calculation of each metric.

EC1: Exclude publications related to cohesion or coupling metrics that do not specify the tools used to automate the calculation of each metrics except for publications with new metrics being proposed.

EC2: Publications of thesis papers, dissertations, books, reports and other works that are not classified as scientific articles (gray literature).

EC3: Duplicate or similar articles.

Based on the criteria for inclusion and exclusion of the primary articles, it was possible to carry out the selection of the articles found in the search without losing the focus of this research. The selection of articles went through four steps, obtaining the results presented below.

In the first step, the search was performed using the search string in the mechanisms quoted. 237 ACM Digital Library, 158 papers from ProQuest, 1114 papers from ScienceDirect, 216 papers from Scopus, 195 papers from Spring Link and 8 papers from World Scientific, totaling 1928 articles, have been identified.

In the second step, the titles, abstracts, and keywords of each article were read. After reading, they were pre-selected (separately in each database) resulting in a total of 152 articles (corresponding to 7.9% of 1928, i.e. the total of articles obtained in the first step) in the six databases, where 30 articles (12.7% of 237) are from the ACM Digital Library, 19 (12% of 158) from ProQuest, 45 (4% of 1114) from ScienceDirect, 16 (7.4% of 216) from Scopus, 40 (20.5% of 195) from Spring Link, and 2 (25% of 8) from World Scientific.

In the third step, the sections of introduction, data collection, results, and conclusion of the 152 pre-selected articles were read. More than 90% of the articles that were removed (in the third step),were by the exclusion criterion 1 (EC1), because these articles do not propose new metrics and do not specify the tools used for calculation. During the third step 95 articles were selected (corresponding to 4.9% of the total result obtained in the first step), among which 13 (5.5%) from the ACM Digital Library, 12 (7.6%) from ProQuest, 27 (2.4%) from ScienceDirect, 10 (4.6%) from Scopus, 33 (16.9%) from Spring Link, and none (0%) from World Scientific.

The 95 selected articles (A1 through A95) presented in Table 2 are: **A1:** (MOSER et al., 2008), **A2:** (ETZKORN et al., 2004), **A3:** (TEMPERO; RALPH, 2018), **A4:** (MA et al., 2010), **A5:** (TOURE; BADRI; LAMONTAGNE, 2014), **A6:** (OKIKE, 2010), **A7:** (HIGO et al., 2011), **A8:** (DALLAL; BRIAND, 2012), **A9:** (BADRI; BADRI, 2004), **A10:** (SCOTTO et al., 2004), **A11:** (MITCHELL; POWER, 2006), **A12:** (AHMED; ABUBAKAR; ALGHAMDI, 2011), **A13:** (KUMAR; MISRA; RATH, 2017), **A14:** (YANG et al., 2016), **A15:** (MUBARAK; COUNSELL; HIERONS, 2010), **A16:** (MALHOTRA; KHANNA, 2017), **A17:** (ZHOU et al., 2012), **A18:** (DALLAL; BRIAND, 2010), **A19:** (SYSTA; YU; MULLER, 2000), **A20:** (CHAHAL; SINGH, 2011).

A21: (CONTRERAS et al., 2010), **A22:** (CHOWDHURY; ZULKERNINE, 2010), **A23:** (MÄKELÄ; LEPPÄNEN, 2009), **A24:** (LINCKE; LUNDBERG; LÖWE, 2008), **A25:** (BUDHKAR; GOPAL, 2012), **A26:** (ENGLISH; CAHILL; BUCKLEY, 2012), **A27:** (BOSHNAKOSKA; MIŠEV, 2010), **A28:** (PADHY; SINGH; SATAPATHY, 2018), **A29:** (ARAR; AYAN, 2016), **A30:** (SINGH; SINGH, 2008), **A31:** (GUPTA; CHHABRA, 2011), **A32:** (GOEL; GUPTA, 2017), **A33:** (HASSOUN; COUNSELL; JOHNSON, 2005), **A34:** (MALHOTRA; KAUR; SINGH, 2010), **A35:** (PADHY; SINGH; SATAPATHY, 2017), **A36:** (OOSTERMAN; IRWIN; CHURCHER, 2011), **A37:** (KHARI; KUMAR, 2017), **A38:** (QU et al., 2015), **A39:** (BADRI; BADRI; TOURE, 2010), **A40:** (DALLAL, 2012a).

A41: (MITTAL; SINGH; KAHLON, 2011), **A42:** (IBRAHIM et al., 2012), **A43:** (DALLAL, 2011a), **A44:** (SURI; SINGHAL, 2015), **A45:** (KANMANI et al., 2004a), **A46:** (MALHOTRA; KHANNA, 2013), **A47:** (MALHOTRA; BANSAL, 2018), **A48:** (XIE et al., 2000), **A49:** (KAKARONTZAS et al., 2013), **A50:** (PAN et al., 2010), **A51:** (DALLAL, 2011b), **A52:** (KANMANI et al., 2004b), **A53:** (ROCHA et al., 2013), **A54:** (SINGH; KAHLON, 2014), **A55:** (LI; HENRY, 1993b), **A56:** (KANMANI et al., 2007), **A57:** (GUPTA; CHHABRA, 2012), **A58:** (TOURE; BADRI; LAMONTAGNE, 2018), **A59:** (DALLAL, 2017), **A60:** (DALLAL; MORASCA, 2014).

A61: (COSCIA et al., 2012), **A62:** (VIJI; RAJKUMAR; DURAISAMY, 2018), **A63:** (SINGH; KAUR; MALHOTRA, 2009), **A64:** (OFFUTT; ABDURAZIK; SCHACH, 2008), **A65:** (MISRA; BHAVSAR, 2003), **A66:** (BRIAND; WÜST; LOUNIS, 2001), **A67:** (WOO et al., 2009), **A68:** (BADRI; BADRI; GUEYE, 2008), **A69:** (EDELMAN; FRAKES; LILLIE, 2008), **A70:** (WANJIKU; OKEYO; CHERUIYOT, 2016), **A71:** (MEYERS; BINKLEY, 2004), **A72:** (ALSHAMMARY; ALENEZI, 2017), **A73:** (GUPTA; SAXENA, 2017), **A74:** (CATAL; DIRI, 2007), **A75:** (REDDY; KHURANA; OJHA, 2015), **A76:** (ARVANITOU et al., 2016), **A77:** (BOUCHER; BADRI, 2017), **A78:** (ILYAS et al., 2017), **A79:**

(SANTOS; AFONSO; COSTA, 2016), **A80:** (VEADO et al., 2016).

A81: (LU et al., 2012), **A82:** (ABUASAD; ALSMADI, 2012), **A83:** (SHATNAWI; LI, 2008), **A84:** (DALLAL, 2012b), **A85:** (KUMAR; KRISHNA; RATH, 2017), **A86:** (EMAM; MELO; MACHADO, 2001), **A87:** (DALLAL, 2011c), **A88:** (BATISTA; SANT'ANNA; SILVA, 2017), **A89:** (GIRGIS; MAHMOUD; NOUR, 2009), **A90:** (CHOWDHURY; ZULKERNINE, 2011), **A91:** (ALMUGRIN; ALBATTAH; MELTON, 2016), **A92:** (POSHYVANYK et al., 2009), **A93:** (MARCUS; POSHYVANYK; FERENC, 2008), **A94:** (GUPTA, 2011), and **A95:** (JOHARI; KAUR, 2012).

Table 2: Selected articles

#	Article title	Database
A1	A Case Study on the Impact of Refactoring on Quality and Productivity in an Agile Team	Spring Link
A2	A comparison of cohesion metrics for object-oriented systems	ScienceDirect
A3	A framework for defining coupling metrics	ScienceDirect
A4	A Hybrid Set of Complexity Metrics for Large-Scale Object-Oriented Software Systems	Spring Link
A5	A metrics suite for JUnit test code: a multiple case study on open source software	Spring Link
A6	A Pedagogical Evaluation and Discussion about the Lack of Cohesion in Method (LCOM) Metric Using Field Experiment	ProQuest
A7	A Pluggable Tool for Measuring Software Metrics from Source Code	Scopus
A8	A Precise Method-Method Interaction-Based Cohesion Metric for Object-Oriented Classes	ACM Digital Library
A9	A proposal of a new class cohesion criterion: An empirical study	Scopus
A10	A relational approach to software metrics	ACM Digital Library
A11	A study of the influence of coverage on the relationship between static and dynamic coupling metrics	ScienceDirect
A12	A study on the uncertainty inherent in class cohesion measurements	ScienceDirect
A13	An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes	ScienceDirect
A14	An empirical investigation into the effect of slice types on slice-based cohesion metrics	ScienceDirect
A15	An Empirical Study of "Removed" Classes in Java Open-Source Systems	Spring Link
A16	An exploratory study for software change prediction in object-oriented systems using hybridized techniques	Spring Link
A17	An in-depth investigation into the relationships between structural metrics and unit testability in object-oriented systems	Spring Link
A18	An object-oriented high-level design-based class cohesion metric	ScienceDirect
A19	Analyzing Java Software by CombiningMetrics and Program Visualization	Scopus
A20	Analyzing Software Evolution Using the MOOD	ProQuest
A21	Balancing flexibility and performance in three dimensional meshing tools	ScienceDirect
A22	Can complexity, coupling, and cohesion metrics be used as early indicators of vulnerabilities?	ACM Digital Library
A23	Client-based cohesion metrics for Java programs	ScienceDirect
A24	Comparing software metrics tools	ACM Digital Library
A25	Component evaluation and component interface identification from Object Oriented System	ProQuest
A26	Construct specific coupling measurement for C++ software	ScienceDirect
A27	Correlation between Object-Oriented Metrics and Refactoring	Spring Link
A28	Cost-effective and fault-resilient reusability prediction model by using adaptive genetic algorithm based neural network for web-of-service applications	Spring Link
A29	Deriving thresholds of software metrics to predict faults on open source software: Replicated case studies	ScienceDirect
A30	DynaMetrics: a runtime metric-based analysis tool for object-oriented software systems	ACM Digital Library
A31	Dynamic cohesion measures for object-oriented software	ScienceDirect
A32	Dynamic Coupling Based Performance Analysis of Object Oriented Systems	ProQuest

A33	Dynamic coupling metric: Proof of concept	Scopus
A34	Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines	Spring Link
A35	Enhanced evolutionary computing based artificial intelligence model for web-solutions software reusability estimation	Spring Link
A36	EvoJava: a tool for measuring evolving software	ACM Digital Library
A37	Evolutionary Computation-Based Techniques Over Multiple Data Sets: An Empirical Assessment	Spring Link
A38	Exploring community structure of software Call Graph and its applications in class cohesion measurement	ScienceDirect
A39	Exploring Empirically the Relationship between Lack of Cohesion and Testability in Object-Oriented Systems	Spring Link
A40	Fault prediction and the discriminative powers of connectivity-based object-oriented class cohesion metrics	ScienceDirect
A41	Identification of Error Prone Classes for Fault Prediction Using Object Oriented Metrics	Spring Link
A42	Identification of Nominated Classes for Software Refactoring Using Object-Oriented Cohesion Metrics	ProQuest
A43	Improving the applicability of object-oriented class cohesion metrics	ScienceDirect
A44	Investigating the OO characteristics of software using CKJM metrics	Scopus
A45	Investigation into the exploitation of Object-Oriented features	ACM Digital Library
A46	Investigation of relationship between object-oriented metrics and change proneness	Spring Link
A47	Investigation of various data analysis techniques to identify change prone parts of an open source software	Spring Link
A48	JBOOMT: Jade Bird Object-Oriented Metrics Tool	Scopus
A49	Layer assessment of object-oriented software: A metric facilitating white-box reuse	ScienceDirect
A50	Measuring Structural Quality of Object-Oriented Softwares via Bug Propagation Analysis on Weighted Software Networks	Spring Link
A51	Measuring the Discriminative Power of Object-Oriented Class Cohesion Metrics	ProQuest
A52	Measuring the Object-Oriented Properties in Small Sized C++ Programs – An Empirical Investigation	Spring Link
A53	Mining the impact of evolution categories on object-oriented metrics	Spring Link
A54	Object oriented software metrics threshold values at quantitative acceptable risk level	Spring Link
A55	Object-oriented metrics that predict maintainability	ScienceDirect
A56	Object-oriented software fault prediction using neural networks	ScienceDirect
A57	Package level cohesion measurement in object-oriented software	ProQuest
A58	Predicting different levels of the unit testing effort of classes using source code metrics: a multiple case study on open-source software	Spring Link
A59	Predicting Fault-Proneness of Reused Object-Oriented Classes in Software Post-Releases	Spring Link
A60	Predicting object-oriented class reuse-proneness using internal quality attributes	Spring Link
A61	Predicting Web Service Maintainability via Object-Oriented Metrics: A Statistics-Based Approach	Spring Link
A62	Prediction of software fault-prone classes using an unsupervised hybrid SOM algorithm	Spring Link
A63	Prediction of Software Quality Model Using Gene Expression Programming	Spring Link
A64	Quantitatively measuring object-oriented couplings	ProQuest
A65	Relationships Between Selected Software Measures and Latent Bug-Density: Guidelines for Improving Quality	Spring Link
A66	Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs	Spring Link
A67	Revising cohesion measures by considering the impact of write interactions between class members	ScienceDirect
A68	Revisiting class cohesion: An empirical investigation on several systems	Scopus
A69	SAM: Simple API for Object-Oriented Code Metrics	Spring Link
A70	Scoped Class Cohesion Metric for Software Process Assessment	ProQuest
A71	Slice-based cohesion metrics and software intervention	Scopus
A72	Software Architecture Understandability in Object-Oriented Systems	ProQuest
A73	Software bug prediction using object-oriented metrics	Spring Link
A74	Software Fault Prediction with Object-Oriented Metrics Based Artificial Immune Recognition System	Spring Link
A75	Software Maintainability Estimation Made Easy: A Comprehensive Tool COIN	ACM Digital Library
A76	Software metrics fluctuation: a property for assisting the metric selection process	ScienceDirect

A77	Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison	ScienceDirect
A78	SourceViz: A Tool for Supporting Software Metrics Visualization	ProQuest
A79	SRT — A computational tool for restructuring Java software	Scopus
A80	TDTool: threshold derivation tool	ACM Digital Library
A81	The ability of object-oriented metrics to predict change-proneness: a meta-analysis	Spring Link
A82	The correlation between source code analysis change recommendations and software metrics	ACM Digital Library
A83	The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process	ScienceDirect
A84	The impact of accounting for special methods in the measurement of object-oriented class cohesion on refactoring and fault prediction activities	ScienceDirect
A85	The impact of feature selection on maintainability prediction of service-oriented applications	Spring Link
A86	The prediction of faulty classes using object-oriented design metrics	ScienceDirect
A87	Transitive-based object-oriented lack-of-cohesion metric	ScienceDirect
A88	Two quasi-experiments on cohesion metrics and program comprehension	ACM Digital Library
A89	UML class diagram metrics tool	Scopus
A90	Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities	ScienceDirect
A91	Using indirect coupling metrics to predict package maintainability and testability	ScienceDirect
A92	Using information retrieval based coupling measures for impact analysis	Spring Link
A93	Using the Conceptual Cohesion of Classes for Fault Prediction in Object-Oriented Systems	ProQuest
A94	Validation of dynamic coupling metrics for object-oriented software	ACM Digital Library
A95	Validation of object oriented metrics using open source software system: an empirical study	ACM Digital Library

The 57 excluded articles (A96 up to A152) are: **A96:** (MANEVA; GROZEV; LILOV, 2010), **A97:** (NÚÑEZ-VARELA et al., 2013), **A98:** (TEMPERO; RALPH, 2016), **A99:** (RATHORE; GUPTA, 2011), **A100:** (OFFUTT; HARROLD; KOLTE, 1993), **A101:** (CHHABRA; GUPTA, 2010), **A102:** (LAMMA; MELLO; RIGUZZI, 2004), **A103:** (ISONG; OBETEN, 2013), **A104:** (KHAN; ELISH; EL-ATTAR, 2012), **A105:** (SUBRAMANIAN; CORBIN, 2001), **A106:** (MEYERS; BINKLEY, 2007).

A107: (CHHILLAR; GAHLOT, 2017), **A108:** (ALEXAN; GAREM; OTHMAN, 2016), **A109:** (SANTHOSHINI; ANBAZHAGAN, 2014), **A110:** (GOEL; BHATIA, 2013), **A111:** (TSUI; IRIELE, 2011), **A112:** (KUMARI; BHASIN, 2011), **A113:** (PRITCHETT; WILLIAM, 1996), **A114:** (ANJOS; GOMES; ZENHA-RELA, 2012), **A115:** (CONCAS et al., 2010), **A116:** (PEREPLETCHIKOV; RYAN; FRAMPTON, 2005), **A117:** (YADAV; YADAV, 2015), **A118:** (SIRBI; KULKARNI, 2010), **A119:** (JAVED; ALENEZI, 2016).

A120: (GEETIKA; SINGH, 2014), **A121:** (POWAR et al., 2017), **A122:** (JOHARI; KAUR, 2011), **A123:** (VANDERFEESTEN; REIJERS; AALST, 2008), **A124:** (KUMAR; RATH, 2016), **A125:** (BAVOTA; LUCIA; OLIVETO, 2011), **A126:** (FERREIRA et al., 2012), **A127:** (RATHEE; CHHABRA, 2018), **A128:** (HALL; TAO; MUNSON, 2005), **A129:** (BRÄUER et al., 2018), **A130:** (ALLEN; GOTTIPATI; GOVINDARAJAN, 2007), **A131:** (DURISIC et al., 2013), **A132:** (SINGH, 2013), **A133:** (YU; CHEN; RAMASWAMY, 2009).

A134: (DESOUKY; ETZKORN, 2014), **A135:** (BELLIN; TYAGI; TYLER, 1994), **A136:** (MCQUILLAN; POWER, 2006), **A137:** (TOMAS; ESCALONA; MEJIAS, 2013), **A138:** (RAZ; YAUNG, 1993), **A139:** (BOWES; HALL; KERR, 2011), **A140:** (NUÑEZ-VARELA et al., 2017), **A141:** (KAUR; KAUR, 2013), **A142:** (CONCAS et al., 2008), **A143:** (EMAM et al., 2001), **A144:** (COUNSELL; SWIFT; CRAMPTON, 2006), **A145:** (DALLAL, 2012c), **A146:** (KEONG et al., 2015), **A147:** (MATHUR; KEEN; ETZKORN, 2011), **A148:** (AMARA; RABAI, 2017), **A149:** (SATO; GOLDMAN; KON, 2007), **A150:** (MITCHELL; POWER, 2005), **A151:** (PEDRYCZ et al., 2001) and **A152:** (BAKAR; BOUGHTON, 2012).

After the third step, all 95 selected articles were read in full. In the next section, the SLM presentation continue to show the extraction of relevant data.

3.1.2.4 Data extraction process

The extraction of data included the filling of a form for each of the selected primary articles to decide which research question was answered.

The form holds the author's name, title, publication location, year of publication, cohesion or coupling metrics used, tool used, and programming language used to extract the data.

The details of the specific research questions answered by each primary study were saved in a spreadsheet for later use during the data synthesis process that culminates in obtaining the results.

3.1.2.5 Data synthesis

The data synthesis was carried out with the goal of collecting and combining facts given in the selected primary articles to formulate answers to the research questions, according to the definition of the synthesis (WEN et al., 2012).

The selection of several articles presenting metrics, tools and comparable results, facilitated the obtaining of research evidence that allowed the reach of conclusive answers to the research questions.

The articles were analyzed and evaluated about quantitative data including the quantity of metrics and used systems, as well as about qualitative data that includes the criteria applied in the selection of the articles.

During the synthesis, to answer the research questions of this work, tables were used to summarize the presentation of the results.

3.2 RESEARCH RESULTS (1ST SLM)

The results obtained in the selected primary articles are presented in this section. The presentation structure starts with an overview of the publishing sources of these articles.

After the overview of each primary article, the answers to the questions of the research follow.

3.2.1 Publishing Source

Table 3 summarizes the details of publications in main conferences and periodicals, corresponding to the selected articles.

Table 3: Details of Publications of Selected Articles

#	Article	Source of Publication	Total	(%)
1	A1	Balancing Agility and Formalism in Software Engineering	1	1.786
2	A2, A14, A18, A40, A43, A56, A67, A76, A77	Information and Software Technology	9	16.071
3	A3, A11, A23	Science of Computer Programming	3	5.357
4	A4, A50	Journal of Computer Science and Technology	2	3.571
5	A5	Journal of Software Engineering Research and Development	1	1.786
6	A6, A42, A70	International Journal of Computer Science Issues (IJCSI)	3	5.357
7	A7	International Workshop on Software Measurement and International Conference on Software Process and Product Measurement	1	1.786
8	A8	ACM Transactions on Software Engineering and Methodology (TOSEM)	1	1.786
9	A9, A68	Journal of Object Technology	2	3.571
10	A10, A22	ACM symposium on Applied computing	2	3.571
11	A12, A31, A90	Journal of Systems Architecture	3	5.357
12	A13	Computer Standards & Interfaces	1	1.786
13	A15	Advanced Techniques in Computing Sciences and Software Engineering	1	1.786
14	A16	Automated Software Engineering	1	1.786
15	A17	Science china information sciences	1	1.786
16	A19	Software Maintenance and Reengineering	1	1.786
17	A20, A25, A32	International Journal of Advanced Research in Computer Science	3	5.357
18	A21	Advances in Engineering Software	1	1.786
19	A24	International symposium on Software testing and analysis	1	1.786
20	A26	Computer Languages, Systems & Structures	1	1.786
21	A27	International Conference on ICT Innovations	1	1.786
22	A28, A35, A62	Cluster Computing	3	5.357
23	A29	Expert Systems with Applications	1	1.786
24	A30, A45, A94, A95	ACM SIGSOFT Software Engineering Notes	4	7.143
25	A33	IEE Proceedings-Software	1	1.786
26	A34, A47	International Journal of System Assurance Engineering and Management	2	3.571
27	A36	Australasian Computer Science Conference	1	1.786

28	A37, A59	Arabian Journal for Science and Engineering	2	3.571
29	A38, A49, A55, A83, A84, A86, A91	Journal of Systems and Software	7	12.5
30	A39	International Conference on Advanced Software Engineering and Its Applications	1	1.786
31	A41	International Conference on Advances in Computing and Communications	1	1.786
32	A44	International Conference on Reliability, Infocom Technologies and Optimization (ICRITO)	1	1.786
33	A46	International Journal of Machine Learning and Cybernetics	1	1.786
34	A48	Chinese Journal of Electronics	1	1.786
35	A51, A93	IEEE Transactions on Software Engineering	2	3.571
36	A52	International Conference on Product Focused Software Process Improvement	1	1.786
37	A53, A64	Software Quality Journal	2	3.571
38	A54	CSI transactions on ICT	1	1.786
39	A57	Journal of the Brazilian Computer Society	1	1.786
40	A58	Innovations in Systems and Software Engineering	1	1.786
41	A60, A66, A81, A92	Empirical Software Engineering	4	7.143
42	A61, A65	International Conference on Computational Science and Its Applications	2	3.571
43	A63, A74	International Conference on Product-Focused Software Process Improvement	2	3.571
44	A69	International Conference on Software Reuse	1	1.786
45	A71	Proceedings - Working Conference on Reverse Engineering, WCRE	1	1.786
46	A72	i-Manager's Journal on Software Engineering	1	1.786
47	A73	Sādhanā	1	1.786
48	A75	International Conference on Computer and Communication Technology	1	1.786
49	A78	International Journal of Information Engineering and Electronic Business	1	1.786
50	A79	International Conference of the Chilean Computer Science Society (SCCC)	1	1.786
51	A80	International Conference on Evaluation and Assessment in Software Engineering	1	1.786
52	A82	International Conference on Information and Communication Systems	1	1.786
53	A85	Service Oriented Computing and Applications	1	1.786
54	A87	Procedia computer science	1	1.786
55	A88	Proceedings of the 11th Brazilian Symposium on Software Components, Architectures, and Reuse	1	1.786
56	A89	International Conference on Computer Engineering & Systems	1	1.786

Publications of the selected articles (95) are distributed as follows:

- 38 sources of publication with 1 article each (1.8%);
- 9 fonts with 2 articles each (3.6%);
- 5 sources with 3 articles each (5.4%);
- 2 fonts with 4 articles each (7.1%);
- 1 Source (Journal of Systems and Software) with 7 articles (12.5%);

- 1 source (Information and Software Technology) with 9 articles (16.1%).

Thus, there were many publications in just two sources, the Journal of Systems and Software published 7 articles (A38, A49, A55, A83, A84, A86, A91) and the Journal Information and Software Technology published 9 articles (A2, A14, A18, A40, A43, A56, A67, A76, A77), both concentrating 16 articles or 28.6% of the total.

3.2.2 Distribution of Articles per Year of Publication

The articles studied after the selection were grouped in Figure 2 according to the year of publication:

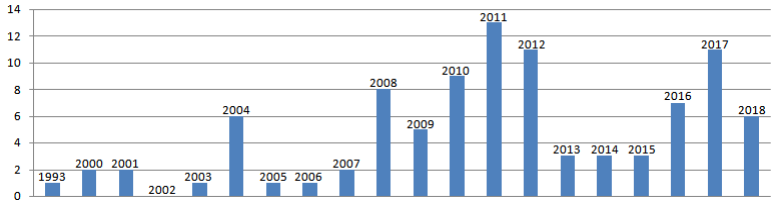

Figure 2: Distribution of Articles per Year of Publication

By the distribution of articles per year of publication, it is noticed that many articles (above 36.8%) were published in 2011, 2012 and 2017. That shows that the current research is still up to date.

3.2.3 Answers to the Research Questions

Through the full reading, analysis, and synthesis of the studies reported in the selected articles, it was possible to obtain answers to the research questions that are presented at the beginning of this revision.

RQ1: Are there tools available to automate the calculation of values corresponding to all cohesion and coupling metrics in OOS?

Answer: There are no tools available to automate the calculation of the values of all cohesion and coupling metrics. In the present work, 30 cohesion and 37 coupling metrics were identified, totaling 67 metrics, for which calculation tools were not found. On the other hand, 65 metrics of cohesion and 81 coupling metrics, totaling 146 metrics, do have tools for calculation.

RQ2: What tools are most used to automate the calculation of the values corresponding to the cohesion and coupling metrics in OOS?

Answer: In total, 64 tools were found to automate the calculation of the values corresponding to the cohesion and coupling metrics in OOS, as shown in Table 4.

Table 4: List of tools used in obtaining the value of cohesion and coupling metrics

#	Tool	Language	Cohesion metric	Coupling metric	Cited in article	Occurrences
1	Ada metric analyzer (LI; HENRY, 1993b)	Ada	LCOM2	MPC, RFC, DAC	A55	1
2	Analyst4j (http://www.codeswat.com)	Java	LCOM2	CBO, RFC	A24	1
3	Borland Together Tool (http://www.borland.com/us/products/together/index.html)	Java, C++	LCOM2, LCOM5, TCC	CBO, CF, RFC, CTA, CTM	A5, A20, A39, A41, A58, A68, A83	7
4	CCCC (http://cccc.sourceforge.net/)	Java, C++	LCOM2	CBO, CF, RFC	A4, A24	2
5	CKJM (http://www.spinellis.gr/sw/ckjm/doc/indexw.html)	Java	LCOM2, LCOM3, CAM	CBO, RFC, Ca, Ce, IC, CBM, DAC	A6, A24, A35, A44, A49, A59, A85, A95	8
6	Codesurfer (http://www.grammatech.com/products/codesurfer/)	C, C++	—	Overlap, Min-Coverage, Coverage, MaxCoverage , Tightness	A71	1
7	Cohesion Measure Tool (CMT) (NANDIGAM, 1995)	Java	LCCI, LCCD, LCC, TCC, CC, Coh, LCOM3	—	A42	1
8	COIN (https://sites.google.com/site/brcreddyse/Tools)	Java	LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, Coh, CAMC, TCC, LCC, DCd, DCi, CC, SCOM,LSCC, NHD, SNHD	CTI	A75	1
9	Columbus (http://www.frontendart.com/)	C++	LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, Coh, ICH, TCC, LCC, Co	PIM, ICP, CBO, MPC, OCMIC, DAC, OCAIC, ACMIC, ACAIC, RFC	A54, A92, A93	3
10	Concerto2/AUDIT (SEMA, 1998)	C++	LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, Coh, Co, LCC, TCC, ICH	CBO, CBO', RFC, RFC1,MPC, ICP, IH-ICP, NIH-ICP, DAC, DAC', IFCAIC, ACAIC, OCAIC, FCAEC, DCAEC, OCAEC, IFCMIC, ACMIC, OCMIC, FCMEC, DCMEC, OCMEC, IFMMIC, AMMIC, OMMIC, FMMEC, DMMEC, OMMEC	A66	1
11	DCRS tool (MALHOTRA et al., 2014)	Java	AMC, Ca, Ce, CAM, LCOM2, LCOM3	CBM, CBO, IC, RFC	A16	1

12	DynaMetrics (SINGH; SINGH, 2008)	Java, C++	LCOM2, Number of Private Methods Accessed at Runtime, Number of Public Methods Accessed at Runtime	CBO, DCBO RFC, Ca, Ce, DCBO, EOC, IOC,CQFS, OQFS,MQFS, Dynamic Efferent Coupling, Dynamic Afferent Coupling	A30, A32	2
13	Dynamic analyser tool (REDDY; CHANDRASEKHAR, 1999)	Java	LCOM1, LCOM2, TCC, RLCOM, DCC, DOC(x), DC_AM(x), DC_MA(x), DC_MM(x), DC_AA(x)	—	A31	1
14	Eclipse Metrics Plugin 1.3.6 (http://sourceforge.net/projects/metrics)	Java	LCOM2	—	A24	1
15	Eclipse Metrics Plugin 3.4 (http://eclipse-metrics.sourceforge.net)	Java	LCOM2, LCOM5	—	A24	1
16	EVOJAVA tool (OOSTERMAN; IRWIN; CHURCHER, 2011)	Java	—	RFC	A36	1
17	Extended CKJM (http://gromit.iiar.pwr.wroc.pl/ p_inf/ckjm/)	Java	CAM, LCOM2, LCOM3	Ca, CBM, CBO, Ce, IC, RFC	A29, A37, A61, A73	4
18	Frama-C (https://frama-c.com/)	C	Coverage, MaxCoverage, MinCoverage, Overlap, Tightness, WFC, NHD, SBFC	—	A14	1
19	GEN++ tool (http://web.cs.ucdavis.edu/ devanbu/genp/)	C++	LCOM2	—	A2	1
20	HYSS tool (http://salmosa.kaist.ac.kr/ hschae/HYSS/)	C++	LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, Coh, TCC, LCC	—	A2	1
21	IRC2M (POSHYVANYK; MARCUS, 2006)	C++	—	CCBC, CCBCm	A92	1
22	IRC3M (RAKESH; SUSHUMNA, 2012)	C++	C3	—	A93	1
23	JBOOMT (XIE et al., 2000)	C++	LCOM2	CBO, RFC	A48	1
24	JCAT (DEMARTINI; IOSIF; SISTO, 1999)	Java	—	PCC, GCC, ICC, TCC	A64	1
25	Jcolumbus (http://www.frontendart.com/)	Java	LCOM5	CBO, RFC	A41	1
26	JHawk (http://www.virtualmachinery.com/ jhawkprod.htm)	Java	LCOM5, LCbC	MPC, PACK, FIN,FOUT, Ca, Ce, CBO	A15, A88, A91	3
27	JMetrics (https://sourceforge.net/ projects/jmetrics/)	Java	—	ACAIC, ACMIC, DCAEC, DCMEC, OCAIC, OCAEC, OCMEC, OCMIC	A86	1
28	JPDA (http://java.sun.com/products/jpda)	Java	—	CBO, IC_CC, IC_CM, IC_CD, EC_CC, EC_CM, EC_CD	A11	1
29	Krakatau Metrics (http://www.powersoftware.com)	C++	—	RFC	A65	1
30	MASU (http://sourceforge.net/ projects/masu/)	Java, C#	LCOM2	CBO, RFC	A7	1
31	Metrics 1.3.6 (http://metrics.sourceforge.net/)	Java	LCOM, LCIC	—	A23, A95	1
32	NASA Metrics Data Program (http://www.mdp.ivv.nasa.gov)	Java, C++	LCOM2	CBO, RFC	A34, A63	2

33	OOMC (KANMANI; PRATHIBHA, 2001)	C++	CO, CO H, ICH, LCC, TCC, LCOM1, LCOM2, LCOM3, LCOM4, LCOM4	ACAIC, ACMIC, AMMIC, CBO, CBO', DAC, DAC', DCAEC, DCMEC, DMMEC, FCAEC, FCMEC, FMMEC, IC, ICP, IFCAIC, IFCMIC, IFMMIC, IHICP, MPC, NIHICP, OCAEC, OCAIC, OCMEC, OCMIC, OMMEC, OMMIC, RFC, RFC1	A45, A52, A56	3
34	OOMeter (http://www.ccse.kfupm.edu.sa/oometer/oometer/)	Java	LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, CBAMU, CCM, TCC, LCC, CAMC, Co, ECCM, OCC, PCC, RCI, CBMC	CBO	A12, A24	2
35	PATRicia (ETZKORN; DAVIS; LI, 1998)	C++	LCOM1, LCOM2	—	A2	1
36	Percerons Client (http://www.percerons.com/)	Java	LCOM2, CAM	DAC, RFC, MPC, DCC	A76	1
37	PROM (http://www.promtools.org/doku.php?id=prom651)	Java	LCOM2	CBO, RFC	A1	1
38	Promise Repository (http://promise.site.uottawa.ca/SERepository/)	C, C++	LCOM2	CBO, RFC	A74	1
39	Qualitas Corpus (http://www.qualitascorpus.com/)	Java	—	DAC, DAC', CBO, CBO', OCAIC, ACAIC	A3	1
40	Quality Measuring Tool (http://www.isc.ku.edu.kw/drjehad/research.htm)	Java	LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, LSCC, CC, SCOM, Coh, TCC, LCC, DCD, DCI, CBMC, ICBMC, OL2, OL3, PCCC, CAMC, NHD, SNHD, MMAC, SCC, TLCOM	RFC, MPC, DAC1, DAC2, OCMEC	A8, A18, A40, A43, A51, A60, A84, A87	8
41	SCCM (https://github.com/geekhack/sccm)	PHP, Java, C++, Java-Script	SCCM, COH	—	A70	1
42	Semmle (http://semmle.com)	Java	LCOM2, LCOM5	RFC	A24	1
43	Shimba reverse engineering (SYSTA, 1999)	Java	LCOM	CBO, RFC	A19	1
44	Source Code Metric (http://plugins.netbeans.org/plugin/42970/sourcecodemetrics)	Java	—	LCC, TCC	A72	1
45	SourceViz (https://sourceviz.wordpress.com/)	Java	—	Ca, Ce	A78	1
46	SRT (SANTOS; AFONSO; COSTA, 2016)	Java	LCOM2, LCOM4, TCC	Ca, Ce, CBO, RFC, MPC	A79	1
47	SSQAT (PAN et al., 2010)	Java	—	CBO, RFC	A50	1
48	TDTool (http://labsoft.dcc.ufmg.br/doku.php?id=about:tdtool)	Java	—	CBO	A80	1

49	tera-PROMISE Repository (http://openscience.us/repo/)	Java	LCOM2, LCOM3, CAM	RFC, Ca, Ce, CBO, CBM, IC	A13, A29, A77	3
50	UML Metrics Tool (GIRGIS; MAHMOUD; NOUR, 2009)	UML	—	DAC, DAC', OCAEC,OCAIC, ACAIC,DCAEC, ACAEC,DCAIC, ACMEC,OCMIC, DCMEC,OCMEC, ACMIC,DCC	A89	1
51	Understand (http://www.scitools.com/index.ph)	Java, C++	LCOM1, LCOM2, LCOM3, LCOM4, Co, Co', LCOM5, Coh, TCC , LCC, ICH, OCC, PCC, DCD, DCI, CAMC, CAMCs, iCAMCs, NHD, NHDs, iNHDs, SNHD, SNHDs, iSNHDs	FanIn, FanOut, RFC, CBO, RFC1, MPC, MPC', DAC, DAC', ICP, IH-ICP, NIH-ICP, ACMIC, OCAIC, OCAEC, OCMIC, OCMEC, AMMIC, DMMEC, OMMIC, OMMEC, CC, UCL, MPIC, MPEC,MMIC, MMEC, MAIC, MAEC,AFM, O_IFMPIC, O_AMPIC, O_OMPIC, O_FMPEC, O_DMPEC, O_OMPEC, MAF	A17, A22, A24, A26, A27, A46, A47, A62, A90	9
52	VizzAnalyzer (http://www.arisa.se)	Java	LCOM2	CBO, RFC	A24	1
53	SCA (SPRINGER et al., 2015)	C#	—	CBO, RFC, MPC, DAC, DAC1, ICP, ACAIC , DCAEC, ACMIC, DCMEC, AMMIC	A82	1
54	WebMetrics (http://www.nist.gov/webmetrics/)	Java, C, C++, Small-talk	LCOM2	CBO, RFC	A10	1
55	ES2 (http://www.psmsc.com/ESx.asp)	Java	OCAIC, OCAE, OCMIC, OCMEC, ACAIC, DCAEC, ACMIC, DCMEC	—	—	0
56	Essential Metrics (http://www.powersoftware.com/em/)	Java, C, C++	LCOM2	CBO, RFC	—	0
57	Imagix 4D (https://www.imagix.com/products/static-analysis-and-metrics.html)	Java, C, C++	LCOM2	CBO, RFC, Fan In,Fan Out	—	0
58	JDepend (https://github.com/clarkware/jdepend)	Java	—	Ca, Ce	—	0
59	JMT (http://jmt.tigris.org/)	Java	LCOM2	CF, CBO, RFC	—	0
60	Jpeek (https://github.com/yegor256/jpeek)	Java	CAMC, LCOM, OCC,PCC,MMAC, NHD, LCOM2, LCOM3, LCOM4, CCM,SCOM, TCC,TLCOM	—	—	0
61	Ndepend (https://www.ndepend.com/)	Java, C++	LCOM2, LCOM5	Ca,Ce	—	0
62	QMOOD++ (BANSIYA, 1997)	C++	CAM	DAM, DCC	—	0
63	Qualitas.class Corpus (http://java.labsoft.dcc.ufmg.br/qualitas.class/)	Java	LCOM5	Ca, Ce	—	0
64	SDMetrics (http://www.sdmetrics.com)	UML	—	Ca, Ce	—	0

The list of tools presented illustrates 4 more commonly used tools (among the 64 tools) to automate the calculation of the values corresponding to the cohesion and coupling metrics in OOS: Understand occupies the first place for being used in 9 articles (A17, A22, A24, A26, A27, A46, A47, A62, and A90), CKJM is second, used in 8 articles (A6, A24, A35, A44, A49, A59, A85, and A95), Quality Measuring Tool also appears in second, used in 8 articles (A8, A18, A40, A43, A51, A60, A84, and A87), and Borland Together Tool occupies the fourth place for being use in 7 articles (A5, A20, A39, A41, A58, A68, and A83). The other tools (60) were used in less than 5 articles.

RQ3: How many cohesion and coupling metrics in OOS are there in the literature?

Answer: According to the first SLM, in the literature there are at least 213 metrics of cohesion and coupling; among them, 95 metrics are about cohesion and 118 metrics are about coupling. Table 5 present the 95 cohesion metrics and Table 6 present the 118 coupling metrics.

Table 5: List of cohesion metrics and their tools

#	Cohesion Metrics	Tools and Articles	# Articles	# Tools
1	**A** (*Adhesiveness*) (BIEMAN; OTT, 1994)	—	0	0
2	**AAC** (*Attribute-Attribute Cohesion*) (DALLAL; BRIAND, 2010)	—	0	0
3	**AMC** (*Attribute-Method Cohesion*) (DALLAL; BRIAND, 2010)	DCRS tool (A16)	1	1
4	**APIU** (*API usage index*) (SARKAR; KAK; RAMA, 2008)	—	0	0
5	**C3** (*Conceptual Cohesion of Classes*) (MARCUS; POSHYVANYK; FERENC, 2008)	IRC3M (IR-based Conceptual Cohesion Class Measurement) (A93)	1	1
6	**CAM** (*Cohesion Among Methods of Class*) (BANSIYA; DAVIS, 2002)	CKJM (A29, A73), DCRS tool (A16), Extended CKJM (A37, A61),Percerons Client (A76), tera-PROMISE Repository (A13, A29), QMOOD++ (-).	8	6
7	**CAMC** (*Cohesion Among Methods in a Class*) (BANSIYA; DAVIS; ETZKORN, 1999)	COIN (COhesion, INheritance) (A75), OOMeter (A12), Quality Measuring Tool (A18, A43, A51, A60, A84), Understand (A17), QMOOD++ (-), jpeek (-).	8	6
8	**CAMCs** (*Cohesion among methods of a class with 'self' parameter*) (COUNSELL; SWIFT; CRAMPTON, 2006) (KAUR; SINGH, 2010)	Understand (A17)	1	1

9	**CBAMU** (*Cohesion Based on Attribute and Method Usage*) (ADAM, 2004)	OOMeter (A12)	1	1
10	**CBMC** (*Cohesion Based on Member Connectivity*) (CHAE; KWON; BAE, 2000) (XU; ZHOU, 2001)	OOMeter (A12), Quality Measuring Tool (A40, A60, A84).	4	2
11	**CC** (*Class Cohesion*) (BONJA; KIDANMARIAM, 2006)	Cohesion Measure Tool (CMT) (A42), COIN (COhesion, INheritance) (A75), Quality Measuring Tool (A8, A40, A43, A51, A60, A84), jpeek (-).	8	4
12	**CC** (*Component cohesion*) (VERNAZZA et al., 2000)	—	0	0
13	**CCM** (*Class Connection Metric*) (WASIQ, 2001) (JARALLAH; WASIQ; AHMED, 2001)	OOMeter (A12), jpeek (-).	1	2
14	**CDC** (*Class Level Cohesion Measures*) (GUPTA; CHHABRA, 2009)	—	0	0
15	**Co'** (ZHOU et al., 2012)	Understand (A17).	1	1
16	**Co** (*Connectivity*) (HITZ; MONTAZERI, 1995)	Columbus (A54), Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (A45, A52, A56), OOMeter (A12), Understand (A17).	7	5
17	**Coh** (*Cohesion of module*) (ALLEN; KHOSHGOFTAAR; CHEN, 2001)	Cohesion Measure Tool (CMT) (A42), COIN (COhesion, INheritance) (A75), Columbus (A93), Concerto2/AUDIT tools (A66), HYSS tool (A2), OOMC (Object Oriented Metric Calculator) (A45, A52, A56), Quality Measuring Tool (A18, A40, A43, A51, A60, A84), Understand (A17).	15	8
18	**Coverage** (WEISER, 1981) (OTT; BIEMAN, 1992) (OTT; THUSS, 1993)	Codesurfer (A71), Frama-C (A14).	2	2
19	**CPC** (*Contextual package cohesion*) (PONISIO, 2006)	—	0	0
20	**CR** (*Cohesion Ratio*) (FENTON; BIEMAN, 2014)	—	0	0
21	**CU** (*Common use*) (PONISIO; NIERSTRASZ, 2006)	—	0	0
22	**DC_AA(x)** (*Dynamic Cohesion due to Reference dependency between attributes*) (GUPTA; CHHABRA, 2011) (GUPTA, 2011)	Dynamic analyser tool (A31)	1	1
23	**DC_AM(x)** (*Dynamic Cohesion due to Write dependency of Attributes on Methods*) (GUPTA; CHHABRA, 2011) (GUPTA, 2011)	Dynamic analyser tool (A31)	1	1

24	**DC_MA(x)** (*Dynamic Cohesion due to Read dependency of Methods on Attributes*) (GUPTA; CHHABRA, 2011) (GUPTA, 2011)	Dynamic analyser tool (A31)	1	1
25	**DC_MM(x)** (*Dynamic Cohesion due to Call dependency between Methods*) (GUPTA; CHHABRA, 2011) (GUPTA, 2011)	Dynamic analyser tool (A31)	1	1
26	**DCACC** (*Dynamic Access Cohesion*) (GUPTA; CHHABRA, 2009)	—	0	0
27	**DCC** (*Dynamic Class Cohesion*) (GUPTA; CHHABRA, 2011) (GUPTA, 2011)	Dynamic analyser tool (A31)	1	1
28	**DCCALL** (*Dynamic Call Cohesion*) (GUPTA; CHHABRA, 2009)	—	0	0
29	**DCD** (*Degree of Cohesion-Direct*) (BADRI; BADRI, 2004)	COIN (COhesion, INheritance) (A75), Quality Measuring Tool (A8, A40, A43, A51, A60, A84), Understand (A17)	8	3
30	**DCDE** (*Cohesion Based on the Direct Relation*) (BADRI; BADRI; GUEYE, 2008)	—	0	0
31	**DCI** (*Degree of Cohesion-Indirect*) (BADRI; BADRI, 2004)	COIN (COhesion, INheritance) (A75), Quality Measuring Tool (A8, A40, A43, A51, A60, A84), Understand (A17)	8	3
32	**DCIE** (*Cohesion Based on the Indirect Relation*) (BADRI; BADRI; GUEYE, 2008)	—	0	0
33	**DCO** (*Degree of cohesion objects*) (MORRIS, 1989)	—	0	0
34	**DMC** (*Dependence Matrix based Class cohesion measure*) (WANG et al., 2005)	—	0	0
35	**DOC(x)** (*Dynamic Object Cohesion*) (GUPTA; CHHABRA, 2011) (GUPTA, 2011)	Dynamic analyser tool (A31)	1	1
36	**ECCM** (*Enhanced Class Connection Metric*) (WASIQ, 2001) (JARALLAH; WASIQ; AHMED, 2001)	OOMeter (A12)	1	1
37	**Fcoh** (*Functional coherence*) (MISIC, 2001)	—	0	0
38	**H** (*Relational Cohesion*) (LORENZ; KIDD, 1994)	Ndepend (-), SDMetrics (Software Design Metrics tool for the UML)	0	2
39	**HC** (*Hamming cohesiveness*) (ZHOU et al., 2009)	—	0	0

40	**iCAMCs** (ZHOU et al., 2012)	Understand (A17)	1	1
41	**ICBMC** (*Improved Cohesion Based on Member Connectivity*) (ZHOU et al., 2002)	Quality Measuring Tool (A40, A60, A84)	3	1
42	**ICH** (*Information-flow based Cohesion*) (LEE et al., 1995)	Columbus (A93), Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (A45, A52), Understand (A17)	5	4
43	**ILCO** (*Internal lack of cohesion*) (PONISIO; NIERSTRASZ, 2006)	—	0	0
44	**iNHDs** (ZHOU et al., 2012)	Understand (A17)	1	1
45	**IPSC** (*Index of package services cohesion*) (ABDEEN; DUCASSE; SAHRAOUI, 2011)	—	0	0
46	**iSNHDs** (ZHOU et al., 2012)	Understand (A17)	1	1
47	**LCbC** (*Lack of Concern-based Cohesion*) (SILVA et al., 2012)	Jhawk (A88)	1	1
48	**LCC** (*Loose Class Cohesion*) (BIEMAN; KANG, 1995)	Cohesion Measure Tool (CMT) (A42), COIN (COhesion, INheritance) (A75), Columbus (A93), Concerto2/AUDIT tools (A66), HYSS tool (A2), OOMC (Object Oriented Metric Calculator) (A45, A52), OOMeter (A12), Quality Measuring Tool (A8, A18, A40, A43, A51, A60, A84), Understand (A17)	16	9
49	**LCCD** (*Lack of Cohesion in the Class-Direct*) (BADRI; BADRI, 2004)	Cohesion Measure Tool (CMT) (A42)	1	1
50	**LCCI** (*Lack of Cohesion in the Class-Indirect*) (BADRI; BADRI, 2004)	Cohesion Measure Tool (CMT) (A42)	1	1
51	**LCD** (*Lack of cohesion - derived from LCOM*) (BADRI; BADRI; TOURE, 2011)	—	0	0
52	**LCIC** (*Lack of Coherence in Clients*) (MÄKELÄ; LEPPÄNEN, 2009)	Metrics 1.3.6 (an Eclipse metrics plugin) (A23)	1	1
53	**LCOM1** (*Lack of Cohesion in Methods 1*) (CHIDAMBER; KEMERER, 1991)	COIN (COhesion, INheritance) (A75), Columbus (A93), Concerto2/AUDIT tools (A66), Dynamic analyser tool (A31), HYSS tool (A2), OOMC (Object Oriented Metric Calculator) (A45, A52, A56), OOMeter (A12), PATRicia (Program Analysis Tool for Reuse) (A2), Quality Measuring Tool (A8, A18, A40, A43, A51, A60, A84), Understand (A17)	17	9

54	**LCOM2** (*Lack of Cohesion in Methods 2*) (CHIDAMBER; KEMERER, 1994)	Analyst4j (A24), Borland Together Tool (A39, A58, A83), CCCC (A4), CKJM (A6, A24, A28, A35, A44, A49, A59, A85, A95), Ada metric analyzer (A55), COIN (COhesion, INheritance) (A75), Columbus (A54, A93), Concerto2/AUDIT tools (A66), DCRS tool (A16), DynaMetrics (A30), dynamic analyser tool (A31), Eclipse Metrics Plugin 3.4 (A24), extended CKJM (A29, A37, A61, A73), GEN++ tool (A2), HYSS tool (A2), JBOOMT (A48), MASU (A7), Metrics 1.3.6 (an Eclipse metrics plugin) (A95), NASA Metrics Data Program (A34, A63), OOMC (Object Oriented Metric Calculator) (A45, A52), OOMeter (A12, A24), PATRicia (Program Analysis Tool for Reuse) (A2), Percerons Client (A76), PROM (A1), Promise Repository (A74), Quality Measuring Tool (A8, A18, A40, A43, A51, A60, A84, A87), Semmle (A24), Shimba reverse engineering (A19), SRT (Software Restructuring Tool) (A79), tera-PROMISE Repository (A13, A29, A77), Understand (A17, A22, A24, A27, A46, A47, A62, A90), VizzAnalyzer (A24), WebMetrics (A10), jpeek (-), Ndepend (-), Imagix 4D (-), Essential Metrics (-)	66	37
55	**LCOM3** (*Lack of Cohesion in Methods 3*) (LI; HENRY, 1993a)	Cohesion Measure Tool (CMT) (A42), COIN (COhesion, INheritance) (A75), Columbus (A93), Concerto2/AUDIT tools (A66), DCRS tool (A16), Extended CKJM (A29, A37, A73), HYSS tool (A2), OOMC (Object Oriented Metric Calculator) (A45, A52), OOMeter (A12), PATRicia (Program Analysis Tool for Reuse) (A2), Quality Measuring Tool (A8, A18, A40, A43, A51, A60, A84), tera-PROMISE Repository (A13, A29), Understand (A17), jpeek (-)	23	14
56	**LCOM4** (*Lack of Cohesion in Methods 4*) (HITZ; MONTAZERI, 1995)	COIN (COhesion, INheritance) (A75), Columbus (A93), Concerto2/AUDIT tools (A66), HYSS tool (A2), OOMC (Object Oriented Metric Calculator) (A45, A52, A56), OOMeter (A12), Quality Measuring Tool (A8, A40, A43, A51, A60, A84), SRT (Software Restructuring Tool) (A79), Understand (A17), jpeek (-)	16	10
57	**LCOM5** (*Lack of Cohesion in Methods 5*) (HENDERSON-SELLERS, 1996)	Borland Together TooL (A39), COIN (COhesion, INheritance) (A75), Columbus (A93), Concerto2/AUDIT tools (A66), Eclipse Metrics Plugin 1.3.6 (A24), Eclipse Metrics Plugin 3.4 (A24), HYSS tool (A2), Jcolumbus (A41), JHawk (A88), OOMC (Object Oriented Metric Calculator) (A45, A52, A56), OOMeter (A12), Quality Measuring Tool (A40, A43, A51, A60, A84), Semmle (A24), Understand (A17), Qualitas.class Corpus (-), Ndepend (-)	20	16
58	**LSCC** (*Low-level design Similarity-based Class Cohesion*) (DALLAL; BRIAND, 2012)	COIN (COhesion, INheritance) (A75), Quality Measuring Tool (A8, A40, A43, A51, A60, A84)	7	2
59	**MaxCoverage** (OTT; BIEMAN, 1992) (OTT; THUSS, 1993)	Codesurfer (A71), Frama-C (A14)	2	2
60	**MinCoverage** (OTT; BIEMAN, 1992) (OTT; THUSS, 1993)	Codesurfer (A71), Frama-C (A14)	2	2

61	**MMAC** (*Method-Method through Attributes Cohesion*) (DALLAL; BRIAND, 2010)	Quality Measuring Tool (A60, A84), jpeek (-)	2	2
62	**MMIC** (*Method–Method Invocation Cohesion*) (DALLAL; BRIAND, 2010)	—	0	0
63	**NHD** (*Normalised Hamming Distance*) (COUNSELL et al., 2002) (COUNSELL; SWIFT; CRAMPTON, 2006)	COIN (COhesion, INheritance) (A75), Frama-C (A14), Quality Measuring Tool (A18, A43, A51, A60, A84), Understand (A17), jpeek (-)	8	5
64	**NHDs** (*NHD with self parameter*) (COUNSELL; SWIFT; CRAMPTON, 2006) (KAUR; SINGH, 2010)	Understand (A17)	1	1
65	**NRC** (*Normalized relational cohesion*) (MARTIN, 2002)	—	0	0
66	**NRCI** (*Neutral Ratio of Cohesive Interactions*) (BRIAND; MORASCA; BASILI, 1994)	—	0	0
67	**Number of Private Methods Accessed at Runtime** (SINGH; SINGH, 2008)	DynaMetrics (A30)	1	1
68	**Number of Public Methods Accessed at Runtime** (SINGH; SINGH, 2008)	DynaMetrics (A30)	1	1
69	**OCC** (*Optimistic Class Cohesion*) (AMAN et al., 2002)	OOMeter (A12), Understand (A17), jpeek (-)	2	3
70	**ODC** (*Object Level Cohesion Measures*) (GUPTA; CHHABRA, 1993)	—	0	0
71	**OL2** (YANG, 2002)	Quality Measuring Tool (A40, A60, A84)	3	1
72	**OL3** (YANG, 2002)	Quality Measuring Tool (A40)	1	1
73	**OLn** (YANG, 2002)	ES2 (-)	0	1
74	**ORCI** (*Optimistic Ratio of Cohesive Interactions*) (BRIAND; MORASCA; BASILI, 1994)	—	0	0
75	**Overlap** (WEISER, 1981) (OTT; BIEMAN, 1992) (OTT; THUSS, 1993)	Codesurfer (A71), Frama-C (A14)	2	2
76	**PCC** (*Pessimistic Class Cohesion*) (AMAN et al., 2002)	OOMeter (A12), Understand (A17), jpeek (-)	2	3
77	**PCCC** (*Path Connectivity Class Cohesion*) (DALLAL, 2012a)	Quality Measuring Tool (A40, A60, A84)	3	1
78	**PCM** (*Package cohesion metrics*) (GORMAN, 2006)	—	0	0

79	**Pcoh** (*Package cohesion measurement*) (GUPTA; CHHABRA, 2012)	—	0	0
80	**PF** (*Index of package goal focus*) (ABDEEN; DUCASSE; SAHRAOUI, 2011)	—	0	0
81	**PRCI** (*Pessimistic Ratio of Cohesive Interactions*) (BRIAND; MORASCA; BASILI, 1994)	—	0	0
82	**RCI** (*Ratio of Cohesive Interactions*) (BRIAND; MORASCA; BASILI, 1994)	OOMeter (A12)	1	1
83	**RLCOM** (*Run-time Simple LCOM*) (MITCHELL; POWER, 2004b) (MITCHELL; POWER, 2004a)	Dynamic analyser tool (A31)	1	1
84	**SBFC** (*Similarity-Based Functional Cohesion)*, (DALLAL, 2009)	Frama-C (A14)	1	1
85	**SCC** (*Similar context cohesiveness*) (ZHOU et al., 2008)	—	0	0
86	**SCC** (*Similarity-based Class Cohesion*) (DALLAL; BRIAND, 2010)	Quality Measuring Tool (A18)	1	1
87	**SCCM** (*Scoped Class Cohesion Metric*) (WANJIKU; OKEYO; CHERUIYOT, 2016)	SCCM (Scoped Class Cohesion Metric) Software Tool (A70)	1	1
88	**SCOM** (*Sensitive Class Cohesion Metric*) (FERNÁNDEZ; PEÑA, 2006)	COIN (COhesion, INheritance) (A75), Quality Measuring Tool (A8, A40, A43, A51, A60, A84), jpeek (-)	7	3
89	**SFC** (*Strong functional cohesion*) (BIEMAN; OTT, 1994)	—	0	0
90	**SNHD** (*Scaled Normalised Hamming Distance*) (COUNSELL; SWIFT; CRAMPTON, 2006)	COIN (COhesion, INheritance) (A75), Quality Measuring Tool (A18, A51, A60), Understand (A17)	5	3
91	**SNHDs** (*Scaled NHD with self parameter*) (COUNSELL; SWIFT; CRAMPTON, 2006) (KAUR; SINGH, 2010)	Understand (A17)	1	1
92	**TCC** (*Tight Class Cohesion*) (OTT et al., 1995) (BIEMAN; KANG, 1995)	Borland Together TooL (A41), Cohesion Measure Tool (CMT) (A42), COIN (COhesion, INheritance) (A75), Columbus (A93), Concerto2/AUDIT tools (A66), Dynamic analyser tool (A31), HYSS tool (A2), OOMC (Object Oriented Metric Calculator) (A45, A52, A56), OOMeter (A12), Quality Measuring Tool (A8, A18, A40, A43, A51, A60, A84), SRT (Software Restructuring Tool) (A79), Understand (A17), jpeek (-)	20	13
93	**Tightness** (WEISER, 1981) (OTT; BIEMAN, 1992) (OTT; THUSS, 1993)	Codesurfer (A71), Frama-C (A14)	2	2

94	**TLCOM** (*Transitive Lack of Cohesion in Methods*) (DALLAL, 2011c)	Quality Measuring Tool (A87), jpeek (-)	1	2
95	**WFC** (*Weak functional cohesion*) (BIEMAN; OTT, 1994)	Frama-C (A14)	1	1

Table 6 presents 118 coupling metrics with the respective tools to automate their calculation and the quantity of articles selected and tools found for each metrics. There are some metrics for which tools were not found, as well as metrics that were mentioned in some works but were not found by the SLM.

Table 6: List of coupling metrics and their tools

#	Coupling Metrics	Tools and Articles	# Articles	# Tools
1	**AC** (*Association-induced coupling index*) (SARKAR; KAK; RAMA, 2008)	—	0	0
2	**ACAIC** (*Ancestors Class-Attribute Import Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Columbus (A92), Concerto2/AUDIT tools (A66), JMetrics (A86), OOMC (Object Oriented Metric Calculator) (A45), Qualitas Corpus (A3), UML Metrics Tool (A89), SCA (Static source Code Analysis) tool [A82], ES2 (-)	7	8
3	**ACMIC** (*Ancestor Class-Method Import Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Columbus (A92), Concerto2/AUDIT tools (A66), JMetrics (A86), OOMC (Object Oriented Metric Calculator) (A45, A52), UML Metrics Tool (A89), Understand (A17), SCA (Static source Code Analysis) tool (A82), ES2 (-)	8	8
4	**AFM** (*Actual Friend Methods*) (ENGLISH; CAHILL; BUCKLEY, 2012)	Understand (A26)	1	1
5	**AMMIC** (*Ancestors class Method–Method Import Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (A45, A52, A56), Understand (A17), SCA (Static source Code Analysis) tool// (A82)	6	4
6	**BCFI** (*Base-class fragility index*) (SARKAR; KAK; RAMA, 2008)	—	0	0
7	**Ca** (*Afferent couplings*) (MARTIN, 1994)	CKJM (A6, A44), DCRS tool (A16), DynaMetrics (A30), Extended CKJM (A29, A37, A73), JHawk (A91), SourceViz (A78), SRT (Software Restructuring Tool) (A79), tera-PROMISE Repository (A13, A29), SDMetrics (Software Design Metrics tool for the UML) (-), JDepend (-), Qualitas.class Corpus (-), Ndepend (-)	12	12
8	**CBM** (*Coupling Between Methods*) (TANG; KAO; CHEN, 1999)	DCRS tool (A16), Extended CKJM (A29, A37, A73), tera-PROMISE Repository (A13, A29)	6	3
9	**CBO'** (CHIDAMBER; KEMERER, 1991)	Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (A45), Qualitas Corpus (A3)	3	3

10	**CBO** (*Coupling between Object Classes*) (CHIDAMBER; KEMERER, 1991)	Analyst4j (A24), Borland Together TooL (A5, A58, A68, A83), CCCC (A4, A24), CKJM (A6, A24, A35, A44, A49, A59, A85, A95), Columbus (A54, A92), Concerto2/AUDIT tools (A66), DCRS tool (A16), DynaMetrics (A30, A32), Extended CKJM (A29, A37, A61, A73), JBOOMT (A48), Jcolumbus (A41), JHawk (A88), JPDA framework (A11), MASU (A7), NASA Metrics Data Program (A34, A63), OOMC (Object Oriented Metric Calculator) (A45, A52, A56), OOMeter (A24), PROM (A1), Promise Repository (A74), Qualitas Corpus (A3), Shimba reverse engineering (A19), SRT (Software Restructuring Tool) (A79), SSQAT (Software Structural Quality Analysis Tool) (A50), TDTool (A80), tera-PROMISE Repository (A13, A29, A77), Understand (A17, A22, A24, A27, A46, A47, A62, A90), VizzAnalyzer (A24), WebMetrics (A10), Imagix 4D (-), Essential Metrics (-), JMT (-)	56	31
11	**CBO_IUB** (*CBO Is Used By*) (KABAILI; KELLER; LUSTMAN, 2001)	—	0	0
12	**CBO_NA** (*CBO No Ancestors*) (KABAILI; KELLER; LUSTMAN, 2001)	—	0	0
13	**CBO_U** (*CBO Using*) (KABAILI; KELLER; LUSTMAN, 2001)	—	0	0
14	**CC** (ZHOU et al., 2012)	Understand (A17)	1	1
15	**CCBC** (*Conceptual Coupling Between two Classes*) (POSHYVANYK et al., 2009)	IRC2M tool (Information Retrieval based Conceptual Coupling Measurement) (A92)	1	1
16	**CCBCm** (*Maximum Conceptual Coupling Between two classes*) (POSHYVANYK et al., 2009)	IRC2M tool (Information Retrieval based Conceptual Coupling Measurement) (A92)	1	1
17	**CDC** (*Class level Dynamic Coupling*) (GUPTA, 2011)	—	0	0
18	**Ce** (*Efferent couplings*) (MARTIN, 1994)	DCRS tool (A16), DynaMetrics (A30), Extended CKJM (A29, A37, A73), JHawk (A91), SourceViz (A78), SRT (Software Restructuring Tool) (A79), tera-PROMISE Repository (A13, A29), SDMetrics (Software Design Metrics tool for the UML) (-), JDepend (-), Qualitas.class Corpus (-), Ndepend (-)	10	11
19	**CF** (*Coupling Factor*) (ABREU, 1995) (ABREU; MELO, 1996)	Borland Together TooL (A20), CCCC (A4), JMT (-)	2	3
20	**CQFS** (*Class Request For Service*) (ZAIDMAN; DEMEYER, 2004)	DynaMetrics (A30)	1	1
21	**CTA** (*Coupling through data abstraction*) (LI, 1998)	Borland Together TooL (A83)	1	1

22	**CTI** (*Coupling Through Inheritance*) (BRIAND; DALY; WUST, 1999)	COIN (COhesion, INheritance) (A75)	1	1
23	**CTM** (*Coupling through message passing*) (LI, 1998)	Borland Together TooL (A83)	1	1
24	**DAC** (*Data Abstraction* **Coupling**) (LI; HENRY, 1993b)	Ada metric analyzer (A55), Columbus (A92), Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (A45, A52), Percerons Client (A76), Qualitas Corpus (A3), Quality Measuring Tool (A60), UML Metrics Tool (A89), Understand (A17), SCA (Static source Code Analysis) tool (A82)	11	10
25	**DAC2** (*Data Abstraction Coupling 2*) (LI; HENRY, 1993b)	Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (A45, A52), Qualitas Corpus (A3), Quality Measuring Tool (A60), UML Metrics Tool (A89), Understand (A17), SCA (Static source Code Analysis) tool (A82)	8	7
26	**Dca** (*Dynamic Afferent Coupling*) (SINGH; SINGH, 2010)	—	0	0
27	**DCAEC** (*Descendant Class-Attribute Export Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Concerto2/AUDIT tools (A66), JMetrics (A86), OOMC (Object Oriented Metric Calculator) (A45, A52), UML Metrics Tool (A89), SCA (Static source Code Analysis) tool (A82), ES2 (-)	6	6
28	**DCAIC** (*Descendant class-attribute import coupling*) (BRIAND; DEVANBU; MELO, 1997)	UML Metrics Tool (A89)	1	1
29	**DCBO** (*Degree of dynamic coupling between two class and degree of dynamic coupling within a given set of classes*) (MITCHELL; POWER, 2005) (MITCHELL; POWER, 2003)	DynaMetrics (A30, A32)	2	1
30	**DCC** (*Direct Class Coupling*) (BANSIYA; DAVIS, 2002)	Percerons Client (A76), UML Metrics Tool (A89), QMOOD++ (-)	2	3
31	**DMC** (Direct Module Coupling) (HWA; LEE; KWON, 2009)	—	0	0
32	**DCM** (*Dynamic Coupling Metric*) (HASSOUN; JOHNSON; COUNSELL, 2004b) (HASSOUN; JOHNSON; COUNSELL, 2004a) (HASSOUN; COUNSELL; JOHNSON, 2005)	—	0	0
33	**DCMEC** (*Descendant Class-Method Export Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Concerto2/AUDIT tools (A66), JMetrics (A86), OOMC (Object Oriented Metric Calculator) (A45, A52), UML Metrics Tool (A89), SCA (Static source Code Analysis) tool (A82), ES2 (-)	6	6
34	**Degree of dynamic coupling between two classes at runtime** (MITCHELL; POWER, 2003)	—	0	0

35	**Degree of dynamic coupling within a given set of classes** (MITCHELL; POWER, 2003)	—	0	0
36	**DFC** (*DynamicFunction Coupling*) (APIWATTANAPONG; ORSO; HARROLD, 2005)	—	0	0
37	**DKC** (*Dynamic Key Class*) (SINGH; SINGH, 2010)	—	0	0
38	**DKCC** (*Dynamic Key Client Class*) (SINGH; SINGH, 2010)	—	0	0
39	**DKSC** (*Dynamic Key Server Class*) (SINGH; SINGH, 2010)	—	0	0
40	**DMMEC** (*Descendant class Method–Method Export Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (A45, A52, A56), Understand (A17)	5	3
41	**DOC** (*Dynamic Object Coupling b/w Objects*) (GUPTA, 2011)	—	0	0
42	**Dynamic Afferent Coupling** (SINGH; SINGH, 2008)	DynaMetrics (A30)	1	1
43	**Dynamic Efferent Coupling** (SINGH; SINGH, 2008)	DynaMetrics (A30)	1	1
44	**EC** (*Export Coupling*) (BRIAND; MORASCA; BASILI, 1994)	—	0	0
45	**EC_CC** (*Export Coupling at the Class level in distinct Classes*) (ARISHOLM; BRIAND; FOYEN, 2004) (ARISHOLM, 2002)	JPDA framework (A11)	1	1
46	**EC_CD** (*Export Coupling at the Class level in Dynamic messages*) (ARISHOLM; BRIAND; FOYEN, 2004) (ARISHOLM, 2002)	JPDA framework (A11)	1	1
47	**EC_CM** (*Export Coupling at the Class level in distinct Methods*) (ARISHOLM; BRIAND; FOYEN, 2004) (ARISHOLM, 2002)	JPDA framework (A11)	1	1
48	**EC_OC** (*Export Coupling at the Object level in distinct Classes*) (ARISHOLM; BRIAND; FOYEN, 2004) (ARISHOLM, 2002)	—	0	0
49	**EC_OD** (*Export Coupling at the Object level in Dynamic messages*) (ARISHOLM; BRIAND; FOYEN, 2004) (ARISHOLM, 2002)	—	0	0
50	**EC_OM** (*Export Coupling at the Object level in distinct Methods*) (ARISHOLM; BRIAND; FOYEN, 2004) (ARISHOLM, 2002)	—	0	0

51	**EOCx** (*Export Object Coupling*) (YACOUB; AMMAR; ROBINSON, 1999)	DynaMetrics (A30)	1	1
52	**FCAEC** (*Friends Class-Attribute Export Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (A45, A52)	3	2
53	**FCMEC** (*Friends Class-Method Export Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (A45, A52)	3	2
54	**FIN** (*Fan In*) (FENTON; BIEMAN, 2014)	Jhawk (A15), Understand (A22, A90)	3	2
55	**FMMEC** (*Friends Method-Method Export Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (A45, A52)	3	2
56	**FOUT** (*Fan Out*) (CHIDAMBER; KEMERER, 1994)	Jhawk (A15), Understand (A22, A90)	3	2
57	**GCC** (*Global Coupling Counts*) (OFFUTT; ABDURAZIK; SCHACH, 2008)	JCAT (Java Coupling Analysis Tool) (A64)	1	1
58	**IC** (*Import Coupling*) (BRIAND; MORASCA; BASILI, 1994)	COIN (COhesion, INheritance) (A75), DCRS tool (A16), Extended CKJM (A29, A37, A73), OOMC (Object Oriented Metric Calculator) (A45, A52), tera-PROMISE Repository (A13, A29)	9	5
59	**IC** (*Inheritance-based Intermodule coupling index*) (SARKAR; KAK; RAMA, 2008)	—	0	0
60	**IC** (*Inheritance coupling*) (TANG; KAO; CHEN, 1999)	—	0	0
61	**IC_CC** (*Import Coupling at the Class level in distinct Classes*) (ARISHOLM, 2002) (ARISHOLM; BRIAND; FOYEN, 2004)	JPDA framework (A11)	1	1
62	**IC_CD** (*Import Coupling at the Class level in Dynamic messages*) (ARISHOLM, 2002) (ARISHOLM; BRIAND; FOYEN, 2004)	JPDA framework (A11)	1	1
63	**IC_CM** (*Import Coupling at the Class level in distinct Methods*) (ARISHOLM, 2002) (ARISHOLM; BRIAND; FOYEN, 2004)	JPDA framework (A11)	1	1
64	**IC_OC** (*Import Coupling at the Object level in distinct Classes*) (ARISHOLM, 2002) (ARISHOLM; BRIAND; FOYEN, 2004)	—	0	0
65	**IC_OD** (*Import Coupling at the Object level in Dynamic messages*) (ARISHOLM, 2002) (ARISHOLM; BRIAND; FOYEN, 2004)	—	0	0

66	**IC_OM** (*Import Coupling at the Object level in distinct Methods*) (ARISHOLM, 2002) (ARISHOLM; BRIAND; FOYEN, 2004)	—	0	0
67	**ICC** (*Inheritance coupling counts*) (OFFUTT; ABDURAZIK; SCHACH, 2008)	JCAT (Java Coupling Analysis Tool) (A64)	1	1
68	**ICP** (*Information-flow-based coupling*) (LEE et al., 1995)	Columbus (A92), Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (A45, A52), Understand (A17), SCA (Static source Code Analysis) tool (A82)	6	5
69	**IFCAIC** (*Inverse Friend Class-Attribute Import Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (45, A52)	3	2
70	**IFCMIC** (*Inverse Friend Class-Method Import Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (45, A52)	3	2
71	**IFMMIC** (*Inverse Friend Method-Method Import Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (45, A52)	3	2
72	**IH-ICP** (*Inheritance Information-flow-based coupling*) (LEE et al., 1995)	Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (A45, A52), Understand (A17)	4	3
73	**IOCx** (*Import Object Coupling*) (YACOUB; AMMAR; ROBINSON, 1999)	DynaMetrics (A30)	1	1
74	**LCC** (*Loose Class Coupling*) (SHARMA; CHUG, 2015) (KAUR; MAINI, 2016)	Source Code Metric (A72)	1	1
75	**MAC** (*Method-Attribute Coupling*) (BOWMAN; BRIAND; LABICHE, 2010)	—	0	0
76	**MAEC** (*Method–Attribute Export Coupling*) (ENGLISH; CAHILL; BUCKLEY, 2012)	Understand (A26)	1	1
77	**MAF** (*Members Accessed by Friends*) (ENGLISH; CAHILL; BUCKLEY, 2012)	Understand (A26)	1	1
78	**MAIC** (*Method–Attribute Import Coupling*) (ENGLISH; CAHILL; BUCKLEY, 2012)	Understand (A26)	1	1
79	**MGC** (*Method-Generalization Coupling*) (BOWMAN; BRIAND; LABICHE, 2010)	—	0	0
80	**MII** (*Module interaction index*) (SARKAR; KAK; RAMA, 2008)	—	0	0
81	**MMC** (*Method-Method Coupling*) (BOWMAN; BRIAND; LABICHE, 2010)	—	0	0
82	**MMEC** (*Method–member interactions*) (ENGLISH; CAHILL; BUCKLEY, 2012)	Understand (A26)	1	1

83	**MMIC** (*Method–member interactions*) (ENGLISH; CAHILL; BUCKLEY, 2012)	Understand (A26)	1	1
84	**MPC'** (ZHOU et al., 2012)	Understand (A17)	1	1
85	**MPC** (*Message Passing Coupling*) (LI; HENRY, 1993b)	Ada metric analyzer (A55), Columbus (A92), Concerto2/AUDIT tools (A66), JHawk (A15), OOMC (Object Oriented Metric Calculator) (A45, A52), Percerons Client (A76), Quality Measuring Tool (A60), SRT (Software Restructuring Tool) (A79), Understand (A17), SCA (Static source Code Analysis) tool (A82)	11	10
86	**MPEC** (*Method–Procedure Export Coupling*) (ENGLISH; CAHILL; BUCKLEY, 2012)	Understand (A26)	1	1
87	**MPIC** (*Method–Procedure Import Coupling*) (ENGLISH; CAHILL; BUCKLEY, 2012)	Understand (A26)	1	1
88	**MQFS** (*Method Request For Service*) (SINGH; SINGH, 2008)	DynaMetrics (A30)	1	1
89	**NC** (*Non-API method closedness index*) (SARKAR; KAK; RAMA, 2008)	—	0	0
90	**NIH-ICP** (*Noninheritance Information-flow-based coupling*) (LEE et al., 1995)	Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (A45, A52), Understand (A17)	4	3
91	**O_AMPIC** (ENGLISH; CAHILL; BUCKLEY, 2012)	Understand (A26)	1	1
92	**O_DMPEC** (ENGLISH; CAHILL; BUCKLEY, 2012)	Understand (A26)	1	1
93	**O_FMPEC** (ENGLISH; CAHILL; BUCKLEY, 2012)	Understand (A26)	1	1
94	**O_IFMPIC** (ENGLISH; CAHILL; BUCKLEY, 2012)	Understand (A26)	1	1
95	**O_OMPEC** (ENGLISH; CAHILL; BUCKLEY, 2012)	Understand (A26)	1	1
96	**O_OMPIC** (ENGLISH; CAHILL; BUCKLEY, 2012)	Understand (A26)	1	1
97	**OCAEC** (*Others Class-Attribute Export Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Concerto2/AUDIT tools (A66), JMetrics (A86), OOMC (Object Oriented Metric Calculator) (A45, A52), UML Metrics Tool (A89), Understand (A17), ES2 (-)	6	6
98	**OCAIC** (*Others Class-Attribute Import Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Columbus (A92), Concerto2/AUDIT tools (A66), JMetrics (A86), OOMC (Object Oriented Metric Calculator) (A45, A52), Qualitas Corpus (A3), UML Metrics Tool (A89), Understand (A17), ES2 (-)	8	8
99	**OCMEC** (*Others Class-Method Export Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Concerto2/AUDIT tools (A66), JMetrics (A86), OOMC (Object Oriented Metric Calculator) (A45, A52), Quality Measuring Tool (A60), UML Metrics Tool (A89), ES2 (-)	6	6
100	**OCMIC** (*Others Class-Method Import Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Columbus (A92), Concerto2/AUDIT tools (A66), JMetrics (A86), OOMC (Object Oriented Metric Calculator) (A45, A52), UML Metrics Tool (A89), Understand (A17), ES2 (-)	7	7

101	**OMMEC** (*Other class Method–Method Export Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (A45, A52, A56), Understand (A17)	5	3
102	**OMMIC** (*Other class Method–Method Import Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (A45, A52, A56), Understand (A17)	5	3
103	**OPFS** (*Object Response for Service*) (YACOUB; AMMAR; ROBINSON, 1999)	—	0	0
104	**OQFS** (*Object Request For Service*) (YACOUB; AMMAR; ROBINSON, 1999)	DynaMetrics (A30)	1	1
105	**PAC** (*Percentage Active Classes*) (SINGH; SINGH, 2010)	—	0	0
106	**PACK** (*Number of imported packages*) (FOWLER et al., 1999)	Jhawk (A15)	1	1
107	**PCC** (*Parameter Coupling Counts*) (OFFUTT; ABDURAZIK; SCHACH, 2008)	JCAT (Java Coupling Analysis Tool) (A64)	1	1
108	**PIM** (BRIAND; WUST; LOUNIS, 1999)	Columbus (A92)	1	1
109	**RDE** (*Runtime export degree of coupling*) (MITCHELL; POWER, 2004a)	—	0	0
110	**RDI** (*Runtime import degree of coupling*) (MITCHELL; POWER, 2004a)	—	0	0
111	**RE** (*Runtime export coupling between objects*) (MITCHELL; POWER, 2004a)	—	0	0

112	**RFC** (*Response for a Class*) (CHIDAMBER; KEMERER, 1991)	Analyst4j (A24), Borland Together TooL (A58, A83), Concerto2/AUDIT tools (A66), CCCC (A4), CKJM (A6, A24, A28, A35, A44, A49, A59, A85, A95), Ada metric analyzer (A55), Columbus (A54), Concerto2/AUDIT tools (A66), DCRS tool (A16), DynaMetrics (A30), EVOJAVA tool (A36), Extended CKJM (A29, A37, A61, A73), JBOOMT (A48), Jcolumbus (A41), Krakatau Metrics (A65), MASU (A7), NASA Metrics Data Program (A34, A63), OOMC (Object Oriented Metric Calculator) (A45, A52, A56), Percerons Client (A76), PROM (A1), Promise Repository (A74), Quality Measuring Tool (A8, A18, A40, A43, A51, A60, A84, A87), SCCM (Scoped Class Cohesion Metric) Software Tool (A70), Semmle (A24), Shimba reverse engineering (A19), Source Code Metric (A72), SourceViz (A78), SRT (Software Restructuring Tool) (A79), SSQAT (Software Structural Quality Analysis Tool) (A50), tera-PROMISE Repository (A13, A29, A77), Understand (A17, A22, A27, A46, A47, A62, A90), VizzAnalyzer (A24), WebMetrics (A10), Imagix 4D (-), Essential Metrics (-), JMT (-)	63	36
113	**RFC1** (*Response for a Class 1*) (CHIDAMBER; KEMERER, 1994)	Concerto2/AUDIT tools (A66), OOMC (Object Oriented Metric Calculator) (A45, A52), Understand (A17)	4	3
114	**RI** (*Runtime import coupling between objects*) (MITCHELL; POWER, 2004a)	—	0	0
115	**TCC** (*Tight Class Coupling*) (SHARMA; CHUG, 2015) (KAUR; MAINI, 2016)	Source Code Metric (A72)	1	1
116	**TCC** (*Total coupling counts*) (OFFUTT; ABDURAZIK; SCHACH, 2008)	JCAT (Java Coupling Analysis Tool) (A64)	1	1
117	**TDOC** (*Total Dynamic Object Coupling*) (GUPTA, 2011)	—	0	0
118	**UCL** (ZHOU et al., 2012)	Understand (A17)	1	1

RQ4: What is the most used programming language in data collection of metrics?

Answer: Java is the programming language most used systems in data collection. Java was used in 67 articles: A1, A3, A4, A5, A6, A7, A8, A10, A11, A12, A13, A15, A16, A17, A18, A19, A20, A22, A23, A24, A26, A27, A29, A30, A31, A32, A34, A35, A36, A37, A39, A40, A41, A42, A43, A44, A46, A47, A49 , A50, A51, A58, A59, A60, A61, A62, A63, A64, A68, A70, A72, A73, A75, A76, A77, A78, A79, A80, A83, A84, A85, A86, A87, A88, A90, A91, and A95.

Among the 64 tools found for calculating cohesion and coupling, 46 (71.9% of the tools found) were compatible with systems

developed in Java. Analyst4j, CKJM, Cohesion Measure Tool (CMT), COIN (COhesion, INheritance), DCRS tool, DynaMetrics, dynamic analyser tool, Eclipse Metrics Plugin 1.3.6, Eclipse Metrics Plugin 3.4, EVOJAVA tool, extended CKJM, JCAT (Java Coupling Analysis Tool), Jcolumbus, JHawk, JMetrics, JPDA framework, MASU (Java/C#), Metrics 1.3.6 (an Eclipse metrics plugin), OOMeter, Percerons Client, PROM, Qualitas Corpus, Quality Measuring Tool, Semmle, Shimba reverse engineering, Source Code Metric, SourceViz, SRT (Software Restructuring Tool), SSQAT (Software Structural Quality Analysis Tool), TDTool, tera-PROMISE Repository, VizzAnalyzer, ES2, JMT, JDepend, jpeek, and Qualitas.class Corpus (a compiled version of the qualitas corpus).

C++ is the programming language that came second for being applied in 32 articles: A2, A4, A5, A10, A17, A20, A22, A24, A26, A27, A30, A32, A34, A39, A41, A45, A46, A47, A48, A52, A56, A58, A62, A63, A65, A68, A70, A74, A83, A90, A92, and A93.

The C language came in third place (in 4 articles: A10, A14, A71, A74), C# in fourth place (in 2 articles: A7, A82), the remaining languages were in the fifth place, namely: JavaScript and PHP (A70), Ada (A55), UML (A89) and Smalltalk (A10).

RQ5: Specifically, which metrics of cohesion or coupling are the most studied?

Answer: Table 7 presents the cohesion metrics most studied. The other cohesion metrics were used in less than 10 articles.

Table 7: The most commonly used cohesion metrics

#	Cohesion Metrics	# Articles	# Tools
1st	**LCOM2**	66	37
2nd	**LCOM3**	23	14
3rd	**LCOM5**	20	16
3rd	**TCC**	20	13
4th	**LCOM1**	17	9
5th	**LCOM4**	16	10
5th	**LCC**	16	9
6th	**Coh**	15	8

Table 8 presents the most studied coupling metrics. The other coupling metrics were used in less than 10 articles.

Table 8: The most commonly used coupling metrics

#	Coupling Metrics	# Articles	# Tools
1st	**RFC**	63	36
2nd	**CBO**	56	31
3rd	**Ca**	12	12
4th	**DAC**	11	10

4th	**MPC**	11	10
5th	**Ce**	10	11

This 1st SLM obtained 95 cohesion and 118 coupling metrics, producing a group of 213 cohesion and coupling metrics, thus surpassing the amount of cohesion and coupling metrics presented in related works in the literature.

The work with more metrics analyzed until now was (BRIAND; WÜST; LOUNIS, 2001) with 38 metrics (10 cohesion metrics and 28 coupling metrics), (GEETIKA; SINGH, 2014) with 34 coupling metrics only, and (AGGARWAL et al., 2009) with 28 metrics (7 cohesion metrics and 21 coupling metrics). Each of the other works found analyzed 10 cohesion and coupling metrics or less.

In the research of Etzkorn et al. (2004) 10 cohesion metrics and their respective tools (3 tools) for C++ language were identified.The focus of their research was the comparison between the metrics of cohesion, which is different of the discovering of the metrics and tools existing in the literature, as is the case of the first SLM.

In another research (RAGAB; AMMAR, 2010), 4 cohesion metrics and 10 coupling metrics were surveyed, including their respective tools (4 tools) for Java and C++ languages. Kayarvizhy (2016), selected 9 tools and 6 metrics of cohesion and coupling.

Among the 213 grouped metrics (present in this caption), 65 cohesion metrics and 81 coupling metrics, totaling 146 metrics, have tools to automate the calculation of the corresponding value. However, for 30 cohesion and 37 coupling metrics, totaling 67 metrics, the respective tools were not found.

By the 1st SLM, it is concluded that the most used tools (among the 64 tools found) are: Understand, CKJM, Quality Measuring Tool and Borland Together Tool.

The most used programming language for collecting metrics was Java, followed by C++. In relation to the tools, most of them also only works for Java.

There are some metrics that have been more studied compared to others. Among the 95 cohesion metrics, the most studied (metrics found in more many articles published) were LCOM2, LCOM3, LCOM5, TCC, LCOM1, LCOM4, LCC, and Coh. The other cohesion metrics were used in less than 10 articles. Among the 118 coupling metrics, the most studied were RFC, CBO, Ca, DAC, MPC and Ce. The other coupling metrics were used in less than 10 articles.

4 APPLICATION OF COHESION AND COUPLING METRICS IN THE FAULT PREDICTION

There are many metrics proposed in literature to measure cohesion and coupling in object-oriented systems (OOS). Some of those metrics are useful for detecting fault-prone systems. However, assessing the utility of such metrics is not a trivial task.

The present chapter makes a careful analysis of the application of cohesion and coupling metrics in fault prediction, to allow the identification of useful metrics. This could ease the planning and elaboration of a new prediction model for fault prone systems. A systematic literature mapping (SLM) was conducted.

4.1 METHODOLOGICAL PROCEDURES (2ND SLM)

In the present chapter, to answer the proposed research questions, a SLM was carried out following the procedures presented in (KITCHENHAM, 2004), with three phases (Figure 3): planning, conducting the SLM, and presenting the results of the SLM.

Figure 3 presents a summary of all phases of the systematic mapping in order. Further details about those phases are presented in the following subsections.

4.1.1 Planning the Review

The planning phase went through two stages, namely the identification of the need for a SLM and the development of the SLM protocol.

4.1.1.1 Identification of the need for SLM

Object-oriented systems (OOS) are developed considering their internal qualities, such as cohesion and coupling and their external qualities, as for example, the propensity for faults.

Cohesion and coupling in software are measured through different metrics proposed in the literature. Some of these metrics may be useful in predicting fault-prone systems. Unfortunately, assessing their utility is not a trivial task.

There are dozens of metrics of cohesion and coupling proposed, and each year new metrics appear. It is possible that some of those metrics measure almost the same aspects as others, or that some are better predictors of faults than others. Therefore, there is

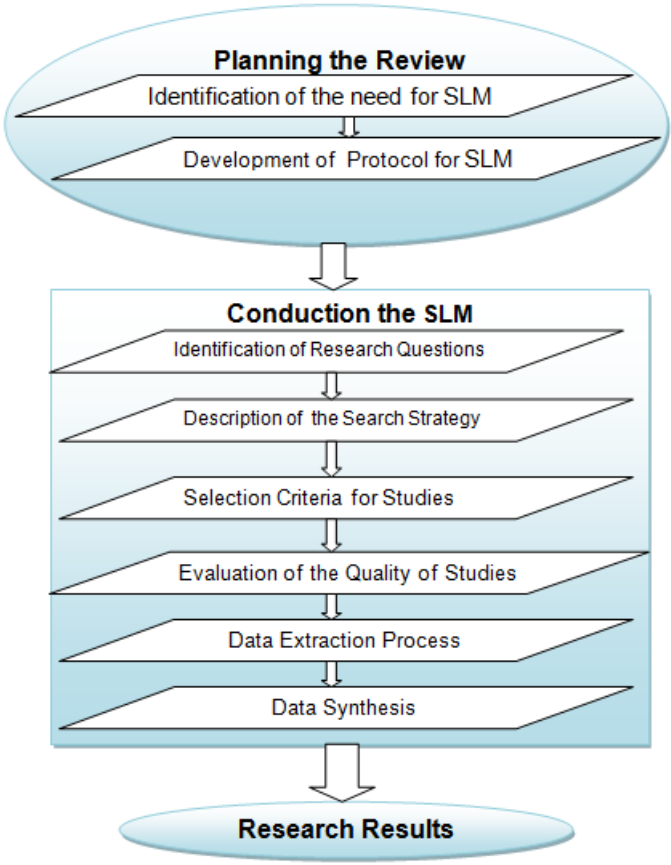

Figure 3: Process of systematic mapping

a constant need to find and evaluate the metrics that can be useful in predicting fault-prone OOS.

A search on databases (Scopus, Portal of Periodicals CAPES/MEC, and Google Scholar) was carried out in order to find similar secondary studies on the subject. Studies have not been found with scope and period similar to those of the revision proposed in this work, as is explained below.

4.1.1.2 Development of the Protocol SLM

The Protocol of the second SLM developed in this work, followed the following steps: identification of research questions, description of the search strategy, criteria for selection of articles, evaluation of the quality of articles, process of data extraction, and data synthesis.

4.1.2 Conducting the SLM

The second SLM of this work was conducted by the steps identified in Section 4.1.1.2. The details of each step are presented in the next subsections obeying the same sequence.

4.1.2.1 Identification of Research Questions

The research questions defined in this study are: (RQ1) What are the cohesion and coupling metrics applied in fault prediction studies? (RQ2) What are the cohesion and coupling metrics most often applied in fault prediction studies? (RQ3) Which systems are most commonly used to extract data? (RQ4) What is the most used programming language in data collection (metrics and faults)? (RQ5) What tools are the most used to measure cohesion and coupling?

4.1.2.2 Description of the Search Strategy

For the realization of the search, its scope was initially defined in terms of period, databases, and search strategy. The definition of the scope of the research was influenced by the goals of the SLM in terms of obtaining the most recent studies. The terms of the scope are described below:

a) Time interval: the search was restricted to articles published in the period from 2014 to 2018 to meet the goals of the research regarding obtaining recent data. The choice for this period (2014 to 2018) is due to the fact that related studies found were based in the period from 1991 to 2015.

b) Electronic databases: three electronic databases, namely: Scopus, Portal of Periodicals CAPES/MEC, and Google Scholar were selected as primary sources for this research. The electronic databases that have been selected have connections to other databases and therefore provide comprehensive sources for events (conferences, workshops and symposia) and journals. The integral texts of all the articles found are available on those bases.

c) Search strategy: automatic research was carried out in the electronic databases with the goal of finding articles relevant to the present work. The searches were performed based on a string, formed by keywords and by logical operators "and" and "or". The words used were: cohesion metrics, coupling metrics, object oriented system, fault prediction, fault prone, fault proneness.

The search string used in the second SLM was: (cohesion OR coupling) AND (metric OR measure) AND (system OR software) AND ("object oriented" OR oo) AND (relationship OR correlation) AND ("fault prediction" OR "fault proneness" OR "fault prone").

4.1.2.3 Selection Criteria for Studies

After the search carried out in the three databases (Scopus, Portal of Periodicals CAPES/MEC, and Google Scholar), it follows the preprocessing and then the selection of the articles by the application of inclusion criteria (IC) and exclusion criteria (EC) as presented below:

IC1: Publications in which the focus is related to fault-proneness based on the cohesion or coupling metrics. **IC2:** Publications on journals, conferences, or events in Computer Science that present results, evaluations, or validations. **IC3:** When finding articles with the same authors and similar research, in different events, only the most recent was included. **EC1:** Publications in which the focus is not related to the fault-proneness based on the cohesion or coupling metrics. **EC2:** Articles presenting proposals, ideas, tools, or strategies without presenting results, evaluations, or validations. **EC3:** Publications of thesis papers, dissertations, books, reports, and other works that are not classified as scientific articles. **EC4:** Duplicated or similar articles and literature review articles.

Based on the criteria for inclusion and exclusion of primary articles, it was possible to carry out the selection of articles without losing the focus of this research. The selection of articles went through four steps, obtaining the results mentioned below.

In the first step, the search string was inserted in the search engines of Scopus, Portal of Periodicals CAPES/MEC, and Google Scholar. Initially, 19 articles of Scopus, 1740 articles of Google Scholar and 71 Articles of Portal of Periodicals CAPES/MEC were identified, in a total of 1830 articles.

In the second step, the articles published before 2014 were eliminated, including 3 articles of Scopus, 606 of Google Scholar, and 26 articles from Portal of Periodicals CAPES/MEC, with a total

of 635 articles, corresponding to 35% of the result obtained in the first step.

In the third step, the titles, summaries, and keywords of each article (635) were read. After that, 60 articles were pre-selected (corresponding to 9.5% of the result obtained in the second stage), being 2 belonging to Scopus, 49 to Google Scholar and 9 to Portal of Periodicals CAPES/MEC.

During the reading of the third stage, the inclusion (IC) and exclusion (EC) criteria were applied. Most deleted articles (537 in total) fit the exclusion criteria EC2 and EC4.

In the fourth step, the results and/or conclusion sections of the 60 pre-selected articles were read. Most excluded articles (31) fit the exclusion criteria EC2 and EC4. It remained only 1 article from Scopus, 24 from Google Scholar, and 4 from the Portal of Periodicals CAPES/MEC, totaling 29 articles selected and detailed in Table 9, corresponding to 4.6% of the total obtained in the second stage.

The 29 selected articles are: **A1:** (SINGH; KAUR; MALHOTRA, 2014), **A2:** (RATHORE; GUPTA, 2014), **A3:** (JIN; JIN, 2014), **A4:** (KAUR; KAUR, 2014), **A5:** (KAUR; PAUL, 2014), **A6:** (SURESH; KUMAR; RATH, 2014), **A7:** (KUMARI; RAJNISH, 2015b), **A8:** (ZHAO et al., 2015), **A9:** (KAUR; SINGH, 2015), **A10:** (YANG et al., 2015), **A11:** (KUMARI; RAJNISH, 2015a), **A12:** (LEE; LI; LI, 2016), **A13:** (ARAR; AYAN, 2016), **A14:** (HUSSAIN et al., 2016), **A15:** (ISONG; IFEOMA; MBODILA, 2016), **A16:** (YOHANNESE; LI, 2017), **A17:** (RATHORE; KUMAR, 2017a), **A18:** (KUMAR; MISRA; RATH, 2017), **A19:** (ANWER et al., 2017), **A20:** (KUMAR et al., 2017), **A21:** (BASSEY, 2017), **A22:** (GOYAL; CHANDRA; SINGH, 2017), **A23:** (DALLAL, 2017), **A24:** (BOUCHER; BADRI, 2017), **A25:** (SHATNAWI, 2017), **A26:** (RATHORE; KUMAR, 2017b), **A27:** (ZHAO et al., 2017), **A28:** (KAUR; KAUR, 2018) e **A29:** (DALLAL; MORASCA, 2018).

Table 9: Selected Articles

#	Article title	Database	Citation (14/04/2018)
A1	A comparative study of models for predicting fault proneness in object-oriented systems	Scopus	4
A2	A comparative study of feature-ranking and feature-subset selection techniques for improved fault prediction	Google Scholar	8
A3	Applications of fuzzy integrals for predicting software fault-prone	Google Scholar	5
A4	Empirical evaluation of machine learning algorithms for fault prediction	Google Scholar	7
A5	Fault Prediction Modeling using Object-Oriented Metrics: An Empirical Study	Google Scholar	0

A6	Statistical and machine learning methods for software fault prediction using CK metric suite: a comparative analysis	Google Scholar	16
A7	A new approach to find predictor of software fault using association rule mining	Google Scholar	1
A8	An empirical analysis of package-modularization metrics: Implications for software fault-proneness	Google Scholar	11
A9	Analysis of CK metrics thresholds to predict faults using log transformation	Google Scholar	1
A10	Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? An empirical study	Google Scholar	20
A11	Investigating the Effect of Object-oriented Metrics on Fault Proneness Using Empirical Analysis	Google Scholar	1
A12	An Investigation of Essential Topics on Software Fault-Proneness Prediction	Google Scholar	1
A13	Deriving thresholds of software metrics to predict faults on open source software: Replicated case studies	Google Scholar	9
A14	Detection of fault-prone classes using logistic regression based object-oriented metrics thresholds	Google Scholar	2
A15	Supplementing Object-Oriented software change impact analysis with fault-proneness prediction	Google Scholar	2
A16	A Combined-Learning Based Framework for Improved Software Fault Prediction	Google Scholar	6
A17	A study on software fault prediction techniques	Google Scholar	3
A18	An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes	Portal of Periodicals CAPES/MEC	2
A19	Effect of coupling on software faults: An empirical study	Google Scholar	0
A20	Effective fault prediction model developed using Least Square Support Vector Machine (LSSVM)	Portal of Periodicals CAPES/MEC	0
A21	Enhancing Software Maintenance via Early Prediction of Fault-Prone Object-Oriented Classes	Google Scholar	1
A22	Fuzzy inferencing to identify degree of interaction in the development of fault prediction models	Google Scholar	4
A23	Predicting Fault-Proneness of Reused Object-Oriented Classes in Software Post-Releases	Google Scholar	0
A24	Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison	Google Scholar	0
A25	The application of ROC analysis in threshold identification, data imbalance and metrics selection for software fault prediction	Google Scholar	2
A26	Towards an ensemble based system for predicting the number of software faults	Google Scholar	4
A27	Understanding the value of considering client usage context in package cohesion for fault-proneness prediction	Google Scholar	4
A28	An empirical evaluation of classification algorithms for fault prediction in open source projects	Portal of Periodicals CAPES/MEC	0
A29	Investigating the impact of fault data completeness over time on predicting class fault-proneness	Portal of Periodicals CAPES/MEC	0

After the fourth stage, all 29 articles were read in full. In the next section, we continue the SLM in order to extract quality data from the articles.

4.1.2.4 Evaluation of the Quality of Studies

According to the suggestions given in (WEN et al., 2012), a quality questionnaire was formulated to assist in evaluating the quality of the selected articles.

Evaluate the quality of primary articles, according to Kitchenham (2004) is important, because in addition to clarifying the criteria for inclusion and exclusion of articles, it can explain whether the quality of selected articles interferes with their result. It is a way of weighing the importance of individual studies when they are consolidated, leading to the interpretation of the results and influencing the inferences that can lead to future publications.

The answers to the proposed quality assessment questions include the values: 1 (yes), 0.5 (partially), and 0 (not). The questions are as follows:

Q1: Is the study reviewed by peers? **Q2:** Are the objectives of the research clearly defined? **Q3:** Does the study explicitly describe the availability of tools used to calculate metrics? **Q4:** Does the study explicitly indicate the systems used to extract data? **Q5:** Does the study explicitly indicate which techniques are used for the prediction of faults? **Q6:** Are the results and conclusions clearly defined? **Q7:** Are the limitations of the study specified? **Q8:** Is the search methodology repeatable? **Q9:** Is there any technique applied to validate the study? **Q10:** Does the article have an appropriate number of average [1] quotes per year?

After assigning grades to each issue, the final score is obtained in the range from 0 to 10 by summing the 10 notes.

4.1.2.5 Data Extraction Process

The data extraction included the filling of a form for each of the selected primary articles to decide which research query was answered.

In the form appear the name of the author, title, publication place, year of publication, cohesion or coupling metric used, tool used, system used to extract data, system programming language, technique applied in the prediction of faults and the form of validation.

[1] To obtain the average it was necessary to add the total number of quotations obtained by all articles published in the same year. The result of the sum was divided by the quantity of selected articles published that same year. The average value of the publication, rounded to the nearest integer, was then compared with the quotes that each article published in the same year obtained to finally verify if the quotations of the article are below the average (grade = 0), above the average (grade = 1) or equal to the average (grade = 0.5).

4.1.2.6 Data Synthesis

The data synthesis was carried out with the goal of accumulating and combining facts and figures of the selected primary articles to formulate a response and to resolve the research questions, according to the definition of the synthesis (WEN et al., 2012).

The articles were analyzed, and both quantitative data, including the amount of metrics (cohesion and coupling) and used systems, as well as qualitative data that included the criteria applied in the post-selection of the articles were assessed. During the synthesis, to answer the research questions of this work, visualization techniques were used, such as column chart and tables to summarize the presentation of the results.

4.2 RESEARCH RESULTS (2ND SLM)

The results obtained for the selected primary articles are presented in this section. The presentation structure starts with an overview of each primary article, including the classification corresponding to the article obtained based on the scores assigned to the quality assessment issues.

After the overview of each primary article it follows the answers to the research questions.

4.2.1 Description of Primary Studies (Post-Selected)

According to Kitchenham (2004), the quality assessment should minimize the bias and increase the internal and external validity of the research.

Scores were attributed to the quality evaluation questions of each selected article (29 articles). To ease the classification, the scores were grouped into: high ($8 < \text{grade} \leq 10$), reasonable ($6 < \text{grade} \leq 8$), medium ($4 < \text{grade} \leq 6$), and low ($0 < \text{grade} \leq 4$).

Each article can have a maximum score of 10 and a minimum of 0. Among the 29 selected articles, a post-selection of 24 articles was carried out based on quality. Thus, 5 articles were excluded because they presented low quality. Figure 4, presents the summary of the quality of the 29 articles (selected) and 24 articles (post-selected).

The selected articles, present high percentages (equal to or greater than 80%) for 6 (six) quality issues, notably Q2, Q4, Q5, Q6, Q8, and Q9 (96.6%, 93.1%, 94.8%, 96.6%, 96.6%, and 84.5%, re-

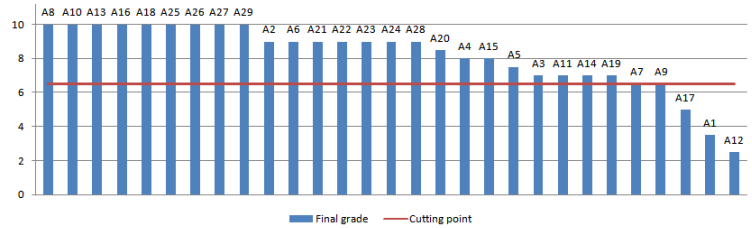

Figure 4: Results of the quality assessment of the selected and post selected articles.

spectively), while the post-selected articles (Figure 4), present high percentages in 7 (seven) issues, notably Q2, Q3, Q4, Q5, Q6, Q8, and Q9 (95.8%, 83.3%, 100%, 100%, 100%, 100%, and 95.8%, respectively).

Comparing the selected and post selected articles, it can be noticed an increase in the quality of the post selected articles, due to the exclusion of the 5 articles (A1, A7, A9, A12, and A17) whose quality is weak, with only 3 (three) issues with high percentages of quality, notably (Table 10) Q2, Q6, and Q8 (100%, 80%, and 80%, respectively).

Table 10: Quality of the papers excluded in post selection

Questions	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Grade 1.0	0	5	2	3	2	4	0	3	1	1
Grade 0.5	0	0	0	0	3	0	0	2	1	0
Grade 0.0	5	0	3	2	0	1	5	0	3	4
Total grade (0-5)	0	5	2	3	3.5	4	0	4	1.5	1
% of quality	0	100	40	60	70	80	0	80	30	20

The post selected articles show that the research goals are clearly defined and that the studies explicitly describe the availability of the tools used to calculate the metrics. They also show that the studies explicitly indicate the systems used to extract data and the techniques used for the prediction of faults. The results and conclusions are clearly defined, and the research procedures are repeatable. Finally, it can also be noticed that techniques were applied to validate the studies.

4.2.2 Publishing Sources

Table 11 summarizes the details of the publications in conferences and journals, corresponding to the post selected articles.

Table 11: Details of Publications of Post-selected Articles

#	Article	Source of Publication	Total	(%)
1	A2	India Software Engineering Conference (ISEC)	1	4.167

12	A16	International Journal of Computational Intelligence Systems	1	4.167
13	A18	Computer Standards & Interfaces	1	4.167
14	A19	International Conference on Communication, Computing and Digital Systems (C-CODE)	1	4.167
15	A20	The Journal of Systems Software	1	4.167
16	A21	International Journal of Software Engineering and Knowledge Engineering	1	4.167
17	A22 and A28	Journal of King Saud University – Computer and Information Sciences	2	8.333
18	A23	Arabian Journal for Science and Engineering	1	4.167
19	A24 amd A29	Information and Software Technology	2	8.333
20	A25	Innovations in Systems and Software Engineering	1	4.167
21	A27	Automated Software Engineering	1	4.167

The publications sources of the post selected articles are evenly distributed in 18 sources with 4.2% each, and in three sources with 8.3% each. The 18 sources with the lowest percentage of publication had only one article each. On the other hand, the three sources with the highest number of publications were: Expert Systems with Applications (published A13 and A26), Journal of King Saud University – Computer and Information Sciences (published A22 and A28) and Information and Software Technology (published A24 and A29).

4.2.3 Distribution of Articles per Year of Publication

The post selected articles were grouped according to the year of publication (Figura 5): **2014:** A2, A3, A4, A5, and A6: five articles (20.8%); **2015:** A8, A10, and A11: three articles (12.5%); **2016:** A13, A14, and A15: three articles (12.5%); **2017:** A16, A18, A19, A20, A21, A22, A23, A24, A25, A26, and A27: eleven articles (45.8%); **2018:** A28 and A29: two articles (8.3%).

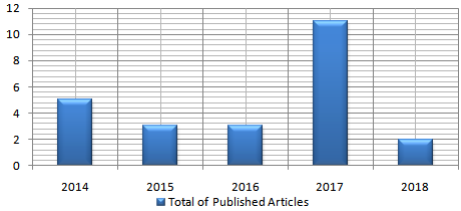

Figure 5: Distribution of Articles per Year of Publication

For the distribution of articles per year of publication, a greater concentration of articles published in 2017 (45.8%) can be seen. That suggests a growing interest in the theme. The year 2018 had

fewer articles, but it should be considered that the survey was carried out in the first trimester of 2018, and even so the quantity of articles almost compares to the quantities obtained from 2015 to 2016.

4.2.4 Answers to Research Questions

By reading (in full) and by the analysis and synthesis of the post selected studies, it was possible to obtain answers to the research questions that were presented at the beginning of this work.

RQ1: What are the cohesion and coupling metrics applied in fault prediction studies?

Answer: The cohesion and coupling metrics applied in fault prediction studies are:

- Cohesion metrics (28): AMC, APIU, CAM, CC, Coh, Coverage, CPC, CR, CU, DCO, Fcoh, HC, ILCO, IPSC, LCOM, LCOM3, MaxCoverage, NHD, NRC, Overlap, PCM, Pcoh, PF, SBFC, SCC, SFC, Tightness, and WFC.

- Coupling metrics (12): AC, BCFI, Ca, Ce, CBM, CBO, FOUT, IC, MII, MPC, NC, and RFC.

RQ2: Which are the cohesion and coupling metrics most often applied in fault prediction studies?

Answer: The cohesion and coupling metrics most often applied in fault predictions are (Figure 6 and 7):

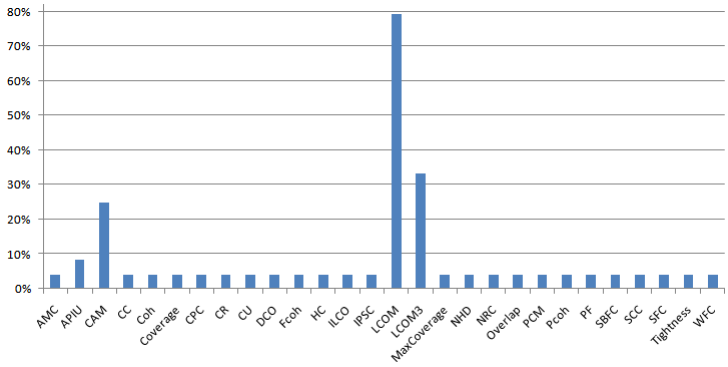

Figure 6: The cohesion metrics applied in fault prediction studies

- Cohesion metrics: LCOM (applied in 79.2% of studies), LCOM3 (applied in 33.3% of studies), and CAM (applied in 25% of studies). Each of the other metrics was applied in less than 8.5% of the studies.

- Coupling metrics: CBO (applied in 83.3% of studies), RFC (applied in 79.2% of studies), Ca (applied in 45.8% of studies), Ce (applied in 41.7% of studies), IC (applied in 33.3% of studies), and CBM applied in 25% of studies). Each of the other metrics was applied in less than 4.5% of studies.

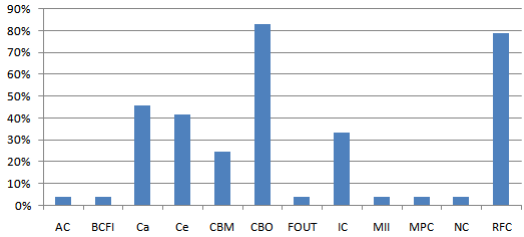

Figure 7: The coupling metrics applied in fault prediction studies

RQ3: What systems are the most used to extract data?

Answer: The PROMISE repository is the most used system to extract data (36.4% of studies). Secondly are SOURCE FORGE and Bug Prediction Dataset in the same proportion (12.1% each). In third appears the NASSA repository (9.1%). Each of the other systems (Bash, Eclipse Bug Data!, Gcc-core, Gimp, Github, GIRA, PROMISE repository of University of Ottawa), Subversion and Vim, were used in less than 6.5% of the studies. Figure 8 presents further details.

RQ4: What is the programming language most used in data collection (metrics and faults)?

Answer: Java was the most used language (Figure 9),appearing in 94.6%of the 277 projects analyzed. The Languages C, C++, and a combination of Java and C++ were applied in less than 3.5% projects.

RQ5: What tools are most used to measure cohesion and coupling?

Answer: The tools most used to measure cohesion and coupling are CKJM (used in 37.5% of studies) and Understand (used in

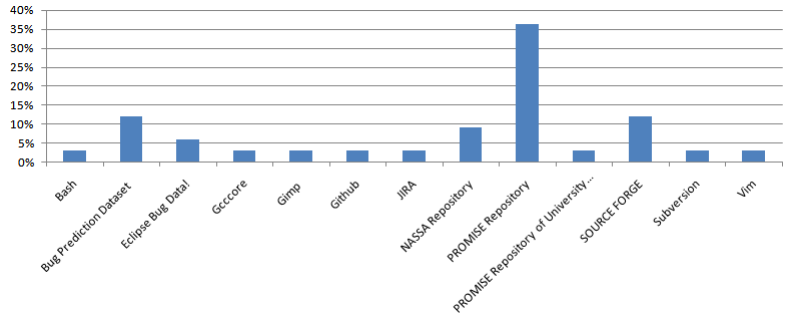

Figure 8: The most used system to extract data

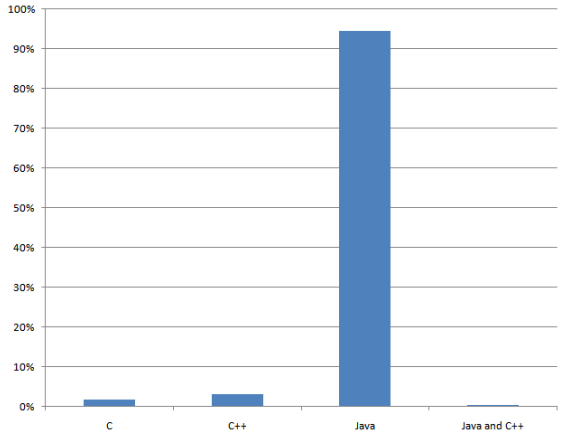

Figure 9: The programming language most used in data collection (metrics and faults)

12.5% of studies). Each of the other tools (Frama-C, JArchitect, JDepend, JHawk NDepend, PROMISE, RefactorIT, and DMS Software Reengineering Toolkit), were used in less than 6.5% of the studies (Figure 10).

Among the metrics found, the usefulness of the LCOM cohesion metric was positively evaluated in the prediction of faults (20.8% of the studies). This positive evaluation also occurred for other cohesion metrics: APIU was evaluated positively in 8.3% of the studies; CAM, CU, DCO, Fcoh, HC, ILCO, IPSC, NHD, Overlap, PF, SBFC, SCC, and Tightness, were also positively evaluated, each in 4.2% of the studies. The AMC metric was not evaluated, and the

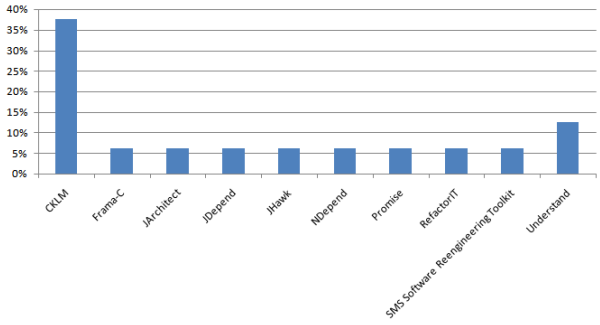

Figure 10: The tools most used to measure cohesion and coupling

other cohesion metrics (CC, Coh, Coverage, CPC, CR, LCOM3, Max-Coverage, NRC, PCM, Pcoh, SFC, and WFC) were not considered useful for fault prediction.

The CBO and RFC coupling metrics were positively evaluated for fault prediction (each metric in 37.5% of studies). This positive evaluation was also obtained for other coupling metrics. The Ce metric was evaluated positively in 12.5% of the studies. The MII, MPC, and NC metrics were also positively evaluated, each in 4.2% of the studies. The other coupling metrics (AC, BCFI, Ca, CBM, FOUT, and IC) were not considered useful for fault prediction.

Among the 41 cohesion and coupling metrics found in this 2nd SLM, the results obtained pointed to the LCOM cohesion metric and the CBO and RFC couplings as being the most analyzed in the studies and their usefulness in the prediction of faults was evaluated positively.

According to Chidamber and Kemerer (1994), LCOM or LCOM2 (Lack of Cohesion in Methods) is the number of pairs of methods in the class using no Attributes in common, minus the number of pairs of methods that do.

CBO (Coupling between Object Classes) for a class, according to Chidamber and Kemerer (1991) can be defined as a count of the number of other classes to which it is coupled. While RFC (Response for a Class) can be defined as set of methods that can be executed in response and messages received a message by the object of that class (CHIDAMBER; KEMERER, 1991).

The relationship between LCOM, CBO, and RFC is presented in the Chapter 5 where by analysis, it is clear that LCOM is positively correlated to CBO and RFC.

5 RELATIONSHIP BETWEEN COHESION AND COUPLING METRICS

Cohesion and coupling are regarded as fundamental features of the internal quality of object-oriented systems (OOS). Analyzing the relationships between cohesion and coupling metrics plays a significant role to develop efficient techniques for determining the external quality of an object-oriented system.

Researchers have proposed several metrics to find cohesion and coupling in object-oriented systems. However, few of them have proposed an analysis of the relationship between cohesion and coupling. This paper, empirically investigates the relationships among several cohesion and coupling metrics in object-oriented systems.

This chapter attempts to find mutual relationships between those metrics by statistically analyzing the results of experiments.

5.1 SELECTED SYSTEMS

We collected some data in (DALLAL, 2015) and (DALLAL, 2013) where a Java tool was developed by Al Dallal (DALLAL, 2015) and the Borland Together Tool was used to automate the extraction of cohesion and coupling metrics. The data was obtained in three Java open-source systems from different domains: Art of Illusion[2], JabRef[3], and FreeMind[4], all this being open source systems randomly selected from SourceForge[5].

The choice of the three systems was motivated by their application in some related works (DALLAL, 2013) (DALLAL, 2015), which facilitated the comparison of the results obtained.

Art of Illusion, JabRef, and FreeMind consist of 430, 306 and 363 concrete classes (not abstract classes or interfaces), respectively. The descriptive statistics for each of the selected cohesion and coupling metrics including the minimum, 25% quartile, mean, median, 75% quartile, maximum value, and standard deviation, adapted from (DALLAL, 2015) and (DALLAL, 2013), is shown Tables 12 and 13.

Table 12 presents the descriptive statistics of 14 cohesion metrics, namely: LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, CBMC, ICBMC, OL_n, PCCC, LSCC, CC, SCOM, TCC, and LCC. However,

[2] https://sourceforge.net/projects/aoi/

[3] https://sourceforge.net/projects/jabref/

[4] http://freemind.sourceforge.net/

[5] http://sourceforge.net

Table 12: Descriptive statistics for the cohesion metrics

N.	Metric	Min	Max	25%	Median	Mean	75%	Std. Dev
01	LCOM1	0	7875	1	10	45	64.00	301.73
02	LCOM2	0	7875	0	3	32	46.98	278.44
03	LCOM3	0	25	1	1	2	1.64	1.77
04	LCOM4	0	25	1	1	2	1.64	1.77
05	LCOM5	0	2	1	0.82	0.97	0.77	0.47
06	CBMC	0	1	0	0	1	0.286	0.439
07	ICBMC	0	1	0	0	1	0.274	0.438
08	OL_n	0	1	0	0	1	0.286	0.439
09	PCCC	0	1	0	0.019	1	0.375	0.463
10	LSCC	0	1	0.032	0.162	1	0.372	0.407
11	CC	0	1	0.072	0.223	1	0.407	0.390
12	SCOM	0	1	0.072	0.298	1	0.450	0.403
13	TCC	0	1	0.005	0.400	1	0.472	0.409
14	LCC	0	1	0.005	0.577	1	0.535	0.426

Table 13 presents the descriptive statistics of 8 coupling metrics, namely: CBO_IUB, CBO_U, CBO, RFC, MPC, DAC, DAC2, and OCMEC.

Table 13: Descriptive statistics for the coupling metrics

N.	Metric	Min	Max	25%	Median	Mean	75%	Std. Dev
01	CBO_IUB	0	287	0	1	2	4.446	16.195
02	CBO_U	0	58	1	2	5	3.980	4.797
03	CBO	0	298	2	4	8	8.426	17.658
04	RFC	0	413	7	12	31	23.489	29.264
05	MPC	0	1739	9	22	60	53.651	102.631
06	DAC	0	131	1	2	4	3.676	6.855
07	DAC2	0	22	1	2	3	2.440	2.849
08	OCMEC	0	22	2	3	5	3.601	3.50

The statistics presented in Tables 12 and 13 were obtained by the statistic package after the execution of the Quality Measuring Tool [6] and extended CKJM [7] that analyzed the source code of the systems (Art of Illusion, JabRef, and FreeMind) and automates the calculation of the corresponding values for each metrics.

5.2 CORRELATION ANALYSIS

The correlation analysis aims to find the relationships between cohesion and coupling metrics for object-oriented systems (OOS). The goal of this correlation analysis is to answer the following questions:

- Is there a correlation between the cohesion and coupling metrics?

[6] http://www.isc.ku.edu.kw/drjehad/research.htm
[7] http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/

- Which correlation exists between the cohesion and coupling metrics?

To answer the questions above, the non-parametric Spearman's correlation coefficient between the considered cohesion and coupling metrics was calculated.

The result of correlation between cohesion and coupling metrics for all three systems is illustrated in Table 14 and the statistically significant (p-value < 0.0001) results are underlined. They lead to the following observations (in concordance with (DALLAL, 2015) and (DALLAL, 2013)):

Table 14: Correlation analysis results

Metric	CBO_IUB	CBO_U	CBO	RFC	MPC	DAC	DAC2	OCMEC
LCOM1	0.33	0.36	0.44	0.52	0.53	0.43	0.44	0.60
LCOM2	0.27	0.23	0.28	0.34	0.34	0.29	0.31	0.36
LCOM3	0.07	0.06	0.08	0.07	0.06	-0.10	-0.08	0.03
LCOM4	0.07	0.06	0.08	0.06	0.05	-0.11	-0.09	0.02
LCOM5	0.13	0.10	0.11	0.13	0.13	0.08	0.07	0.02
CBMC	-0.141	-0.329	-0.309	-0.474	-0.486	-0.401	-0.407	-0.453
ICBMC	-0.141	-0.329	-0.309	-0.474	-0.485	-0.401	-0.407	-0.452
OLn	-0.141	-0.329	-0.309	-0.474	-0.486	-0.401	-0.407	-0.453
PCCC	-0.199	-0.353	-0.368	-0.552	-0.572	-0.372	-0.370	-0.445
LSCC	-0.287	-0.284	-0.330	-0.372	-0.362	-0.597	-0.599	-0.375
CC	-0.241	-0.265	-0.290	-0.334	-0.323	-0.559	-0.557	-0.345
SCOM	-0.282	-0.250	-0.298	-0.350	-0.339	-0.534	-0.544	-0.370
TCC	-0.106	-0.119	-0.134	-0.086	-0.081	-0.009	-0.016	-0.086
LCC	-0.068	-0.082	-0.083	-0.031	-0.025	0.065	0.053	-0.036

1. In CBMC, ICBMC, OL_n, PCCC, LSCC, CC, and SCOM, the coupling is always (100%) negatively correlated to cohesion. This observation is indicated by the negative signs of the results. Therefore, the results indicate that coupling and cohesion are negatively correlated.

2. Only in LCOM1 and LCOM2, the coupling is always (100%) positively correlated to cohesion. LCOM5 and coupling metrics are 87.5% positively correlated. LCOM3 and coupling metrics are 62% positively correlated and 25% negatively correlated. While LCOM4 and coupling metrics are 50% positively correlated and 25% negatively correlated.

3. The correlations between each of LSCC, CC, and SCOM, which are the similarity-based measures, and the coupling measures are higher than the correlations between each of TCC and LCC and the coupling measures. TCC and coupling metrics

are 75% negatively correlated. While LCC and coupling metrics are 37.5% negatively correlated and 12.5% positively correlated. This result is expected because the similarity-based measures quantify cohesion more precisely than the metrics that ignore the degree of cohesion between a pair of methods.

4. Among all pairs of the considered cohesion and coupling metrics, the LCOM1 and OCMEC, PCCC and MPC, PCCC and RFC, LSCC and DAC1, LSCC and DAC2, CC and DAC1, CC and DAC2, have higher correlation than other considered pairs of metrics. This gives a sign that those metrics indicate cohesion or coupling more precisely and accurately than any of the other cohesion or coupling metrics considered.

5. Among all (112) relationships between the pairs of cohesion (14 metrics) and coupling metrics (8 metrics) studied on this research, 61.61% (69 pairs) were negatively correlated, 29.46% (33 pairs) were positively correlated and only 8.93% (10 pairs) were not correlated significantly. Therefore, these results indicate that cohesion and coupling in object-oriented system are negatively correlated.

The current chapter was restrcted to the cohesion and coupling metrics that were much discussed in literature, and whose validity is not contested.

6 ANALYSIS AND INTERPRETATION

As shown in section 4.2.4, the cohesion metrics LCOM, APIU, CAM, CU, DCO, Fcoh, HC, ILCO, IPSC, NHD, Overlap, PF, SBFC, SCC, and Tightness, and coupling metrics CBO, RFC, Ce, MII, MPC, and NC were found useful in fault prediction. On the other hand, cohesion metrics CC, Coh, Coverage, CPC, CR, LCOM3, MaxCoverage, NRC, PCM, Pcoh, SFC, and WFC, and coupling metrics AC, BCFI, Ca, CBM, FOUT, and IC were not fould useful in fault prediction.

Cohesion metrics LCOM1 and LCOM2 were found positively correlated to the coupling metrics CBO_IUB, CBO_U, CBO, RFC, MPC, DAC, DAC2, and OCMEC; however, among those metrics, only LCOM2, CBO, RFC, and MPC were found useful for fault prediction. About the other metrics (LCOM1, CBO_IUB, CBO_U, DAC2, and OCMEC), nothing could be said about their utility in fault prediction, because they were not found in the studies analyzed in the second systematic literature mapping (SLM). However, it can be inferred that the correlated cohesion and coupling metrics are effective for fault prediction, that is, the cohesion and coupling metrics that are correlated are also useful for fault prediction.

Among the 213 cohesion and coupling metrics found in this work, only 40 were analyzed with respect to their utility for fault prediction, which means that at least 173 cohesion and coupling metrics were not analyzed. It is possible the other useful metrics exist.

Among the 40 analyzed metrics only 15 cohesion and 6 coupling (totaling 21 or 52.5% of the 40 metrics) were found useful for fault prediction.

There are different cohesion and coupling metrics that share the same acronym and other metrics with more than one acronym. This may prejudice the understanding of fresh researchers or professionals in the area. For example, LCOM is the same as LCOM2, LCOM3=NLCOM (Normalized version of LCOM), CF=COF, DAC'= DAC1=DAC2, CBO'=CBO1, $RFC = RFC\infty$, RFC'=RFC1=RFC-1, DCBO=RCBO, CC (Class Cohesion) $\neq$ CC (Component cohesion), SCC (Similar context cohesiveness) $\neq$ SCC (Similarity-based Class Cohesion), IC (Import Coupling) $\neq$ IC (inheritance-based Intermodule coupling index) $\neq$ IC (Inheritance coupling), TCC (Tight Class Cohesion) $\neq$ TCC (Tight Class Coupling), LCC (Loose Class Cohesion) $\neq$ LCC (Loose Class Coupling).

According to Etzkorn et al. (2004), it is confusing that the HYSS tool, the PATRicia tool, and the GEN++ metrics tool claim to calculate the same metric (LCOM2), but values from the differ-

ent tools for this metric for the same classes have different values. This information was obtained during the research of Etzkorn et al. (2004), that used three tools (HYSS, PATRicia e GEN++ metrics tool) that automated the calculation of the value of some cohesion metrics (e. g. LCOM2). Given the same input (source code from one of the classes) the tools returned different values for LCOM2 (output).

The divergence obtained from the output of some tools confirms the need for more research about the tools used to automatize the calculation of the cohesion and coupling metrics. It is necessary to identify the most reliable tools (which follow rigorously the definition of the metrics) in order to avoid the production of wrong conclusions in research based on tools that produce wrong values. In order to verify that it is sufficient to calculate each metrics by hand and then to run the tools for finally compare the values obtained in each case; that should allow to certify the reliability of each tool.

The most used tool (in the studies of the first SLM) were: 1st Understand (used in 9 studies), 2nd CKJM and Quality Measuring Tool (each used in 8 studies), and in 3rd Borland Together Tool (used in 7 studies). However, the Quality Measure Tool, despite occupying the second place, was only found in studies of its own creator (Professor Jehad Al Dallal), which appeared as one of the authors of 8 studies found relevant. That tool is available at the Internet, but its use demands permission by the author. This may be one of the motives that it is not used by third parties. Another motive could be the fact that additional effort is necessary to tune the source code in order to make the tool work in different environments.

On the other hand, the tools Understand (commercial, supports Java and C++), CKJM (free, supports Java), and Borland Together Tool (commercial, supports Java and C++) are very popular and used by different researchers. Among the three very popular tools, only CKJM is free and exclusive to obtain the metrics of cohesion (LCOM) and coupling (CBO, RFC, Ca) of systems developed in Java.This may be the reason that Java is the language most used in studies as well as the metrics LCOM, LCOM3, CAM, CBO, RFC, Ca, Ce, CBM, and IC (Inheritance coupling) be the most used too. This may be the reason why CKJM is the most used tool (1st place) in fault prediction studies (2nd SLM). However, CKJM has an extended version, called Extended CKJM to obtain the cohesion metrics LCOM, LCOM3, CAM, and coupling metrics CBO, RFC, Ca, Ce,

CBM, and IC, of systems developed in Java.

7 COMPARISON TO RESULTS OBTAINED IN RELATED WORKS

This chapter compares the results obtained in Chapters 3, 4, and 5 with the works related to the grouping of metrics (appearing in Chapter 3), application of the metrics for foreseeing faults (Chapter 4) and the relations among coupling and cohesion metrics (Chapter 5).

7.1 GROUPING COHESION AND COUPLING METRICS

Table 15 summarize the comparison of obtained results in grouping cohesion and coupling metrics (including tools) to related works.

Table 15: Related work in grouping cohesion and coupling metrics including tools

Studies	**Cohesion Metrics**	**# Cohesion Metrics**	**Coupling Metrics**	**# Coupling Metrics**	**Tools**	**# Tools**
ETZKORN *et al.*, 2004	LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, Coh, TCC, LCC, CBMC (without tool) and ICH (without tool)	10	—	—	HYSS tool, PATRicia, and GEN++ tool	3
AGGARWAL *et al.*, 2009	LCOM1, LCOM2, LCOM3, LCOM4, TCC, LCC, and ICH	7	CBO, CBO1, DAC, DAC1, MPC, ICP, IHICP, NIHICP, RFC, ACAIC, OCAIC, DCAEC, OCAEC, ACMIC, OCMIC, DCMEC, OCMEC, AMMIC, OMMIC, DMMEC, and OMMEC	21	—	—
RAGAB; AMMAR, 2010	LCOM1 (without tool), LCOM2 (without tool), Conn Comp and H	4	Change propagation probability, Dep_in, Dep_Out, EC_Attr, EC_Par, IC_Attr, IC_Par, MsgRecv (without tool), MsgSent (without tool), Size Of Change	10	SDMetrics, Sara Tool, ES2 (without metric), and SAAT (without metric)	4

GEETIKA; SINGH, 2014	—	—	EOC, IOC, OQFS, OPFS, DCBO, Degree of dynamic coupling between two classes at runtime, Degree of dynamic coupling within a given set of classes, RI, RE, RDI, RDE, DCM, CQFS, IC_OD, IC_OM, IC_OC, IC_CD, IC_CM, IC_CC, EC_OD, EC_OM, EC_OC, EC_CD, EC_CM, EC_CC, DFC, DCa, DKSC, DKCC, DKC, PAC, DOC, TDOC, and CDC	34	—	—
KAYARVIZHY, 2016	LCOM	1	Ca, Ce, CBO, CF, RFC	5	CKJM, Eclipse Metrics Plugin 1.3.6, Eclipse Metrics Plugin 3.4, Jdepend, JHawk, JMetrics, JMT, QMOOD++ (without metric) and SDMetrics	9
The present work	Presented in chapter 3	95	Presented in chapter 3	118	Presented in chapter 3	64

In the research of Etzkorn et al. (2004) 10 cohesion metrics and their respective tools (3 tools) for C++ language were identified, such as LCOM1 (HYSS tool), LCOM2 (HYSS tool, PATRicia (ETZKORN; DAVIS; LI, 1998) and GEN++ tool), LCOM3 (HYSS tool, PATRicia (ETZKORN; DAVIS; LI, 1998)), LCOM4 (HYSS tool), LCOM5 (HYSS tool), Coh (HYSS tool), TCC (HYSS tool), LCC (HYSS tool), CBMC (without tool) and ICH (without tool). The focus of their research was the comparison between the metrics of cohesion, which is different of the discovering of the metrics and tools existing in the literature, as is the case of the first SLM.

AGGARWAL *et al.* (2009) e 21 de acoplamento studied 7 cohesion and 21 coupling metrics. However, the do not analyzed the tools used to calculate them.

In another research (RAGAB; AMMAR, 2010), four cohesion metrics and 10 coupling metrics were surveyed, including their respective tools (4 tools) for Java and C++ languages, such as: Dep_in (SDMetrics), MsgRecv, EC_Attr (SDMetrics), EC_Par (SDMetrics), Dep_Out (SDMetrics), MsgSent, IC_Attr (SDMetrics), IC_Par (SDMetrics), Change Propagation Probability (Sara Tool), Size Of Change

(Sara Tool), LCOM1, LCOM2, H (SDMetrics), and Conn Comp (SDMetrics). They also described the tools ES2 and SAAT (MUSKENS; CHAUDRON; WESTGEEST, 2002) without presenting the metrics that can be obtained from them.

GEETIKA; SINGH (2014) studied 34 coupling metrics. However they did not analyzed the tools for calculating their values.

Kayarvizhy (2016) selected 9 tools and 6 metrics of cohesion and coupling: SDMetrics (Ca, Ce), JHawk (LCOM, CBO, RFC), QMOOD++ (BANSIYA, 1997), CKJM (LCOM, CBO, RFC, Ca), JMetrics (LCOM), JMT (CF, LCOM, CBO, RFC), JDepend (Ca, Ce), Eclipse Metrics Plugin 1.3.6 (LCOM, Ce, Ca), and Eclipse Metrics Plugin 3.4 (LCOM).

The current work obtained 95 cohesion and 118 coupling metrics, producing a group of 213 cohesion and coupling metrics, thus surpassing the amount of cohesion and coupling metrics presented in related works in the literature. Among the 213 grouped metrics, 65 cohesion metrics and 81 coupling metrics, totaling 146 metrics, have tools to automate the calculation of the corresponding value. However, for 30 cohesion and 37 coupling metrics, totaling 67 metrics, the respective tools were not found.

7.2 APPLICATION OF COHESION AND COUPLING METRICS IN FAULT PREDICTION

Table 16 summarize the comparison of obtained results in application of the cohesion and coupling metrics in the prediction of faults to related works.

Table 16: Related work in application of cohesion and coupling metrics in Fault Prediction

Studies	**Search Period**	**Tools**	**# Tools**	**The most used tools (tools that appeared in more publications)**
RADJENOVIĆ *et al.*, 2013	1991 - 2011	—	—	—
ISONG; EKABUA, 2015	1995 - 2012	Borland Together, Columbus, Java static analysis tool, M-System based on GEN++, Metric tool integrated with Rigi, Software System Markup Language, and WebMetrics	7	M-System based on GEN++ (1^{st}), and Software System Markup Language (2^{nd})
SANTOS *et al.*, 2016	2004 - 2013	—	—	—
MALHOTRA, 2015	—	—	—	—

NUÑEZ-VARELA *et al.*, 2017	2010 - 2015	Analizo, Borland Together, CCCC, CKJM, Jhawk, Metrics 1.3.6, Moose software analysis platform, Rational Software Analyzer, SDMetrics, src2srcml, and Understand	11	Understand (1st), and CKJM (2nd)
The present work	2014 - 2018	CKJM, Understand, Frama-C, JArchitect, JDepend, JHawk, NDepend, PROMISE, RefactorIT, and DMS Software Reengineering Toolkit	10	CKJM (1st), and Understand (1st)

Studies	**The most used language**	**The most used system**	**Useful Cohesion Metrics**	**Useful Coupling Metrics**
RADJENOVIĆ *et al.*, 2013	C++ (1st), Java (2nd), and C (3rd)	PROMISE repository (1st), and NASA (2nd)	—	—
ISONG; EKABUA, 2015	C++ (1st), and Java (2nd)	NASA (1st), and PROMISE (2nd)	LCOM	CBO and RFC
SANTOS *et al.*, 2016	Java (1st), and C++ (2nd)	—	—	—
MALHOTRA, 2015	—	NASA (1st), and PROMISE (2nd)	LCOM	CBO and RFC
NUÑEZ-VARELA *et al.*, 2017	Java (1st), AspectJ (2nd), and C++ (3rd)	PROMISE repository of University of Ottawa (1st), and PROMISE repository (2nd)	—	—
The present work	Java (1st), C++ (2nd), and C (3rd)	PROMISE (1st), SOURCE FORGE (2nd), Bug Prediction Dataset (3rd), and NASA (4th)	LCOM, APIU, CAM, CU, DCO, Fcoh, HC, ILCO, IPSC, NHD, Overlap, PF, SBFC, SCC, and Tightness	CBO, RFC, Ce, MII, MPC, and NC

Between January 2014 and April 2018, the most used tool to extract metrics from OOS, among the 10 found, was CKJM, followed by Understand. However, in (NUÑEZ-VARELA et al., 2017), an SRL with data from 2010 to 2015, among 11 tools found, Understand occupied the first place and CKJM the second. This difference shows that CKJM has become the most used tool in research in recent years.

According to the results of the second systematic literature mapping (SLM), from January 2014 to April 2018, the most used programming language in the collection of metrics and fault information was Java with 94.6% of projects using this language. However, according to (RADJENOVIĆ et al., 2013), from 1991 to 2011, the most commonly used languages were: C++ (35%), followed by Java (34%) and C (16%). According to (ISONG; EKABUA, 2015), from January 1995 to December 2012, about 54% of applications were developed in C++, and only 43% in Java.

The results obtained in (SANTOS et al., 2016), indicate that from 2004 to 2013, Java was the most used language (55%) fol-

lowed by C++ (21%), C (7%), C# (3%), Python (2%), PHP (1%), Perl (1%), and Ada (3%). Finally, according to (NUÑEZ-VARELA et al., 2017), between 2010 and 2015 the Java language was the most used (64.7%), followed by AspectJ (12.6%), C++ (10.1%), C (5%), C# (2.5%), Jak (1.7%), Ada (0.8%), Cobol (0.4%), Javascript(0.4%), Pharo (0.4%), PHP (0.4%), Python (0.4%), and Ruby (0.4%).

In the second SLM, the results point to PROMISE repository as being the most used system, followed by NASA, which confirms the results presented in (RADJENOVIĆ et al., 2013), but diverges from the results presented in (NUÑEZ-VARELA et al., 2017) which pointed to the PROMISE repository of University of Ottawa in first and PROMISE repository in second place. It also diverges from Malhotra (2015) and Isong and Ekabua (2015) that point to NASA in the first place and PROMISE in second.

This divergence reflects the selection criteria of the studies applied by each author, according to the objectives of the research, because (NUÑEZ-VARELA et al., 2017) is a systematic mapping study focusing on the source code of the metrics (not on the application of the metrics in the prediction of faults), (MALHOTRA, 2015) is an SRL focused on machine learning techniques for predicting software faults, and (ISONG; EKABUA, 2015) is an SRL that focused on empirical validation of metrics in the prediction of fault-prone software. All these studies were carried out at different times and with different objectives as well.

The results obtained in the second SLM point to LCOM as being the cohesion metric that was most positively evaluated in the prediction of failures (20.8% of the studies). This positive evaluation was also obtained for the following cohesion metrics: APIU evaluated positively in 8.3% of studies, CAM, CU, DCO, Fcoh, HC, ILCO, IPSC, NHD, Overlap, PF, SBFC, SCC, and Tightness, evaluated positively in 4.2% of studies each. The AMC metric has not been evaluated. Other cohesion metrics (CC, Coh, Coverage, CPC, CR, LCOM3, MaxCoverage, NRC, PCM, Pcoh, SFC, and WFC) were not found useful for fault prediction.

In addition to the cohesion metrics, the second SLM also points to the existence of coupling metrics, such as CBO and RFC that were positively evaluated in the prediction of faults (each metric in 37.5% of studies). This positive evaluation was also obtained for the following coupling metrics: Ce, positively evaluated in 12.5% of studies, MII, MPC, and NC, also positively evaluated in 4.2% of the studies each. Other coupling metrics (AC, BCFI, Ca, CBM, FOUT,

and IC) have not been found useful for fault prediction.

Thus, about the application of the cohesion and coupling metrics in the prediction of faults, the second SLM obtained results that agree with those obtained in (MALHOTRA, 2015) and (ISONG; EKABUA, 2015) concerning the application of the CBO, RFC, and LCOM metrics. This comparison was not possible with (RADJENOVIĆ et al., 2013), (NUÑEZ-VARELA et al., 2017), and (SANTOS et al., 2016) because these works did not present the application of any cohesion metrics or coupling in the prediction of faults. In addition to the three most studied metrics (CBO, RFC, and LCOM), the present work also addressed other metrics of cohesion and coupling that were proposed more recently, since those three were proposed in 1991 (CHIDAMBER; KEMERER, 1991). Thus, it is considered that there is a need to address the most recent metrics in research, even if there is a certain difficulty in obtaining tools to incorporate the new metrics into the studies.

7.3 RELATIONSHIP BETWEEN COHESION AND COUPLING METRICS

The results obtained (relationship between cohesion and coupling metrics) in this work are different from those obtained by Kabaili, Keller and Lustman (2001), which stated that there was no correlation between cohesion and coupling metrics.

On the other hand, the results of the present work are similar to (emphasizing) the results obtained by Badri, Badri and Gueye (2008), Rathore and Gupta (2012), Dallal (2013), and Dallal (2015), which stated that there was correlation between cohesion and coupling metrics.

In Kabaili, Keller and Lustman (2001), Badri, Badri and Gueye (2008), Rathore and Gupta (2012), Dallal (2013), and Dallal (2015), the authors analyzed the correlation between cohesion and coupling metrics, but conclusions were hard to generalize because small number of metrics were used.

This work analyzes fourteen cohesion metrics (LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, CBMC, ICBMC, OLn , PCCC, LSCC, CC, SCOM, TCC, and LCC) and eight coupling metrics (CBO_IUB, CBO_U, CBO, RFC, MPC, DAC, DAC2, and OCMEC), surpassing the quantity of metrics analyzed in related works, allowing obtaining results closer to reality.

8 LIMITATIONS

There are some threats that can affect the results obtained in this work. Although the present work has achieved a greater number (213) of cohesion and coupling metrics in relation to those found in literature, it is possible that there are some metrics of cohesion and coupling (including their tools) that were not included in the present research, but by the criteria applied in the first SLM, it is believed that the number of cohesion and coupling metrics that were not included is negligible (in percentage terms).

The metrics presented with each tool are based on the articles read in full (by one reader only) and not in the empirical test and verification, assuming as true the description of the articles studied about the relationship between metrics and tools. The conclusions presented in this study are based solely on the cohesion and coupling metrics found in the studies analyzed.

Among 213 cohesion and coupling metrics found in the first SLM, only 40, being 28 cohesion and 12 coupling, were found in the second SLM (application of those metrics in fault prediction). This is not to say that the other metrics are not useful in predicting faults, but that the comparison of useful metrics was only made between the metrics evaluated in the recent studies consulted.

Thus, in addition to the metrics presented in the present work, older metrics (not found in the present work) could also be good indicators in the prediction of faults. The conclusions presented in this study are based solely on the cohesion and coupling metrics found in the studies analyzed between 2014 and 2018.

9 CONCLUSION AND FUTURE WORK

We conducted two systematic literature mappings (SLM) with the main goal of grouping the cohesion and coupling metrics with the respective tools used to obtain the values of each metric, and to analyze the cohesion and coupling metrics applied to fault prediction.

In the analyzed studies, there were producing a set of 213 cohesion and coupling metrics, which surpassing the amount of cohesion and coupling metrics presented in related works existing in the Literature. Among this metrics, 68.5% have tools to automate the calculation of the corresponding value.

There are some cohesion metrics (LCOM2, LCOM3, LCOM5 and TCC) and coupling metrics (RFC and CBO) that were more intensively studied (found in many published articles) in relation to others.

LCOM2 is the cohesion metric that has been positively evaluated to predict faults in more studies. This positive evaluation was also obtained by APIU, CAM, CU, DCO, Fcoh, HC, ILCO, IPSC, NHD, Overlap, PF, SBFC, SCC, and Tightness. The cohesion metrics CC, Coh, Coverage, CPC, CR, LCOM3, MaxCoverage, NRC, PCM, Pcoh, SFC, and WFC were not considered useful for fault prediction.

Coupling metrics such as CBO and RFC were positively evaluated to predict faults in more studies. This positive evaluation was also obtained for the Ce, MII, MPC, and NC metrics. However, the other coupling metrics (AC, BCFI, Ca, CBM, FOUT, and IC) were not considered useful for fault prediction.

The obtained results of correlation between cohesion and coupling metrics showed that, there is in fact a negative correlation between cohesion and coupling for the studied systems. That is, if you increase the cohesion of a class (which is good), there is a greater likelihood of reducing the coupling (which is also good), and vice versa.

The cohesion metrics LCOM1 and LCOM2 were found positively correlated with the coupling metrics CBO_IUB, CBO_U, CBO, RFC, MPC, DAC, DAC2, and OCMEC, but among these metrics, only LCOM2, CBO, RFC, and MPC were found useful in predicting faults. However, it can be inferred that the relationship between cohesion and coupling takes effect in the prediction of fault, that is, the metrics of cohesion and coupling that are correlated, have the same behavior as their usefulness in the prediction of fault.

Results produced by the present work may contribute for academic research and for the industry practice by providing a general

view of the relations among cohesion and coupling metrics and fault prediction, and easing the identification of some metrics and their respective tools.

In the future it is intended to include the test and functionality analysis of each tool to reveal to what extent these tools are faithful in calculating the values of each cohesion and coupling metrics, taking into consideration the definition of these metrics.

In addition, it is intended to include in the study other characteristics of the internal quality of software, such as size and complexity, as these were also incorporated in some models of prediction of faults existing in literature. In addition to studies of SLM, it is intended to perform empirical studies for proposing a useful model of prediction of faults.

REFERENCES

ABDEEN, H.; DUCASSE, S.; SAHRAOUI, H. Modularization metrics: Assessing package organization in legacy large object-oriented software. In: IEEE. *Reverse Engineering (WCRE), 2011 18th Working Conference on*. [S.l.], 2011. p. 394–398.

ABDELMOEZ, W. M.; GOSEVA-POPSTOJANOVA, K.; AMMAR, H. H. Methodology for maintainability-based risk assessment. In: IEEE. *Reliability and Maintainability Symposium, 2006. RAMS'06. Annual*. [S.l.], 2006. p. 337–342.

ABREU, F. B. e. Design metrics for object-oriented software systems. In: *Workshop Quantitative Methods for Object-Oriented Systems Development, ECOOP*. [S.l.: s.n.], 1995. v. 95.

ABREU, F. B. e; MELO, W. Evaluating the impact of object-oriented design on software quality. In: IEEE. *Software Metrics Symposium, 1996., Proceedings of the 3rd International*. [S.l.], 1996. p. 90–99.

ABUASAD, A.; ALSMADI, I. M. The correlation between source code analysis change recommendations and software metrics. In: ACM. *Proceedings of the 3rd International Conference on Information and Communication Systems*. [S.l.], 2012. p. 2.

ADAM, A. *Implementation and Validation of OO Design-level Cohesion Metrics*. Tese (Doutorado) — MS Thesis, Information and Computer Science Department, King Fahd University of Petroleum and Minerals, 2004.

AGGARWAL, K. et al. Investigating effect of design metrics on fault proneness in object-oriented systems. *Journal of Object Technology*, v. 6, n. 10, p. 127–141, 2007.

AGGARWAL, K. et al. Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: a replicated case study. *Software process: Improvement and practice*, Wiley Online Library, v. 14, n. 1, p. 39–62, 2009.

AHMED, M. A.; ABUBAKAR, A.; ALGHAMDI, J. S. A study on the uncertainty inherent in class cohesion measurements. *Journal of Systems Architecture*, Elsevier, v. 57, n. 4, p. 474–484, 2011.

ALEXAN, N.; GAREM, R. E.; OTHMAN, H. An extendible open source tool measuring software metrics for indicating software quality. In: IEEE. *Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA), 2016*. [S.l.], 2016. p. 172–176.

ALGHAMDI, J.; ELISH, M.; AHMED, M. A tool for measuring inheritance coupling in object-oriented systems. *information SCiences*, Elsevier, v. 140, n. 3-4, p. 217–227, 2002.

ALLEN, E. B.; GOTTIPATI, S.; GOVINDARAJAN, R. Measuring size, complexity, and coupling of hypergraph abstractions of software: An information-theory approach. *Software Quality Journal*, Springer, v. 15, n. 2, p. 179–212, 2007.

ALLEN, E. B.; KHOSHGOFTAAR, T. M.; CHEN, Y. Measuring coupling and cohesion of software modules: an information-theory approach. In: IEEE. *Software Metrics Symposium, 2001. METRICS 2001. Proceedings. Seventh International*. [S.l.], 2001. p. 124–134.

ALMUGRIN, S.; ALBATTAH, W.; MELTON, A. Using indirect coupling metrics to predict package maintainability and testability. *Journal of Systems and Software*, Elsevier, v. 121, p. 298–310, 2016.

ALSHAMMARY, T. F.; ALENEZI, M. Software architecture understandability in object-oriented systems. *i-Manager's Journal on Software Engineering*, iManager Publications, v. 12, n. 2, p. 1, 2017.

AMAN, H. et al. A proposal of class cohesion metrics using sizes of cohesive parts. In: *Proc. of Fifth Joint Conference on Knowledge-based Software Engineering*. [S.l.: s.n.], 2002. p. 102–107.

AMARA, D.; RABAI, L. B. A. Towards a new framework of software reliability measurement based on software metrics. *Procedia Computer Science*, Elsevier, v. 109, p. 725–730, 2017.

ANABALON, D. et al. Controlling complexity of web services interfaces through a metrics-driven approach. In: IEEE. *Computing Networking and Informatics (ICCNI), 2017 International Conference on*. [S.l.], 2017. p. 1–9.

ANJOS, E. G. dos; GOMES, R. D.; ZENHA-RELA, M. Assessing maintainability metrics in software architectures using cosmic and uml. In: SPRINGER. *International Conference on Computational Science and Its Applications*. [S.l.], 2012. p. 132–146.

ANWER, S. et al. Effect of coupling on software faults: An empirical study. In: IEEE. *Communication, Computing and Digital Systems (C-CODE), International Conference on*. [S.l.], 2017. p. 211–215.

APIWATTANAPONG, T.; ORSO, A.; HARROLD, M. J. Efficient and precise dynamic impact analysis using execute-after sequences. In: ACM. *Proceedings of the 27th international conference on Software engineering*. [S.l.], 2005. p. 432–441.

ARAR, Ö. F.; AYAN, K. Deriving thresholds of software metrics to predict faults on open source software: Replicated case studies. *Expert Systems with Applications*, Elsevier, v. 61, p. 106–121, 2016.

ARISHOLM, E. Dynamic coupling measures for object-oriented software. In: *Proceedings of the 8th International Symposium on Software Metrics*. Washington, DC, USA: IEEE Computer Society, 2002. (METRICS '02), p. 33–. ISBN 0-7695-1339-5. Available from Internet: <http://dl.acm.org/citation.cfm?id=823457.824024>.

ARISHOLM, E.; BRIAND, L. C.; FOYEN, A. Dynamic coupling measurement for object-oriented software. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 30, n. 8, p. 491–506, 2004. ISSN 0098-5589. Available from Internet: <http://dx.doi.org/10.1109/TSE.2004.41>.

ARVANITOU, E.-M. et al. Software metrics fluctuation: a property for assisting the metric selection process. *Information and Software Technology*, Elsevier, v. 72, p. 110–124, 2016.

BADRI, L.; BADRI, M. A proposal of a new class cohesion criterion: an empirical study. *Journal of Object Technology*, v. 3, n. 4, p. 145–159, 2004.

BADRI, L.; BADRI, M.; GUEYE, A. B. Revisiting class cohesion: An empirical investigation on several systems. *Journal of Object technology*, ETH Zuric ©JOT, v. 7, n. 6, p. 55–75, 2008.

BADRI, L.; BADRI, M.; TOURE, F. Exploring empirically the relationship between lack of cohesion and testability in object-oriented systems. In: SPRINGER. *International Conference on Advanced Software Engineering and Its Applications*. [S.l.], 2010. p. 78–92.

BADRI, L.; BADRI, M.; TOURE, F. An empirical analysis of lack of cohesion metrics for predicting testability of classes. *International Journal of Software Engineering and Its Applications*, v. 5, n. 2, p. 69–85, 2011.

BAKAR, N. S. A. A.; BOUGHTON, C. V. Validation of measurement tools to extract metrics from open source projects. In: IEEE. *Open Systems (ICOS), 2012 IEEE Conference on*. [S.l.], 2012. p. 1–6.

BANSIYA, J. *A hierarchical model for quality assessment of object-oriented designs*. [S.l.]: The University of Alabama in Huntsville, 1997.

BANSIYA, J.; DAVIS, C.; ETZKORN, L. An entropy-based complexity measure for object-oriented designs. *Theory and practice of object systems*, Wiley Online Library, v. 5, n. 2, p. 111–118, 1999.

BANSIYA, J.; DAVIS, C. G. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on software engineering*, IEEE, v. 28, n. 1, p. 4–17, 2002.

BASILI, V. R.; BRIAND, L. C.; MELO, W. L. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on software engineering*, IEEE, v. 22, n. 10, p. 751–761, 1996.

BASSEY, I. Enhancing software maintenance via early prediction of fault-prone object-oriented classes. *International Journal of Software Engineering and Knowledge Engineering*, World Scientific, v. 27, n. 04, p. 515–537, 2017.

BATISTA, E. B.; SANT'ANNA, C.; SILVA, B. C. da. Two quasi-experiments on cohesion metrics and program comprehension. In: ACM. *Proceedings of the 11th Brazilian Symposium on Software Components, Architectures, and Reuse*. [S.l.], 2017. p. 2.

BAVOTA, G.; LUCIA, A. D.; OLIVETO, R. Identifying extract class refactoring opportunities using structural and semantic cohesion measures. *Journal of Systems and Software*, Elsevier, v. 84, n. 3, p. 397–414, 2011.

BELLIN, D.; TYAGI, M.; TYLER, M. Object-oriented metrics: an overview. In: IBM PRESS. *Proceedings of the 1994 conference of the Centre for Advanced Studies on Collaborative research*. [S.l.], 1994. p. 4.

BIEMAN, J. M.; KANG, B.-K. Cohesion and reuse in an object-oriented system. In: ACM. *ACM SIGSOFT Software Engineering Notes*. [S.l.], 1995. v. 20, n. SI, p. 259–262.

BIEMAN, J. M.; OTT, L. M. Measuring functional cohesion. *IEEE transactions on Software Engineering*, IEEE, v. 20, n. 8, p. 644–657, 1994.

BONJA, C.; KIDANMARIAM, E. Metrics for class cohesion and similarity between methods. In: ACM. *Proceedings of the 44th annual Southeast regional conference*. [S.l.], 2006. p. 91–95.

BOSHNAKOSKA, D.; MIŠEV, A. Correlation between object-oriented metrics and refactoring. In: SPRINGER. *International Conference on ICT Innovations*. [S.l.], 2010. p. 226–235.

BOUCHER, A.; BADRI, M. Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison. *Information and Software Technology*, Elsevier, 2017.

BOWES, D.; HALL, T.; KERR, A. Program slicing-based cohesion measurement: the challenges of replicating studies using metrics. In: ACM. *Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics*. [S.l.], 2011. p. 75–80.

BOWMAN, M.; BRIAND, L. C.; LABICHE, Y. Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms. *IEEE Transactions on Software Engineering*, IEEE, v. 36, n. 6, p. 817–837, 2010.

BRÄUER, J. et al. Measuring object-oriented design principles: The results of focus group-based research. *Journal of Systems and Software*, Elsevier, v. 140, p. 74–90, 2018.

BRIAND, L.; DEVANBU, P.; MELO, W. An investigation into coupling measures for c++. In: ACM. *Proceedings of the 19th international conference on Software engineering*. [S.l.], 1997. p. 412–421.

BRIAND, L.; MORASCA, S.; BASILI, V. R. Defining and validating high-level design metrics. Citeseer, 1994.

BRIAND, L. C.; DALY, J. W.; WUST, J. K. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on software Engineering*, IEEE, v. 25, n. 1, p. 91–121, 1999.

BRIAND, L. C.; WUST, J.; LOUNIS, H. Using coupling measurement for impact analysis in object-oriented systems. In: IEEE. *Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on*. [S.l.], 1999. p. 475–482.

BRIAND, L. C.; WÜST, J.; LOUNIS, H. Replicated case studies for investigating quality factors in object-oriented designs. *Empirical software engineering*, Springer, v. 6, n. 1, p. 11–58, 2001.

BUDHKAR, S. A.; GOPAL, A. Component evaluation and component interface identification from object oriented system. *International Journal of Advanced Research in Computer Science*, International Journal of Advanced Research in Computer Science, v. 3, n. 4, 2012.

CATAL, C.; DIRI, B. Software fault prediction with object-oriented metrics based artificial immune recognition system. In: SPRINGER. *International Conference on Product Focused Software Process Improvement*. [S.l.], 2007. p. 300–314.

CHAE, H. S.; KWON, Y. R.; BAE, D.-H. A cohesion measure for object-oriented classes. *Software-Practice and Experience*, London, New York, Wiley Interscience [etc.], v. 30, n. 12, p. 1405–1432, 2000.

CHAHAL, K.; SINGH, H. Analyzing software evolution using the mood. *International Journal of Advanced Research in Computer Science*, International Journal of Advanced Research in Computer Science, v. 2, n. 1, 2011.

CHHABRA, J. K.; GUPTA, V. A survey of dynamic software metrics. *Journal of computer science and technology*, Springer, v. 25, n. 5, p. 1016–1029, 2010.

CHHILLAR, R. S.; GAHLOT, S. An evolution of software metrics: A review. In: ACM. *Proceedings of the International Conference on Advances in Image Processing*. [S.l.], 2017. p. 139–143.

CHIDAMBER, S. R.; KEMERER, C. F. *Towards a metrics suite for object oriented design*. [S.l.]: ACM, 1991.

CHIDAMBER, S. R.; KEMERER, C. F. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, IEEE, v. 20, n. 6, p. 476–493, 1994.

CHOWDHURY, I.; ZULKERNINE, M. Can complexity, coupling, and cohesion metrics be used as early indicators of vulnerabilities? In: ACM. *Proceedings of the 2010 ACM Symposium on Applied Computing*. [S.l.], 2010. p. 1963–1969.

CHOWDHURY, I.; ZULKERNINE, M. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. *Journal of Systems Architecture*, Elsevier, v. 57, n. 3, p. 294–313, 2011.

CONCAS, G. et al. Study of the evolution of an agile project featuring a web application using software metrics. In: SPRINGER. *International Conference on Product Focused Software Process Improvement*. [S.l.], 2008. p. 386–399.

CONCAS, G. et al. Assessing traditional and new metrics for object-oriented systems. In: ACM. *Proceedings of the 2010 ICSE*

Workshop on Emerging Trends in Software Metrics. [S.l.], 2010. p. 24–31.

CONTRERAS, F. et al. Balancing flexibility and performance in three dimensional meshing tools. *Advances in Engineering Software*, Elsevier, v. 41, n. 3, p. 471–479, 2010.

COSCIA, J. L. O. et al. Predicting web service maintainability via object-oriented metrics: a statistics-based approach. In: SPRINGER. *International Conference on Computational Science and Its Applications*. [S.l.], 2012. p. 29–39.

COUNSELL, S. et al. *Evaluation of an object-oriented cohesion metric through Hamming distances*. [S.l.], 2002.

COUNSELL, S.; SWIFT, S.; CRAMPTON, J. The interpretation and utility of three cohesion metrics for object-oriented design. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM, v. 15, n. 2, p. 123–149, 2006.

DALLAL, J. A. A design-based cohesion metric for object-oriented classes. *International Journal of Computer Science and Engineering*, Citeseer, v. 1, n. 3, p. 195–200, 2007.

DALLAL, J. A. Software similarity-based functional cohesion metric. *IET software*, IET, v. 3, n. 1, p. 46–57, 2009.

DALLAL, J. A. Improving the applicability of object-oriented class cohesion metrics. *Information and Software Technology*, Elsevier, v. 53, n. 9, p. 914–928, 2011.

DALLAL, J. A. Measuring the discriminative power of object-oriented class cohesion metrics. *IEEE Transactions on Software Engineering*, IEEE, v. 37, n. 6, p. 788–804, 2011.

DALLAL, J. A. Transitive-based object-oriented lack-of-cohesion metric. *Procedia computer science*, Elsevier, v. 3, p. 1581–1587, 2011.

DALLAL, J. A. Fault prediction and the discriminative powers of connectivity-based object-oriented class cohesion metrics. *Information and Software Technology*, Elsevier, v. 54, n. 4, p. 396–416, 2012.

DALLAL, J. A. The impact of accounting for special methods in the measurement of object-oriented class cohesion on refactoring and fault prediction activities. *Journal of Systems and Software*, Elsevier, v. 85, n. 5, p. 1042–1057, 2012.

DALLAL, J. A. Theoretical analysis for the impact of including special methods in lack-of-cohesion computation. *Procedia Technology*, Elsevier, v. 1, p. 167–171, 2012.

DALLAL, J. A. Empirical analysis of the relation between object-oriented class lack-of-cohesion and coupling. *AWERProcedia Information Technology & Computer Science*, Citeseer, v. 4, p. 34–39, 2013.

DALLAL, J. A. Empirical exploration for the correlation between class object-oriented connectivity-based cohesion and coupling. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, scholar.waset.org/1999.4/10000969, v. 9, n. 4, p. 934–937, 2015.

DALLAL, J. A. Predicting fault-proneness of reused object-oriented classes in software post-releases. *Arabian Journal for Science and Engineering*, Springer, p. 1–14, 2017.

DALLAL, J. A.; BRIAND, L. C. An object-oriented high-level design-based class cohesion metric. *Information and software technology*, Elsevier, v. 52, n. 12, p. 1346–1361, 2010.

DALLAL, J. A.; BRIAND, L. C. A precise method-method interaction-based cohesion metric for object-oriented classes. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM, v. 21, n. 2, p. 8, 2012.

DALLAL, J. A.; MORASCA, S. Predicting object-oriented class reuse-proneness using internal quality attributes. *Empirical Software Engineering*, Springer, v. 19, n. 4, p. 775–821, 2014.

DALLAL, J. A.; MORASCA, S. Investigating the impact of fault data completeness over time on predicting class fault-proneness. *Information and Software Technology*, Elsevier, 2018.

DEMARTINI, C.; IOSIF, R.; SISTO, R. A deadlock detection tool for concurrent java programs. *Software: Practice and Experience*, Wiley Online Library, v. 29, n. 7, p. 577–603, 1999.

DESOUKY, A. F.; ETZKORN, L. H. Object oriented cohesion metrics: a qualitative empirical analysis of runtime behavior. In: ACM. *Proceedings of the 2014 ACM Southeast Regional Conference*. [S.l.], 2014. p. 58.

DURISIC, D. et al. Measuring the impact of changes to the complexity and coupling properties of automotive software systems. *Journal of Systems and Software*, Elsevier, v. 86, n. 5, p. 1275–1293, 2013.

EDELMAN, A.; FRAKES, W.; LILLIE, C. Sam: Simple api for object-oriented code metrics. In: SPRINGER. *International Conference on Software Reuse*. [S.l.], 2008. p. 347–359.

ELISH, M. O.; AL-YAFEI, A. H.; AL-MULHEM, M. Empirical comparison of three metrics suites for fault prediction in packages of object-oriented systems: A case study of eclipse. *Advances in Engineering Software*, Elsevier, v. 42, n. 10, p. 852–859, 2011.

EMAM, K. E. et al. The confounding effect of class size on the validity of object-oriented metrics. *IEEE Transactions on Software Engineering*, IEEE, v. 27, n. 7, p. 630–650, 2001.

EMAM, K. E.; MELO, W.; MACHADO, J. C. The prediction of faulty classes using object-oriented design metrics. *Journal of Systems and Software*, Elsevier, v. 56, n. 1, p. 63–75, 2001.

ENGLISH, M.; CAHILL, T.; BUCKLEY, J. Construct specific coupling measurement for c++ software. *Computer Languages, Systems & Structures*, Elsevier, v. 38, n. 4, p. 300–319, 2012.

ENGLISH, M. et al. Fault detection and prediction in an open-source software project. In: ACM. *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*. [S.l.], 2009. p. 17.

ETZKORN, L.; DAVIS, C.; LI, W. A practical look at the lack of cohesion in methods metric. In: CITESEER. *Journal of Object-Oriented Programming*. [S.l.], 1998.

ETZKORN, L.; DELUGACH, H. Towards a semantic metrics suite for object-oriented design. In: IEEE. *Technology of Object-Oriented Languages and Systems, 2000. TOOLS 34. Proceedings. 34th International Conference on*. [S.l.], 2000. p. 71–80.

ETZKORN, L. H. et al. A comparison of cohesion metrics for object-oriented systems. *Information and Software Technology*, Elsevier, v. 46, n. 10, p. 677–687, 2004.

FENTON, N.; BIEMAN, J. *Software metrics: a rigorous and practical approach*. [S.l.]: CRC Press, 2014.

FERNÁNDEZ, L.; PEÑA, R. A sensitive metric of class cohesion. Institute of Information Theories and Applications FOI ITHEA, 2006.

FERREIRA, K. A. et al. Identifying thresholds for object-oriented software metrics. *Journal of Systems and Software*, Elsevier, v. 85, n. 2, p. 244–257, 2012.

FOWLER, M. et al. *Refactoring: improving the design of existing code*. [S.l.]: Addison-Wesley Professional, 1999.

GEETIKA, R.; SINGH, P. Dynamic coupling metrics for object oriented software systems: a survey. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 39, n. 2, p. 1–8, 2014.

GENERO, M.; PIATTINI, M.; CALERO, C. Empirical validation of class diagram metrics. In: IEEE. *Empirical Software Engineering, 2002. Proceedings. 2002 International Symposium n*. [S.l.], 2002. p. 195–203.

GIRGIS, M. R.; MAHMOUD, T. M.; NOUR, R. R. Uml class diagram metrics tool. In: IEEE. *International Conference on Computer Engineering & Systems, ICCES*. [S.l.], 2009. p. 423–428.

GOEL, B. M.; BHATIA, P. K. Analysis of reusability of object-oriented systems using object-oriented metrics. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 38, n. 4, p. 1–5, 2013.

GOEL, B. M.; GUPTA, S. B. Dynamic coupling based performance analysis of object oriented systems. *International Journal of Advanced Research in Computer Science*, v. 8, n. 5, 2017.

GORMAN, J. Oo design principles & metrics. *Online verfügbar unter http://www. parlezuml. com/metrics/OO% 20Design% 20Principles% 20&% 20Metrics. pdf, zuletzt geprüft am*, v. 15, p. 2009, 2006.

GOYAL, R.; CHANDRA, P.; SINGH, Y. Fuzzy inferencing to identify degree of interaction in the development of fault prediction models. *Journal of King Saud University-Computer and Information Sciences*, Elsevier, v. 29, n. 1, p. 93–102, 2017.

GUPTA, D. L.; SAXENA, K. Software bug prediction using object-oriented metrics. *Sādhanā*, Springer, v. 42, n. 5, p. 655–669, 2017.

GUPTA, V. Validation of dynamic coupling metrics for object-oriented software. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 36, n. 5, p. 1–3, 2011.

GUPTA, V.; CHHABRA, J. K. Object level run-time cohesion measurement. *INFORMATION THEORIES & APPLICATIONS*, p. 136, 1993.

GUPTA, V.; CHHABRA, J. K. Object level run-time cohesion measurement. *INFORMATION THEORIES & APPLICATIONS*, v. 16, p. 136, 2009.

GUPTA, V.; CHHABRA, J. K. Dynamic cohesion measures for object-oriented software. *Journal of Systems Architecture*, Elsevier, v. 57, n. 4, p. 452–462, 2011.

GUPTA, V.; CHHABRA, J. K. Package level cohesion measurement in object-oriented software. *Journal of the Brazilian Computer Society*, Springer, v. 18, n. 3, p. 251–266, 2012.

GYIMOTHY, T.; FERENC, R.; SIKET, I. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software engineering*, IEEE, v. 31, n. 10, p. 897–910, 2005.

HALL, G. A.; TAO, W.; MUNSON, J. C. Measurement and validation of module coupling attributes. *Software Quality Journal*, Springer, v. 13, n. 3, p. 281–296, 2005.

HASSOUN, Y.; COUNSELL, S.; JOHNSON, R. Dynamic coupling metric: proof of concept. *IEE Proceedings-Software*, IET, v. 152, n. 6, p. 273–279, 2005.

HASSOUN, Y.; JOHNSON, R.; COUNSELL, S. A dynamic runtime coupling metric for meta-level architectures. In: IEEE. *Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on*. [S.l.], 2004. p. 339–346.

HASSOUN, Y.; JOHNSON, R.; COUNSELL, S. Emprical validation of a dynamic coupling metric. *Birkbeck College London, Technical Report BBKCS-04-03*, 2004.

HENDERSON-SELLERS, B. *Software metrics*. [S.l.]: Prentice Hall, Hemel Hempstaed, UK, 1996.

HIGO, Y. et al. A pluggable tool for measuring software metrics from source code. In: IEEE. *Software Measurement, 2011 Joint Conference of the 21st Int'l Workshop on and 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA)*. [S.l.], 2011. p. 3–12.

HITZ, M.; MONTAZERI, B. Measuring coupling and cohesion in object-oriented systems. na, 1995.

HUSSAIN, S. et al. Detection of fault-prone classes using logistic regression based object-oriented metrics thresholds. In: IEEE. *Software Quality, Reliability and Security Companion (QRS-C), 2016 IEEE International Conference on*. [S.l.], 2016. p. 93–100.

HWA, J.; LEE, S.; KWON, Y. R. Hierarchical understandability assessment model for large-scale oo system. In: IEEE. *Software Engineering Conference, 2009. APSEC'09. Asia-Pacific*. [S.l.], 2009. p. 11–18.

IBRAHIM, S. M. et al. Identification of nominated classes for software refactoring using object-oriented cohesion metrics. *International Journal of Computer Science Issues (IJCSI)*, v. 9, n. 2, p. 68–76, 2012.

ILYAS, M. et al. Sourceviz: A tool for supporting software metrics visualization. *International Journal of Information Engineering and Electronic Business*, Modern Education and Computer Science Press, v. 9, n. 3, p. 18, 2017.

ISONG, B.; EKABUA, O. State-of-the-art in empirical validation of software metrics for fault proneness prediction: Systematic review. *International Journal of Computer Science Engineering Survey (IJCSES)*, v. 6, n. 6, 2015.

ISONG, B.; IFEOMA, O.; MBODILA, M. Supplementing object-oriented software change impact analysis with fault-proneness prediction. In: IEEE. *Computer and Information Science (ICIS), 2016 IEEE/ACIS 15th International Conference on*. [S.l.], 2016. p. 1–8.

ISONG, B.; OBETEN, E. A systematic review of the empirical validation of object-oriented metrics towards fault-proneness prediction. *International Journal of Software Engineering and Knowledge Engineering*, World Scientific, v. 23, n. 10, p. 1513–1540, 2013.

JARALLAH, A.; WASIQ, M.; AHMED, M. Principle and metrics for cohesion-based object-oriented component assessment. *Confidential Draft Copy*, 2001.

JAVED, Y.; ALENEZI, M. Defectiveness evolution in open source software systems. *Procedia Computer Science*, Elsevier, v. 82, p. 107–114, 2016.

JIN, C.; JIN, S.-W. Applications of fuzzy integrals for predicting software fault-prone. *Journal of Intelligent & Fuzzy Systems*, IOS Press, v. 26, n. 2, p. 721–729, 2014.

JOHARI, K.; KAUR, A. Effect of software evolution on software metrics: an open source case study. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 36, n. 5, p. 1–8, 2011.

JOHARI, K.; KAUR, A. Validation of object oriented metrics using open source software system: an empirical study. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 37, n. 1, p. 1–4, 2012.

KABAILI, H.; KELLER, R.; LUSTMAN, F. Class cohesion as predictor of changeability: An empirical study. Citeseer, 2001.

KAKARONTZAS, G. et al. Layer assessment of object-oriented software: A metric facilitating white-box reuse. *Journal of Systems and Software*, Elsevier, v. 86, n. 2, p. 349–366, 2013.

KANMANI, S.; PRATHIBHA, V. *Object-Oriented Metric Calculator*. [S.l.]: Technical Report, Pondicherry Engineering College, 2001.

KANMANI, S. et al. Investigation into the exploitation of object-oriented features. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 29, n. 2, p. 7–7, 2004.

KANMANI, S. et al. Measuring the object-oriented properties in small sized c++ programs–an empirical investigation. In: SPRINGER. *International Conference on Product Focused Software Process Improvement*. [S.l.], 2004. p. 185–202.

KANMANI, S. et al. Object-oriented software fault prediction using neural networks. *Information and software technology*, Elsevier, v. 49, n. 5, p. 483–492, 2007.

KAUR, A.; KAUR, I. Empirical evaluation of machine learning algorithms for fault prediction. *Lecture Notes on Software Engineering*, IACSIT Press, v. 2, n. 2, p. 176, 2014.

KAUR, A.; KAUR, I. An empirical evaluation of classification algorithms for fault prediction in open source projects. *Journal of King Saud University-Computer and Information Sciences*, Elsevier, 2018.

KAUR, A.; KAUR, K. Statistical comparison of modelling methods for software maintainability prediction. *International Journal of Software Engineering and Knowledge Engineering*, World Scientific, v. 23, n. 06, p. 743–774, 2013.

KAUR, H.; SINGH, S. Analysis of ck metrics thresholds to predict faults using log transformation. *Analysis*, v. 4, n. 3, 2015.

KAUR, K.; SINGH, H. Exploring design level class cohesion metrics. *Journal of Software Engineering and Applications*, Scientific Research Publishing, v. 3, n. 04, p. 384, 2010.

KAUR, N.; PAUL, A. Fault prediction modeling using object-oriented metrics: An empirical study. 2014.

KAUR, S.; MAINI, R. Analysis of various software metrics used to detect bad smells. *Int J Eng Sci (IJES)*, v. 5, n. 6, p. 14–20, 2016.

KAYARVIZHY, N. Systematic review of object oriented metric tools. *International Journal of Computer Applications*, Citeseer, v. 135, n. 2, 2016.

KEONG, C. K. et al. Toward using software metrics as indicator to measure power consumption of mobile application: A case study. In: IEEE. *Software Engineering Conference (MySEC), 2015 9th Malaysian*. [S.l.], 2015. p. 172–177.

KHAN, Y. A.; ELISH, M. O.; EL-ATTAR, M. A systematic review on the impact of ck metrics on the functional correctness of object-oriented classes. In: SPRINGER. *International Conference on Computational Science and Its Applications*. [S.l.], 2012. p. 258–273.

KHARI, M.; KUMAR, P. Evolutionary computation-based techniques over multiple data sets: An empirical assessment. *Arabian Journal for Science and Engineering*, Springer, p. 1–11, 2017.

KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele University*, v. 33, n. 2004, p. 1–26, 2004.

KUMAR, L.; KRISHNA, A.; RATH, S. K. The impact of feature selection on maintainability prediction of service-oriented applications. *Service Oriented Computing and Applications*, Springer, v. 11, n. 2, p. 137–161, 2017.

KUMAR, L.; MISRA, S.; RATH, S. K. An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes. *Computer Standards & Interfaces*, Elsevier, v. 53, p. 1–32, 2017.

KUMAR, L.; RATH, S. K. Hybrid functional link artificial neural network approach for predicting maintainability of object-oriented software. *Journal of Systems and Software*, Elsevier, v. 121, p. 170–190, 2016.

KUMAR, L. et al. Effective fault prediction model developed using least square support vector machine (lssvm). *Journal of Systems and Software*, Elsevier, 2017.

KUMARI, D.; RAJNISH, K. Investigating the effect of object-oriented metrics on fault proneness using empirical analysis. *International Journal of Software Engineering and Its Applications*, v. 9, n. 2, p. 171–188, 2015.

KUMARI, D.; RAJNISH, K. A new approach to find predictor of software fault using association rule mining. *Int. J. Eng. Technol*, v. 7, n. 5, p. 1671–1684, 2015.

KUMARI, U.; BHASIN, S. Application of object-oriented metrics to c++ and java: A comparative study. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 36, n. 2, p. 1–10, 2011.

LAMMA, E.; MELLO, P.; RIGUZZI, F. A system for measuring function points from an er–dfd specification. *The Computer Journal*, Oxford University Press, v. 47, n. 3, p. 358–372, 2004.

LEE, S.-Y.; LI, D.; LI, Y. An investigation of essential topics on software fault-proneness prediction. In: IEEE. *System and Software Reliability (ISSSR), International Symposium on*. [S.l.], 2016. p. 37–46.

LEE, Y. et al. Measuring the coupling and cohesion of an object-oriented program based on information flow. In: *Proc. International Conference on Software Quality, Maribor, Slovenia*. [S.l.: s.n.], 1995. p. 81–90.

LI, W. Another metric suite for object-oriented programming. *Journal of Systems and Software*, Elsevier, v. 44, n. 2, p. 155–162, 1998.

LI, W.; HENRY, S. Maintenance metrics for the object oriented paradigm. In: IEEE. *Software Metrics Symposium, 1993. Proceedings., First International*. [S.l.], 1993. p. 52–60.

LI, W.; HENRY, S. Object-oriented metrics that predict maintainability. *Journal of systems and software*, Elsevier, v. 23, n. 2, p. 111–122, 1993.

LINCKE, R.; LUNDBERG, J.; LÖWE, W. Comparing software metrics tools. In: ACM. *Proceedings of the 2008 international symposium on Software testing and analysis*. [S.l.], 2008. p. 131–142.

LORENZ, M.; KIDD, J. *Object-oriented software metrics*. [S.l.]: Prentice Hall Englewood Cliffs, 1994.

LU, H. et al. The ability of object-oriented metrics to predict change-proneness: a meta-analysis. *Empirical software engineering*, Springer, v. 17, n. 3, p. 200–242, 2012.

MA, Y.-T. et al. A hybrid set of complexity metrics for large-scale object-oriented software systems. *Journal of Computer Science and Technology*, Springer, v. 25, n. 6, p. 1184–1201, 2010.

MÄKELÄ, S.; LEPPÄNEN, V. Client-based cohesion metrics for java programs. *Science of Computer Programming*, Elsevier, v. 74, n. 5-6, p. 355–378, 2009.

MALHOTRA, R. A systematic review of machine learning techniques for software fault prediction. *Applied Soft Computing*, Elsevier, v. 27, p. 504–518, 2015.

MALHOTRA, R.; BANSAL, A. Investigation of various data analysis techniques to identify change prone parts of an open source software. *International Journal of System Assurance Engineering and Management*, Springer, v. 9, n. 2, p. 401–426, 2018.

MALHOTRA, R.; KAUR, A.; SINGH, Y. Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines. *International Journal of System Assurance Engineering and Management*, Springer, v. 1, n. 3, p. 269–281, 2010.

MALHOTRA, R.; KHANNA, M. Investigation of relationship between object-oriented metrics and change proneness. *International Journal of Machine Learning and Cybernetics*, Springer, v. 4, n. 4, p. 273–286, 2013.

MALHOTRA, R.; KHANNA, M. An exploratory study for software change prediction in object-oriented systems using hybridized techniques. *Automated Software Engineering*, Springer, v. 24, n. 3, p. 673–717, 2017.

MALHOTRA, R. et al. Defect collection and reporting system for git based open source software. In: IEEE. *Data Mining and Intelligent Computing (ICDMIC), 2014 International Conference on*. [S.l.], 2014. p. 1–7.

MANEVA, N.; GROZEV, N.; LILOV, D. A framework for source code metrics. In: ACM. *Proceedings of the 11th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing on International Conference on Computer Systems and Technologies*. [S.l.], 2010. p. 113–118.

MARCUS, A.; POSHYVANYK, D. The conceptual cohesion of classes. In: IEEE. *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*. [S.l.], 2005. p. 133–142.

MARCUS, A.; POSHYVANYK, D.; FERENC, R. Using the conceptual cohesion of classes for fault prediction in object-oriented systems. *IEEE Transactions on Software Engineering*, IEEE, v. 34, n. 2, p. 287–300, 2008.

MARTIN, R. Oo design quality metrics. *An analysis of dependencies*, v. 12, p. 151–170, 1994.

MARTIN, R. C. *Agile software development: principles, patterns, and practices*. [S.l.]: Prentice Hall, 2002.

MATEOS, C. et al. Keeping web service interface complexity low using an oo metric-based early approach. In: IEEE. *Computing Conference (CLEI), 2016 XLII Latin American*. [S.l.], 2016. p. 1–12.

MATHUR, R.; KEEN, K. J.; ETZKORN, L. H. Towards a measure of object oriented runtime cohesion based on number of instance variable accesses. In: ACM. *Proceedings of the 49th Annual Southeast Regional Conference*. [S.l.], 2011. p. 255–257.

MCQUILLAN, J. A.; POWER, J. F. On the application of software metrics to uml models. In: SPRINGER. *International Conference on Model Driven Engineering Languages and Systems*. [S.l.], 2006. p. 217–226.

MEYERS, T. M.; BINKLEY, D. Slice-based cohesion metrics and software intervention. In: IEEE. *Proceedings - Working Conference on Reverse Engineering, WCRE*. [S.l.], 2004. p. 256–265.

MEYERS, T. M.; BINKLEY, D. An empirical study of slice-based cohesion and coupling metrics. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM, v. 17, n. 1, p. 2, 2007.

MISIC, V. B. Cohesion is structural, coherence is functional: Different views, different measures. In: IEEE. *Software Metrics Symposium, 2001. METRICS 2001. Proceedings. Seventh International*. [S.l.], 2001. p. 135–144.

MISRA, S. C.; BHAVSAR, V. C. Relationships between selected software measures and latent bug-density: Guidelines for improving quality. In: SPRINGER. *International Conference on Computational Science and Its Applications*. [S.l.], 2003. p. 724–732.

MITCHELL, A.; POWER, J. F. Runtime coupling metrics for the analysis ofjava programs – preliminary results from spec and grand suites. In: TECHNICAL REPORT NUIMCS- TR2003-07. *Dept. ofComputer Science, National University of Ireland, Maynooth, Co. Kildare,Ireland*. [S.l.], 2003.

MITCHELL, Á.; POWER, J. F. An empirical investigation into the dimensions of run-time coupling in java programs. In: TRINITY COLLEGE DUBLIN. *Proceedings of the 3rd international symposium on Principles and practice of programming in Java*. [S.l.], 2004. p. 9–14.

MITCHELL, A.; POWER, J. F. Run-time cohesion metrics: An empirical investigation. 2004.

MITCHELL, A.; POWER, J. F. Using object-level run-time metrics to study coupling between objects. In: ACM. *Proceedings of the 2005 ACM symposium on Applied computing*. [S.l.], 2005. p. 1456–1462.

MITCHELL, Á.; POWER, J. F. A study of the influence of coverage on the relationship between static and dynamic coupling metrics. *Science of Computer Programming*, Elsevier, v. 59, n. 1-2, p. 4–25, 2006.

MITTAL, P.; SINGH, S.; KAHLON, K. Identification of error prone classes for fault prediction using object oriented metrics. In: SPRINGER. *International Conference on Advances in Computing and Communications*. [S.l.], 2011. p. 58–68.

MORRIS, K. L. *Metrics for object-oriented software development environments*. Tese (Doutorado) — Massachusetts Institute of Technology, 1989.

MOSER, R. et al. A case study on the impact of refactoring on quality and productivity in an agile team. In: *Balancing Agility and Formalism in Software Engineering*. [S.l.]: Springer, 2008. p. 252–266.

MUBARAK, A.; COUNSELL, S.; HIERONS, R. M. An empirical study of "removed" classes in java open-source systems. In: *Advanced Techniques in Computing Sciences and Software Engineering*. [S.l.]: Springer, 2010. p. 99–104.

MUSKENS, J.; CHAUDRON, M.; WESTGEEST, R. Software architecture analysis tool: Software architecture metrics collection. In: STW TECHNOLOGY FOUNDATION. *Proceedings 3rd PROGRESS Workshop on Embedded Systems (Utrecht, The Netherlands, October 24, 2002)*. [S.l.], 2002.

NANDIGAM, J. *A measure for module cohesion*. Tese (Doutorado) — Citeseer, 1995.

NÚÑEZ-VARELA, A. et al. A methodology for obtaining universal software code metrics. *Procedia Technology*, Elsevier, v. 7, p. 336–343, 2013.

NUÑEZ-VARELA, A. S. et al. Source code metrics: A systematic mapping study. *Journal of Systems and Software*, Elsevier, v. 128, p. 164–197, 2017.

OFFUTT, A. J.; HARROLD, M. J.; KOLTE, P. A software metric system for module coupling. *Journal of Systems and Software*, Elsevier, v. 20, n. 3, p. 295–308, 1993.

OFFUTT, J.; ABDURAZIK, A.; SCHACH, S. R. Quantitatively measuring object-oriented couplings. *Software Quality Journal*, Springer, v. 16, n. 4, p. 489–512, 2008.

OKIKE, E. A pedagogical evaluation and discussion about the lack of cohesion in method (lcom) metric using field experiment. *International Journal of Computer Science Issues (IJCSI)*, 2010.

OLAGUE, H. M. et al. Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. *IEEE Transactions on software Engineering*, IEEE, v. 33, n. 6, 2007.

OOSTERMAN, J.; IRWIN, W.; CHURCHER, N. Evojava: A tool for measuring evolving software. In: AUSTRALIAN COMPUTER SOCIETY,

INC. *Proceedings of the Thirty-Fourth Australasian Computer Science Conference*. [S.l.], 2011. v. 113, p. 117–126.

OTT, L. et al. Developing measures of class cohesion for object-oriented software. In: *7th Annual Oregon Workshop on Software Metrics*. [S.l.: s.n.], 1995.

OTT, L. M.; BIEMAN, J. M. Effects of software changes on module cohesion. In: IEEE. *Software Maintenance, 1992. Proceerdings., Conference on*. [S.l.], 1992. p. 345–353.

OTT, L. M.; THUSS, J. J. Slice based metrics for estimating cohesion. In: IEEE. *Software Metrics Symposium, 1993. Proceedings., First International*. [S.l.], 1993. p. 71–81.

PADHY, N.; SINGH, R.; SATAPATHY, S. C. Enhanced evolutionary computing based artificial intelligence model for web-solutions software reusability estimation. *Cluster Computing*, Springer, p. 1–18, 2017.

PADHY, N.; SINGH, R.; SATAPATHY, S. C. Cost-effective and fault-resilient reusability prediction model by using adaptive genetic algorithm based neural network for web-of-service applications. *Cluster Computing*, Springer, p. 1–23, 2018.

PAN, W.-F. et al. Measuring structural quality of object-oriented softwares via bug propagation analysis on weighted software networks. *Journal of Computer Science and Technology*, Springer, v. 25, n. 6, p. 1202–1213, 2010.

PEDRYCZ, W. et al. Using self-organizing maps to analyze object-oriented software measures. *Journal of Systems and Software*, Elsevier, v. 59, n. 1, p. 65–82, 2001.

PEREPLETCHIKOV, M.; RYAN, C.; FRAMPTON, K. Comparing the impact of service-oriented and object-oriented paradigms on the structural properties of software. In: SPRINGER. *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. [S.l.], 2005. p. 431–441.

PONISIO, L.; NIERSTRASZ, O. *Using contextual information to assess package cohesion*. [S.l.], 2006.

PONISIO, M. L. Exploiting client usage to manage program modularity. *Väitöskirja, University of Bern, Sveitsi, kesäkuu*, Citeseer, 2006.

POSHYVANYK, D.; MARCUS, A. The conceptual coupling metrics for object-oriented systems. In: IEEE. *Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on*. [S.l.], 2006. p. 469–478.

POSHYVANYK, D. et al. Using information retrieval based coupling measures for impact analysis. *Empirical software engineering*, Springer, v. 14, n. 1, p. 5–32, 2009.

POWAR, P. et al. Dynamic software metric estimation (dsme): Tool using argouml. *International Journal of Advanced Research in Computer Science*, International Journal of Advanced Research in Computer Science, v. 8, n. 5, 2017.

PRITCHETT, I.; WILLIAM, W. Applying object-oriented metrics to ada 95. *ACM SIGAda Ada Letters*, ACM, v. 16, n. 5, p. 48–58, 1996.

QU, Y. et al. Exploring community structure of software call graph and its applications in class cohesion measurement. *Journal of Systems and Software*, Elsevier, v. 108, p. 193–210, 2015.

RADJENOVIĆ, D. et al. Software fault prediction metrics: A systematic literature review. *Information and Software Technology*, Elsevier, v. 55, n. 8, p. 1397–1418, 2013.

RAGAB, S. R.; AMMAR, H. H. Object oriented design metrics and tools a survey. In: IEEE. *Informatics and Systems (INFOS), 2010 The 7th International Conference on*. [S.l.], 2010. p. 1–7.

RAKESH, K.; SUSHUMNA, S. Implementation of measuring conceptual cohesion. International Journal of Computer Science and Information Technologies (IJCSIT), v. 3, n. 3, p. 4237–4243, 2012.

RATHEE, A.; CHHABRA, J. K. Improving cohesion of a software system by performing usage pattern based clustering. *Procedia Computer Science*, Elsevier, v. 125, p. 740–746, 2018.

RATHORE, M. N. P. S.; GUPTA, R. A novel class, object and inheritance based coupling measure (coicm) to find better oop paradigm using java. *International Journal*, Citeseer, 2011.

RATHORE, S. S.; GUPTA, A. Investigating object-oriented design metrics to predict fault-proneness of software modules. In: IEEE. *Software Engineering (CONSEG), 2012 CSI Sixth International Conference on*. [S.l.], 2012. p. 1–10.

RATHORE, S. S.; GUPTA, A. A comparative study of feature-ranking and feature-subset selection techniques for improved fault prediction. In: ACM. *Proceedings of the 7th India Software Engineering Conference*. [S.l.], 2014. p. 7.

RATHORE, S. S.; KUMAR, S. A study on software fault prediction techniques. *Artificial Intelligence Review*, Springer, p. 1–73, 2017.

RATHORE, S. S.; KUMAR, S. Towards an ensemble based system for predicting the number of software faults. *Expert Systems with Applications*, Elsevier, v. 82, p. 357–382, 2017.

RAZ, T.; YAUNG, A. Process clustering with an algorithm based on a coupling metric. *Journal of Systems and Software*, Elsevier, v. 22, n. 3, p. 217–223, 1993.

REDDY, B. R.; KHURANA, S.; OJHA, A. Software maintainability estimation made easy: a comprehensive tool coin. In: ACM. *Proceedings of the Sixth International Conference on Computer and Communication Technology*. [S.l.], 2015. p. 68–72.

REDDY, G. V.; CHANDRASEKHAR, A. Tools and techniques for testing of flight critical software. *Defence Science Journal*, Defence Scientific Information & Documentation Centre, v. 49, n. 4, p. 317, 1999.

ROCHA, H. et al. Mining the impact of evolution categories on object-oriented metrics. *Software Quality Journal*, Springer, v. 21, n. 4, p. 529–549, 2013.

SANTHOSHINI, G.; ANBAZHAGAN, K. An object based software tool for software measurement. In: IEEE. *Information Communication and Embedded Systems (ICICES), 2014 International Conference on*. [S.l.], 2014. p. 1–5.

SANTOS, D.; AFONSO, P.; COSTA, H. Srt—a computational tool for restructuring java software. In: IEEE. *35th International Conference of the Chilean Computer Science Society (SCCC)*. [S.l.], 2016. p. 1–12.

SANTOS, M. et al. Metrics and statistical techniques used to evaluate internal quality of object-oriented software: A systematic mapping. In: IEEE. *Computer Science Society (SCCC), 2016 35th International Conference of the Chilean*. [S.l.], 2016. p. 1–11.

SANYAL, A.; SINGH, B. A systematic literature survey on various techniques for software fault prediction. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2014.

SARKAR, S.; KAK, A. C.; RAMA, G. M. Metrics for measuring the quality of modularization of large-scale object-oriented software. *IEEE Transactions on Software Engineering*, IEEE, v. 34, n. 5, p. 700–720, 2008.

SATO, D.; GOLDMAN, A.; KON, F. Tracking the evolution of object-oriented quality metrics on agile projects. In: SPRINGER. *International Conference on Extreme Programming and Agile Processes in Software Engineering*. [S.l.], 2007. p. 84–92.

SCOTTO, M. et al. A relational approach to software metrics. In: ACM. *Proceedings of the 2004 ACM symposium on Applied computing*. [S.l.], 2004. p. 1536–1540.

SEMA, G. *User Manual Concerto2/Audit-CC++*. [S.l.]: 56 rue Roger Salengro, 94126 Fontenay-Sous-Bois Cedex - France, 1998.

SHARMA, H.; CHUG, A. Dynamic metrics are superior than static metrics in maintainability prediction: An empirical case study. In: IEEE. *Reliability, Infocom Technologies and Optimization (ICRITO)(Trends and Future Directions), 2015 4th International Conference on*. [S.l.], 2015. p. 1–6.

SHATNAWI, R. The application of roc analysis in threshold identification, data imbalance and metrics selection for software fault prediction. *Innovations in Systems and Software Engineering*, Springer, v. 13, n. 2-3, p. 201–217, 2017.

SHATNAWI, R.; LI, W. The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. *Journal of systems and software*, Elsevier, v. 81, n. 11, p. 1868–1882, 2008.

SILVA, B. et al. Concern-based cohesion: Unveiling a hidden dimension of cohesion measurement. In: IEEE. *Program Comprehension (ICPC), 2012 IEEE 20th International Conference on*. [S.l.], 2012. p. 103–112.

SINGH, G. Metrics for measuring the quality of object-oriented software. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 38, n. 5, p. 1–5, 2013.

SINGH, P.; SINGH, H. Dynametrics: a runtime metric-based analysis tool for object-oriented software systems. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 33, n. 6, p. 1–6, 2008.

SINGH, P.; SINGH, H. Class-level dynamic coupling metrics for static and dynamic analysis of object-oriented systems. *International Journal of Information and Telecommunication Technology*, v. 1, n. 1, p. 16–28, 2010.

SINGH, S.; KAHLON, K. Object oriented software metrics threshold values at quantitative acceptable risk level. *CSI transactions on ICT*, Springer, v. 2, n. 3, p. 191–205, 2014.

SINGH, Y.; KAUR, A.; MALHOTRA, R. Prediction of software quality model using gene expression programming. In: SPRINGER. *International Conference on Product-Focused Software Process Improvement*. [S.l.], 2009. p. 43–58.

SINGH, Y.; KAUR, A.; MALHOTRA, R. A comparative study of models for predicting fault proneness in object-oriented systems. *International Journal of Computer Applications in Technology*, Inderscience Publishers Ltd, v. 49, n. 1, p. 22–41, 2014.

SIRBI, K. S.; KULKARNI, P. J. Coupling metrics for aspect oriented programming. *International Journal of Advanced Research in Computer Science*, International Journal of Advanced Research in Computer Science, v. 1, n. 3, 2010.

SPRINGER, J. et al. Accelerating sca compliance testing with advanced development tools. *Analog Integrated Circuits and Signal Processing*, Springer, p. 1–13, 2015.

SUBRAMANIAN, G.; CORBIN, W. An empirical study of certain object-oriented software metrics. *Journal of Systems and Software*, Elsevier, v. 59, n. 1, p. 57–63, 2001.

SURESH, Y.; KUMAR, L.; RATH, S. K. Statistical and machine learning methods for software fault prediction using ck metric suite: a comparative analysis. *ISRN Software Engineering*, Hindawi Publishing Corporation, v. 2014, 2014.

SURI, B.; SINGHAL, S. Investigating the oo characteristics of software using ckjm metrics. In: IEEE. *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO): Trends and Future Directions*. [S.l.], 2015. p. 1–6.

SYSTA, T. On the relationships between static and dynamic models in reverse engineering java software. In: IEEE. *Reverse Engineering, 1999. Proceedings. Sixth Working Conference on*. [S.l.], 1999. p. 304–313.

SYSTA, T.; YU, P.; MULLER, H. Analyzing java software by combining metrics and program visualization. In: IEEE. *Software Maintenance and Reengineering, 2000. Proceedings of the Fourth European*. [S.l.], 2000. p. 199–208.

TANG, M.-H.; KAO, M.-H.; CHEN, M.-H. An empirical study on object-oriented metrics. In: IEEE. *Software Metrics Symposium, 1999. Proceedings. Sixth International*. [S.l.], 1999. p. 242–249.

TEMPERO, E.; RALPH, P. A model for defining coupling metrics. In: IEEE. *Software Engineering Conference (APSEC), 2016 23rd Asia-Pacific*. [S.l.], 2016. p. 145–152.

TEMPERO, E.; RALPH, P. A framework for defining coupling metrics. *Science of Computer Programming*, Elsevier, 2018.

TOMAS, P.; ESCALONA, M. J.; MEJIAS, M. Open source tools for measuring the internal quality of java software products. a survey. *Computer Standards & Interfaces*, Elsevier, v. 36, n. 1, p. 244–255, 2013.

TOURE, F.; BADRI, M.; LAMONTAGNE, L. A metrics suite for junit test code: a multiple case study on open source software. *Journal of Software Engineering Research and Development*, SpringerOpen, v. 2, n. 1, p. 14, 2014.

TOURE, F.; BADRI, M.; LAMONTAGNE, L. Predicting different levels of the unit testing effort of classes using source code metrics: a multiple case study on open-source software. *Innovations in Systems and Software Engineering*, Springer, v. 14, n. 1, p. 15–46, 2018.

TSUI, F.; IRIELE, S. Analysis of software cohesion attribute and test case development complexity. In: ACM. *Proceedings of the 49th Annual Southeast Regional Conference*. [S.l.], 2011. p. 237–242.

VANDERFEESTEN, I.; REIJERS, H. A.; AALST, W. M. Van der. Evaluating workflow process designs using cohesion and coupling metrics. *Computers in industry*, Elsevier, v. 59, n. 5, p. 420–437, 2008.

VEADO, L. et al. Tdtool: threshold derivation tool. In: ACM. *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*. [S.l.], 2016. p. 24.

VERNAZZA, T. et al. Defining metrics for software components. In: CITESEER. *5th World Multi-Conference on Systemics, Cybernetics and Informatics, Florida*. [S.l.], 2000. v. 11, p. 16–23.

VIJI, C.; RAJKUMAR, N.; DURAISAMY, S. Prediction of software fault-prone classes using an unsupervised hybrid som algorithm. *Cluster Computing*, Springer, p. 1–11, 2018.

WANG, J. et al. Dmc: a more precise cohesion measure for classes. *Information and Software Technology*, Elsevier, v. 47, n. 3, p. 167–180, 2005.

WANJIKU, R.; OKEYO, G.; CHERUIYOT, W. Scoped class cohesion metric for software process assessment. *International Journal of Computer Science Issues (IJCSI)*, International Journal of Computer Science Issues (IJCSI), v. 13, n. 2, p. 12, 2016.

WASIQ, M. *Measuring class cohesion in object-oriented systems*. Tese (Doutorado) — King Fahd University of Petroleum and Minerals, 2001.

WEISER, M. Program slicing. In: IEEE PRESS. *Proceedings of the 5th international conference on Software engineering*. [S.l.], 1981. p. 439–449.

WEN, J. et al. Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*, Elsevier, v. 54, n. 1, p. 41–59, 2012.

WOO, G. et al. Revising cohesion measures by considering the impact of write interactions between class members. *Information and Software Technology*, Elsevier, v. 51, n. 2, p. 405–417, 2009.

XENOS, M. et al. Object-oriented metrics-a survey. In: *Proceedings of the FESMA*. [S.l.: s.n.], 2000. p. 1–10.

XIE, T. et al. Jboomt: Jade bird object-oriented metrics tool. *Chinese Journal of Electronics*, 2000.

XU, B.; ZHOU, Y. Comments on 'a cohesion measure for object-oriented classes' by heung seok chae, yong rae kwon and doo hwan bae (softw. pract. exper. 2000; 30: 1405–1431). *Software: Practice and Experience*, Wiley Online Library, v. 31, n. 14, p. 1381–1388, 2001.

YACOUB, S. M.; AMMAR, H. H.; ROBINSON, T. Dynamic metrics for object oriented designs. In: IEEE. *Software Metrics Symposium, 1999. Proceedings. Sixth International*. [S.l.], 1999. p. 50–61.

YADAV, H. B.; YADAV, D. K. Construction of membership function for software metrics. *Procedia Computer Science*, Elsevier, v. 46, p. 933–940, 2015.

YANG, X. *Research on cohesion measures for classes*. Tese (Doutorado) — Master Thesis, Department of Computer Science & Engineering in Southeast University, 2002.

YANG, Y. et al. An empirical investigation into the effect of slice types on slice-based cohesion metrics. *Information and Software Technology*, Elsevier, v. 75, p. 90–104, 2016.

YANG, Y. et al. Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? an empirical study. *IEEE Transactions on Software Engineering*, IEEE, v. 41, n. 4, p. 331–357, 2015.

YOHANNESE, C. W.; LI, T. A combined-learning based framework for improved software fault prediction. *International Journal Of Computational Intelligence Systems*, v. 10, n. 1, 2017.

YU, L.; CHEN, K.; RAMASWAMY, S. Multiple-parameter coupling metrics for layered component-based software. *Software Quality Journal*, Springer, v. 17, n. 1, p. 5–24, 2009.

YU, P.; SYSTA, T.; MULLER, H. Predicting fault-proneness using oo metrics. an industrial case study. In: IEEE. *Software Maintenance and Reengineering, 2002. Proceedings. Sixth European Conference on*. [S.l.], 2002. p. 99–107.

ZAIDMAN, A.; DEMEYER, S. Analyzing large event traces with the help of coupling metrics. In: *Proc. the 5th International Workshop on OO Reengineering, Oslo, Norway*. [S.l.: s.n.], 2004.

ZHAO, Y. et al. An empirical analysis of package-modularization metrics: Implications for software fault-proneness. *Information and Software Technology*, Elsevier, v. 57, p. 186–203, 2015.

ZHAO, Y. et al. Understanding the value of considering client usage context in package cohesion for fault-proneness prediction. *Automated Software Engineering*, Springer, v. 24, n. 2, p. 393–453, 2017.

ZHOU, T. et al. Measuring package cohesion based on context. In: IEEE. *Semantic Computing and Systems, 2008. WSCS'08. IEEE International Workshop on*. [S.l.], 2008. p. 127–132.

ZHOU, T.-L. et al. Measuring package cohesion based on client usages. *Journal of Software*, v. 20, n. 2, p. 256–270, 2009.

ZHOU, Y. et al. An in-depth investigation into the relationships between structural metrics and unit testability in object-oriented systems. *Science china information sciences*, Springer, v. 55, n. 12, p. 2800–2815, 2012.

ZHOU, Y. et al. Icbmc: an improved cohesion measure for classes. In: IEEE. *Software Maintenance, 2002. Proceedings. International Conference on*. [S.l.], 2002. p. 44–53.

A APPENDIX

A.1 DESCRIPTION OF COHESION METRICS

Table 17: Description of cohesion metrics

#	Cohesion Metrics	Description
1	**A** (*Adhesiveness*) (BIEMAN; OTT, 1994)	The extent to which the glue tokens in the module are adhesive (measured as the ratio of the amount of the adhesiveness to the total possible adhesiveness)
2	**AAC** (*Attribute-Attribute Cohesion*) (DALLAL; BRIAND, 2010)	The AAC is the average cohesion of all pairs of attributes
3	**AMC** (*Attribute-Method Cohesion*) (DALLAL; BRIAND, 2010)	The AMC is the ratio of the number of 1s in the AT (Attribute Type) matrix to the total size of the matrix
4	**APIU** (*API usage index*) (SARKAR; KAK; RAMA, 2008)	The extent to which the cohesiveness and segregation properties are followed by the S-APIs of a module
5	**C3** (*Conceptual Cohesion of Classes*) (MARCUS; POSHYVANYK; FERENC, 2008)	C3 captures the conceptual aspects of class cohesion, as it measures how strongly the methods of a class relate to each other conceptually. C3 is based on the analysis of textual information in the source code, expressed in comments and identifiers. Once again, this part of the source code, although closer to natural language, is still different from it.
6	**CAM** (*Cohesion Among Methods of Class*) (BANSIYA; DAVIS, 2002)	This metric computes the relatedness among methods of a class based upon the parameter list of the methods. A metric value close to 1.0 (range 0 to 1) is preferred.
7	**CAMC** (*Cohesion Among Methods in a Class*) (BANSIYA; DAVIS; ETZKORN, 1999)	The CAMC metric is defined as the ratio of the total number of 1s in the matrix to the total size of the matrix
8	**CAMCs** (*Cohesion among methods of a class with 'self' parameter*) (COUNSELL; SWIFT; CRAMPTON, 2006) (KAUR; SINGH, 2010)	A variation on CAMC metric. The CAMCs metric includes the "self" parameter type in the parameter occurrence matrix.
9	**CBAMU** (*Cohesion Based on Attribute and Method Usage*) (ADAM, 2004)	CBAMU is defined based on both attribute usage and method usage (invocation) within a class. The metric does not only considers, in its definition, the direct interaction between methods and attributes but also the interaction between methods which may serve as a means of capturing the indirect relationship among methods. The metric (CBAMU) is also normalized so that cohesion values obtained from the metric lies between 0 and 1. The CBAMU of a class = 0 if there is no interactions among the components of the class and CBAMU = 1 if all methods are directly or indirectly connected (i.e. if the interactions among the components of the class is maximal).
10	**CBMC** (*Cohesion Based on Member Connectivity*) (CHAE; KWON; BAE, 2000) (XU; ZHOU, 2001)	CBMC, a metric that accounts for the connectivity pattern in the reference graph Gc that represents class C. This graph does not represent special methods (i.e., constructors, destructors, delegation, and access methods). The set of glue methods is the minimum set of methods by which their removal causes the graph to become disjointed.
11	**CC** (*Class Cohesion*) (BONJA; KIDANMARIAM, 2006)	CC is the ratio of the summation of the similarities between all pairs of methods to the total number of pairs of methods

12	**CC** (*Component cohesion*) (VERNAZZA et al., 2000)	CC is the percentage of directed relationships between intra-package classes
13	**CCM** (*Class Connection* *Metric*) (WASIQ, 2001) (JARALLAH; WASIQ; AHMED, 2001)	CCM is based on the connection graph GC of the class C. The connection graph GC has one vertex for each method of the class and there is an edge between two vertices if and only if the corresponding methods are connected according to the connection criterion defined by the metric.
14	**CDC** (*Class Level* *Cohesion Measures*) (GUPTA; CHHABRA, 2009)	Class level Dynamic Cohesion for a class is defined as the average of the values of Object level Dynamic Cohesion for all objects of a class created at run-time
15	**Co'** (ZHOU et al., 2012)	A variation on Co metric
16	**Co** (*Connectivity*) (HITZ; MONTAZERI, 1995)	The definition of Co considers the impact of the method invocation. For two methods mi and mj, they are considered to be related when one of the methods has an invocation to the other or they reference some common instance variables.
17	**Coh** (*Cohesion of module*) (ALLEN; KHOSHGOFTAAR; CHEN, 2001)	Coh accounts for the relative number of attributes referenced by methods, its value ranges within the interval [0, 1], and it is undefined for classes that do not include methods or attributes
18	**Coverage** (WEISER, 1981) (OTT; BIEMAN, 1992) (OTT; THUSS, 1993)	Coverage compares the length of slices to the length of the entire module
19	**CPC** (*Contextual package* *cohesion*) (PONISIO, 2006)	The extent to which the classes reused by common clients appear in the same package
20	**CR** (*Cohesion Ratio*) (FENTON; BIEMAN, 2014)	The ratio of the number of functionally cohesive classes to the total number of classes
21	**CU** (*Common use*) (PONISIO; NIERSTRASZ, 2006)	The extent to which the classes in a package are used together by common clients
22	**DC_AAX** (*Dynamic Cohesion* *due to Reference* *dependency between* *attributes*) (GUPTA; CHHABRA, 2011) (GUPTA, 2011)	This category of cohesion exists between attributes of an object when these attributes are referred together in a method of the object during program execution. The reference of a pair of attributes together in a single method induces some dependency between them.
23	**DC_AMX** (*Dynamic Cohesion* *due to Write* *dependency of Attributes* *on Methods*) (GUPTA; CHHABRA, 2011) (GUPTA, 2011)	This type of dynamic cohesion exists between attributes and methods of an object when a method of an object writes an attribute of the object during execution of a specific scenario x
24	**DC_MAX** (*Dynamic Cohesion* *due to Read* *dependency of Methods* *on Attributes*) (GUPTA; CHHABRA, 2011) (GUPTA, 2011)	This kind of cohesion exists between methods and attributes of an object when a method of an object reads an attribute during execution of a specific scenario x
25	**DC_MMX** (*Dynamic Cohesion* *due to Call* *dependency between* *Methods*) (GUPTA; CHHABRA, 2011) (GUPTA, 2011)	This type of cohesion exists between a pair of methods of an object when a method mi calls other method mj of the object during program execution

26	**DCACC** (*Dynamic Access Cohesion*) (GUPTA; CHHABRA, 2009)	This type of dynamic cohesion for an object o is defined as the ratio of actual number of distinct dependence relations between all methods and all attributes to the maximum possible number of dependence relations of this type between them
27	**DCC** (*Dynamic Class Cohesion*) (GUPTA; CHHABRA, 2011) (GUPTA, 2011)	Dynamic Class Cohesion is defined as the average of the values of Dynamic Object Cohesion for all objects of a class created during all execution scenarios of the application
28	**DCCALL** (*Dynamic Call Cohesion*) (GUPTA; CHHABRA, 2009)	This kind of dynamic cohesion of an object o is defined as the ratio of actual count of distinct dependence relations between all ordered pairs of methods to the maximum possible number of relations of this type between them
29	**DCD** (*Degree of Cohesion-Direct*) (BADRI; BADRI, 2004)	The DCD measures the fraction of the directly connected pairs of methods where two methods are directly connected if they are directly connected to an attribute or if they directly or transitively invoke the same method
30	**DCDE** (*Cohesion Based on the Direct Relation*) (BADRI; BADRI; GUEYE, 2008)	Let ED be the number of edges in the graph GD. The degree of cohesion in the class C based on the direct relation between its public methods is defined as: $DCDE = \|ED\|/[n * (n-1)/2] \in [0,1]$
31	**DCI** (*Degree of Cohesion-Indirect*) (BADRI; BADRI, 2004)	The DCI measures the fraction of the directly and transitively connected pairs of methods where the two methods are transitively connected if they are directly or indirectly connected to an attribute or if the two methods directly or transitively invoke the same method
32	**DCIE** (*Cohesion Based on the Indirect Relation*) (BADRI; BADRI; GUEYE, 2008)	Let EI be the number of edges in the graph GI. The degree of cohesion in the class C in this case (direct and indirect relations) is defined as: $DCIE = \|EI\|/[n * (n-1)/2] \in [0,1]$
33	**DCO** (*Degree of cohesion objects*) (MORRIS, 1989)	The average number of client packages with respect to all classes in a package
34	**DMC** (*Dependence Matrix based Class cohesion measure*) (WANG et al., 2005)	Class cohesion metric based on a dependency matrix that represents the degree of dependence among the instance variables and methods in a class
35	**DOCX** (*Dynamic Object Cohesion*) (GUPTA; CHHABRA, 2011) (GUPTA, 2011)	Dynamic cohesion of an object i in an application scope is the average of the dynamic cohesion values measured during all execution scenarios of the application for the object
36	**ECCM** (*Enhanced Class Connection Metric*) (WASIQ, 2001) (JARALLAH; WASIQ; AHMED, 2001)	ECCM(C)=CCM(C) x (1- PenaltyFactor(C)), Where CCM(C) is Class Connection Metric, CCM is based on the connection graph GC of the class C, PenaltyFactor(C)= NORM(C)/NOIM(C), NORM(C) is the number of re-implemented methods and NOIM(C) is the number of inherited methods.
37	**Fcoh** (*Functional coherence*) (MISIC, 2001)	The extent to which the classes in a package contribute toward a common purpose
38	**H** (*Relational Cohesion*) (LORENZ; KIDD, 1994)	The connected components formed by the classes and interfaces of the package
39	**HC** (*Hamming cohesiveness*) (ZHOU et al., 2009)	The extent to which the classes in a package are always reused together by its clients
40	**iCAMCs** (ZHOU et al., 2012)	A variation on CAMCs metric
41	**ICBMC** (*Improved Cohesion Based on Member Connectivity*) (ZHOU et al., 2002)	ICBMC is an improved version of CBMC, that considers the cut sets instead of glue methods. The cut set is the minimum set of edges such that their removal causes the method-attribute interaction graph to become disjoint

42	**ICH** (*Information-flow based Cohesion*) (LEE et al., 1995)	ICH for a method is defined as the number of invocations of other methods of the same class, weighted by the number of parameters of the invoked method. The ICH of a class is the sum of the ICH values of its methods.
43	**ILCO** (*Internal lack of cohesion*) (PONISIO; NIERSTRASZ, 2006)	ILCO is the degree of internal lack of cohesion
44	**iNHDs** (ZHOU et al., 2012)	A variation on NHDs metric
45	**IPSC** (*Index of package services cohesion*) (ABDEEN; DUCASSE; SAHRAOUI, 2011)	The average cohesiveness of package services from the similarity-of-purpose perspective
46	**iSNHDs** (ZHOU et al., 2012)	A variation on SNHDs metric
47	**LCbC** (*Lack of Concern-based Cohesion*) (SILVA et al., 2012)	LCbC captures whether a module is cohesive or not by counting the number of concerns it implements. LCbC relies on the identification of the source code elements implementing each concern of a system. LCbC simply looks into what matters for the implementation of a module - the amount of concerns placed on it.
48	**LCC** (*Loose Class Cohesion*) (BIEMAN; KANG, 1995)	LCC is the relative number of directly or transitively connected pairs of methods, wherein two methods are transitively connected if they are directly or indirectly connected to an attribute
49	**LCCD** (*Lack of Cohesion in the Class-Direct*) (BADRI; BADRI, 2004)	$LCCD = 1 - DCD \in [0, 1]$. DCD gives the percentage of public methods pairs, which are directly (as defined below) related. The degree of cohesion in the class C based on the direct relation between its public methods is defined as: $DCD = \|ED\|/[n * (n-1)/2] \in [0, 1]$.
50	**LCCI** (*Lack of Cohesion in the Class-Indirect*) (BADRI; BADRI, 2004)	The lack of cohesion in the class C is then given by: $LCCI = 1 - DCI \in [0, 1]$. DCI gives the percentage of methods pairs, which are directly or indirectly related. The degree of cohesion in the class C based on the direct and indirect relations between its public methods is defined as: $DCI = \|EI\|/[n * (n-1)/2] \in [0, 1]$.
51	**LCD** (*Lack of cohesion - derived from LCOM*) (BADRI; BADRI; TOURE, 2011)	Let us consider a class C with n methods. The number of methods pairs is $[n * (n-1)/2]$. Consider an undirected graph GD, where the vertices are the methods of the class C, and there is an edge between two vertices if the corresponding methods are directly related. Let ED be the number of edges in the graph GD. $LCD = [n * (n-1)/2] - 2 * \|ED\|$. When the difference is negative, LCD is set to zero.
52	**LCIC** (*Lack of Coherence in Clients*) (MÄKELÄ; LEPPÄNEN, 2009)	LCIC that measures how coherently the clients of a class use it. If all clients use all features of the class, then it has clients that have a similar view of the class. Otherwise, the class might have unnecessary features, or it might implement several different concepts.
53	**LCOM1** (*Lack of Cohesion in Methods 1*) (CHIDAMBER; KEMERER, 1991)	The number of pairs of methods in the class using no instance variables in common
54	**LCOM2** (*Lack of Cohesion in Methods 2*) (CHIDAMBER; KEMERER, 1994)	LCOM2 is the number of pairs of methods in the class using no Attributes in common, minus the number of pairs of methods that do. If this difference is negative, LCOM2 will set to zero
55	**LCOM3** (*Lack of Cohesion in Methods 3*) (LI; HENRY, 1993a)	Consider an undirected graph G, where the vertices are the methods of a class, and there is an edge between two vertices if the corresponding methods use at least an attribute in common. LCOM3 is then defined as the number of connected components of G.
56	**LCOM4** (*Lack of Cohesion in Methods 4*) (HITZ; MONTAZERI, 1995)	LCOM4 is based on counting the number of interactions rather than depending on the connectivity pattern. That is, the connectivity factor does not distinguish between two connected graphs that have the same number of nodes and edges but different connectivity patterns.

57	**LCOM5** (*Lack of Cohesion in Methods 5*) (HENDERSON-SELLERS, 1996)	Measures a lack of cohesion in methods by considering the number of methods referencing each attribute. In this case, $LCOM5 = (a - kl)/(l - kl)$, where l is the number of attributes, k is the number of methods, and a is the number of distinct attributes accessed in the methods of a class.
58	**LSCC** (*Low-level design Similarity-based Class Cohesion*) (DALLAL; BRIAND, 2012)	LSCC is based on a precise MMI definition that satisfies widely accepted class cohesion properties and is useful as an indicator for restructuring weakly cohesive classes. Also, LSCC can easily account for class inheritance and direct and transitive interactions. In addition, LSCC differentiate between various types of methods (i.e., access methods, constructors, and destructors).
59	**MaxCoverage** (OTT; BIEMAN, 1992) (OTT; THUSS, 1993)	MaxCoverage is the ratio of the largest slice in a module to the module's length
60	**MinCoverage** (OTT; BIEMAN, 1992) (OTT; THUSS, 1993)	MinCoverage is the ratio of the smallest slice in a module to the module's length
61	**MMAC** (*Method-Method through Attributes Cohesion*) (DALLAL; BRIAND, 2010)	MMAC considers method-method interactions, and it is defined as the ratio of the summation of the similarities between all pairs of methods to the total number of possible pairs of methods
62	**MMIC** (*Method–Method Invocation Cohesion*) (DALLAL; BRIAND, 2010)	MMIC is the average number of method–method-invocation interactions. This is represented by the ratio of the number of 1s in the MI matrix to the total size of the matrix.
63	**NHD** (*Normalised Hamming Distance*) (COUNSELL et al., 2002) (COUNSELL; SWIFT; CRAMPTON, 2006)	The metric calculates the average parameter agreement between each pair of methods. The parameter agreement between a pair of methods is defined as the number of entries in which the corresponding rows in the parameter-occurrence matrix match.
64	**NHDs** (*NHD with self parameter*) (COUNSELL; SWIFT; CRAMPTON, 2006) (KAUR; SINGH, 2010)	A variation on NHD metric. NHDs is defined by considering the 'self' parameter.
65	**NRC** (*Normalized relational cohesion*) (MARTIN, 2002)	The average number of intra-package dependencies per class
66	**NRCI** (*Neutral Ratio of Cohesive Interactions*) (BRIAND; MORASCA; BASILI, 1994)	$NRCI(sp) = \|K(sp)\|/[\|M(sp)\| - \|U(sp)\|]$, where K(sp) is the set of known interactions of a software part sp, U(sp) is the set of unknown interactions, and M(sp) is the maximal set of cohesive interactions of the software part sp, i.e., the set that includes all of sp's possible cohesive interactions, obtained by linking every data declaration to every other data declaration and subroutine with which it can interact. NRCI(sp) is undefined if and only if all interactions are unknown.
67	**Number of Private Methods Accessed at Runtime** (SINGH; SINGH, 2008)	Is the number of private methods accessed at runtime
68	**Number of Public Methods Accessed at Runtime** (SINGH; SINGH, 2008)	Is the number of public methods accessed at runtime

69	**OCC** (*Optimistic Class Cohesion*) (AMAN et al., 2002)	Consider an undirected graph G, w here the vertices are the methods of a class, and there is an edge between two vertices if the corresponding methods directly or indirectly reference at least one attribute in common. Let n be the total number of methods in the class. Then, OCC is the maximum of the ratios of the number of methods reachable from method $mi(1 \leq i \leq n)$ in the class to $n - 1$ if $n > 1$ or 0 otherw ise.
70	**ODC** (*Object Level Cohesion Measures*) (GUPTA; CHHABRA, 1993)	Object level dynamic cohesion for an object is defined as the weighted summation of two types of cohesions
71	**OL2** (YANG, 2002)	OL2 is a variation on OLn metric, for n=2
72	**OL3** (YANG, 2002)	OL3 is a variation on OLn metric, for n=3
73	**OLn** (YANG, 2002)	OLn is defined as the average strengths of the attributes in the class.
74	**ORCI** (*Optimistic Ratio of Cohesive Interactions*) (BRIAND; MORASCA; BASILI, 1994)	$ORCI(sp) = [\|K(sp)\| + \|U(sp)\|]/\|M(sp)\|$, where K(sp) is the set of known interactions of a software part sp, U(sp) is the set of unknown interactions, and M(sp) is the maximal set of cohesive interactions of the software part sp, i.e., the set that includes all of sp's possible cohesive interactions, obtained by linking every data declaration to every other data declaration and subroutine with which it can interact
75	**Overlap** (WEISER, 1981) (OTT; BIEMAN, 1992) (OTT; THUSS, 1993)	Overlap is a measure of how many statements in a slice are found only in that slice
76	**PCC** (*Pessimistic Class Cohesion*) (AMAN et al., 2002)	Same as OCC, except G is a directed graph, in which there is an arc from one vertice representing method m1 to another vertice representing method m2 if m1 directly or indirectly w rites attribute a and m2 directly or indirectly reads a.
77	**PCCC** (*Path Connectivity Class Cohesion*) (DALLAL, 2012a)	PCCC metric, which is a connectivity metric that accounts for the degree of connectivity among the class members. This metric is based on counting the number of class reference graph paths that satisfy a certain criterion. PCCC is undefined for the meaningless case of a class with no attributes and methods. If a class with several methods does not have any attributes, the methods will be considered unrelated, and the cohesion will be the minimum value, which is zero. If a class with attributes does not have methods, the attributes declare a class structure that does not have behavior. In this case, the attributes describe the features of the same object, and the class is expected to be fully cohesive.
78	**PCM** (*Package cohesion metrics*) (GORMAN, 2006)	The average percentage of the internal classes directly or indirectly depended by each class in the package
79	**Pcoh** (*Package cohesion measurement*) (GUPTA; CHHABRA, 2012)	The degree of intra-package dependencies among its elements
80	**PF** (*Index of package goal focus*) (ABDEEN; DUCASSE; SAHRAOUI, 2011)	The ratio of the average number of interfaces used by client packages to the total number of interfaces in a package
81	**PRCI** (*Pessimistic Ratio of Cohesive Interactions*) (BRIAND; MORASCA; BASILI, 1994)	$PRCI(sp) = \|K(sp)\|/\|M(sp)\|$, where K(sp) is the set of known interactions of a software part sp, and M(sp) is the maximal set of cohesive interactions of the software part sp, i.e., the set that includes all of sp's possible cohesive interactions, obtained by linking every data declaration to every other data declaration and subroutine with which it can interact

82	**RCI** (*Ratio of Cohesive Interactions*) (BRIAND; MORASCA; BASILI, 1994)	The Ratio of Cohesive Interactions for sp is $RCI(sp) = \|CI(sp)\|/\|M(sp)\|$, Where CI(sp) is is the union of the sets of DS-interactions (Data declaration-Subroutine interactions) and DD-interactions (Data declaration-Data declaration Interaction), with the exception of those DD-interactions between a data declaration and a subroutine formal parameter. M(sp) is the maximal set of cohesive interactions of the software part sp, i.e., the set that includes all of sp's possible cohesive interactions, obtained by linking every data declaration to every other data declaration and subroutine with which it can interact.
83	**RLCOM** (*Run-time Simple LCOM*) (MITCHELL; POWER, 2004b) (MITCHELL; POWER, 2004a)	The RLCOM metric uses pairs of methods
84	**SBFC** (*Similarity-Based Functional Cohesion)*, (DALLAL, 2009)	The extent to which the slices are similar (measured as the average degree of the normalized similarity between columns in the data-token-level slice occurrence matrix of the module)
85	**SCC** (*Similar context cohesiveness*) (ZHOU et al., 2008)	The ratio of the sum of weights of context and data relations to the number of all possible context and data relations
86	**SCC** (*Similarity-based Class Cohesion*) (DALLAL; BRIAND, 2010)	The SCC metric uses a matrix called a Direct Attribute Type (DAT) matrix, which has a row for each method and a column for each distinct parameter type that matches an attribute type. The value in row i and column j in the matrix equals 1 when the jth data type is a type of at least one of the parameters or return of the ith method, and it equals 0 otherwise. The SCC metric is the weighted average of four different metrics that consider methodmethod, attribute-attribute, and attribute-method direct and transitive interactions.
87	**SCCM** (*Scoped Class Cohesion Metric*) (WANJIKU; OKEYO; CHERUIYOT, 2016)	The metric values used by the SCCM use a rational scale with a minimum value of natural 0 and a maximum value of 1
88	**SCOM** (*Sensitive Class Cohesion Metric*) (FERNÁNDEZ; PEÑA, 2006)	SCOM considers the cardinality of the intersection between each pair of methods
89	**SFC** (*Strong functional cohesion*) (BIEMAN; OTT, 1994)	The extent to which all the slices in the module belong together (measured as the ratio of the number of super-glue tokens to the total number of data tokens of the module)
90	**SNHD** (*Scaled Normalised Hamming Distance*) (COUNSELL; SWIFT; CRAMPTON, 2006)	Scaled NHD (SNHD) is a metric that represents the closeness of the NHD metric to the maximum value of NHD compared to the minimum value, and therefore, it can be used as a basis for normalizing NHD. The SNHD value ranges within the interval [1, 1], where the closer the value is to zero, the less cohesive is the class.
91	**SNHDs** (*Scaled NHD with self parameter*) (COUNSELL; SWIFT; CRAMPTON, 2006) (KAUR; SINGH, 2010)	A variation on SNHD metric. SNHDs is defined by considering the 'self' parameter.
92	**TCC** (*Tight Class Cohesion*) (OTT et al., 1995) (BIEMAN; KANG, 1995)	The measure TCC is defined as the percentage of pairs of public methods of the class with common attribute usage
93	**Tightness** (WEISER, 1981) (OTT; BIEMAN, 1992) (OTT; THUSS, 1993)	Tightness measures the number of statements in every slice

94	**TLCOM** (*Transitive Lack of Cohesion in Methods*) (DALLAL, 2011c)	Transitive LCOM (TLCOM) is defined as the number of method pairs that directly or transitively share a common attribute. A method transitively references an attribute when the method directly or indirectly calls another method that directly references the attribute. A pair of methods "transitively" shares a common attribute when the common attribute is referenced transitively by both methods or referenced transitively by one of the methods and directly by the other.
95	**WFC** (*Weak functional cohesion*) (BIEMAN; OTT, 1994)	The extent to which the slices in the module belong together (measured as the ratio of the number of glue tokens to the total number of data tokens of the module)
96	**Conn Comp** (LORENZ; KIDD, 1994)	This is the average number of internal relationships per class/interface, and is calculated as the ratio of to the 1+R number of classes and interfaces in the package
97	**CCM** (*Component Cohesion Metric*) (BUDHKAR; GOPAL, 2012)	CCM is the Number of component's self couplings/Total number of couplings of that component. Where total number of couplings of component = self coupling + coupling with other components within system. The value of CCM lies between 0 and 1. A higher CCM value indicates more similar behavior is grouped together i.e. more tightly coupled classes are grouped together. CCM value 1 indicates high cohesion within component.
98	**D3C2** (*distance design-based direct class cohesion*) (DALLAL, 2007)	D3C2 uses the relation between the types of the parameters and the types of the attributes to predict the interactions between the methods and attributes
99	**MCC** (*Method Community Cohesion*) (QU et al., 2015)	The basic idea of MCC is to quantify how many methods of a certain class reside in the same community
100	**MCEC** (*Method Community Entropy Cohesion*) (QU et al., 2015)	MCEC uses the standard notion of In-formation Entropy to quantify the distribution of all the methods of a class among communities.
101	**LCSM** (*Lack of Conceptual Cohesion in Methods*) (MARCUS; POSHYVANYK, 2005)	LCSM uses the same information, indexed with LSI (Latent Semantic Indexing), and represents classes as graphs that have methods as nodes. It uses a counting mechanism similar to LCOM. LSI is a corpus-based statistical method for inducing and representing aspects of the meanings of words and passages (of the natural language) reflective of their usage in large bodies of text. LSI is based on a vector space model (VSM) as it generates a real-valued vector description for documents of text.
102	**LORM** (*LORM (Logical Relatedness of Methods*) (ETZKORN; DELUGACH, 2000)	LORM uses natural language processing techniques for the analysis needed to measure the conceptual similarity of methods and represents a class as a semantic network

A.2 DESCRIPTION OF COUPLING METRICS

Table 18: Description of coupling metrics

#	Coupling Metrics	Description
1	**AC** (*Association-induced coupling index*) (SARKAR; KAK; RAMA, 2008)	The extent to which there are no association-based inter-module dependencies between a module and other modules
2	**ACAIC** (*Ancestors Class-Attribute Import Coupling*) (BRIAND; DEVANBU; MELO, 1997)	Counts all CA(Class-Attribute)-interactions from class c to ancestors of class c

3	**ACMIC** (*Ancestor Class-Method Import Coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first characters indicate the type of coupling relationship between classes (A: ancestor). CM: There is a cass–method interaction if class x consists of a method that has parameter of type class y. The last two characters indicate the locus of impact: IC: Import coupling, counts the number of other classes called by class x.
4	**AFM** (*Actual Friend Methods*) (ENGLISH; CAHILL; BUCKLEY, 2012)	Actual Friend Methods (AFM), of a class declared as a friend, is proposed to count the number of methods in a class that access hidden members of classes which declare the current class as a friend. The largest value AFM can have is the number of methods in the class.
5	**AMMIC** (*Ancestors class Method–Method Import Coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first characters indicate the type of coupling relationship between classes (A: ancestor). MM: There is a method–method interaction if class x calls method of another class y, or class x has a method of class y as a parameter. The last two characters indicate the locus of impact: IC: Import coupling, counts the number of other classes called by class x.
6	**BCFI** (*Base-class fragility index*) (SARKAR; KAK; RAMA, 2008)	The value for BCFI (S) ranges from 0 to 1, with 0 indicating a system that has the worst base-class violation and 1 indicating a system that is completely free of base-class violation.
7	**Ca** (*Afferent couplings*) (MARTIN, 1994)	It is the count of the number of classes that call a given class
8	**CBM** (*Coupling Between Methods*) (TANG; KAO; CHEN, 1999)	Number of function dependency relationships between inherited and new/redefined methods
9	**CBO'** (CHIDAMBER; KEMERER, 1991)	Same as CBO, except that inheritance-based coupling is not counted.
10	**CBO** (*Coupling between Object Classes*) (CHIDAMBER; KEMERER, 1991)	CBO for a class is a count of the number of other classes to which it is coupled
11	**CBO_IUB** (*CBO Is Used By*) (KABAILI; KELLER; LUSTMAN, 2001)	CBO Is Used By: the part of CBO that consists of the classes using the target class
12	**CBO_NA** (*CBO No Ancestors*) (KABAILI; KELLER; LUSTMAN, 2001)	CBO No Ancestors: same as CBO, but the coupling between the target class and its ancestors is not taken into consideration
13	**CBO_U** (*CBO Using*) (KABAILI; KELLER; LUSTMAN, 2001)	CBO Using: the part of CBO that consists of the classes used by the target class
14	**CC** (ZHOU et al., 2012)	—
15	**CCBC** (*Conceptual Coupling Between two Classes*) (POSHYVANYK et al., 2009)	Conceptual coupling between two classes $ck \in C$ and $cj \in C$ is the average of the couplings between all unordered pairs of methods from class ck and class cj.
16	**CCBCm** (*Maximum Conceptual Coupling Between two classes*) (POSHYVANYK et al., 2009)	While CCBC coupling measure uses the same type of information as CCBCm, it uses a different counting mechanism based on average similarities as opposed to CCBCm, which is based on the strongest coupling link between classes.
17	**CDC** (*Class level Dynamic Coupling*) (GUPTA, 2011)	"The average of dynamic couplings of all objects of the class created during execution of the program."

18	**Ce** (*Efferent couplings*) (MARTIN, 1994)	A class's efferent coupling is a measure of how many other classes is used by the specific class.
19	**CF** (*Coupling Factor*) (ABREU, 1995) (ABREU; MELO, 1996)	Coupling factor is the ratio of actual number of couplings to the maximum possible pair wise couplings in a system. It does not include inheritance based class relationships.
20	**CQFS** (*Class Request For Service*) (ZAIDMAN; DEMEYER, 2004)	CQFS registers every message that the instantiations of a certain class sends during the execution of the program.
21	**CTA** (*Coupling through data abstraction*) (LI, 1998)	CTA counts the number of reference types used in the attribute declarations
22	**CTI** (*Coupling Through Inheritance*) (BRIAND; DALY; WUST, 1999)	Equal to the DIT (Depth of inheritance tree = "The length of the longest path from the class to the root class in the inheritance tree") plus the number of children (NOC = "Counts the number of direct subclasses per class").
23	**CTM** (*Coupling through message passing*) (LI, 1998)	CTM counts the number of method call expressions made into body of the measured method
24	**DAC** (*Data Abstraction* **Coupling**) (LI; HENRY, 1993b)	DAC is the number of attributes in a class that have another class as their type
25	**DAC2** (*Data Abstraction Coupling 2*) (LI; HENRY, 1993b)	DAC2 is the number of distinct classes used as types of the attributes of the class.
26	**Dca** (*Dynamic Afferent Coupling*) (SINGH; SINGH, 2010)	The percentage of number of classes accessing the methods of a class at runtime to the total number of classes
27	**DCAEC** (*Descendant Class-Attribute Export Coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first characters indicate the type of coupling relationship between classes (D: descendents) The next two characters indicate the type of interaction: CA: There is a class–attribute interaction if class x has an attribute of type class y. The last two characters indicate the locus of impact: EC: Export coupling, count number of other classes using class y.
28	**DCAIC** (*Descendant class-attribute import coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first characters indicate the type of coupling relationship between classes (D: descendents) The next two characters indicate the type of interaction: CA: There is a class–attribute interaction if class x has an attribute of type class y. The last two characters indicate the locus of impact: IC: Import coupling, counts the number of other classes called by class x.
29	**DCBO** (*Degree of dynamic coupling between two class and degree of dynamic coupling within a given set of classes*) (MITCHELL; POWER, 2005) (MITCHELL; POWER, 2003)	DCBO is the number of classes that are accessed by another class at run-time

30	**DCC** (*Direct Class Coupling*) (BANSIYA; DAVIS, 2002)	DCC is the number of other classes that the class is directly related to (by attribute declarations and message passing).
31	**DMC** (Direct Module Coupling) (HWA; LEE; KWON, 2009)	DMC is the number of Modules which a module (MD) is directly related.
32	**DCM** (*Dynamic Coupling Metric*) (HASSOUN; JOHNSON; COUNSELL, 2004b) (HASSOUN; JOHNSON; COUNSELL, 2004a) (HASSOUN; COUNSELL; JOHNSON, 2005)	The DCM is an object coupling metric derived from the intuitive observation that coupling between two objects can be defined in terms of the time during which one object influences the other.
33	**DCMEC** (*Descendant Class-Method Export Coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first characters indicate the type of coupling relationship between classes (D: descendents) The next two characters indicate the type of interaction: CM: There is a cass–method interaction if class x consists of a method that has parameter of type class y. The last two characters indicate the locus of impact: EC: Export coupling, count number of other classes using class y.
34	**Degree of dynamic coupling between two classes at runtime** (MITCHELL; POWER, 2003)	A count of the amount of times a class A accesses methods or instances variables from a class B as a percentage of the total number of methods or instance variables accessed by A
35	**Degree of dynamic coupling within a given set of classes** (MITCHELL; POWER, 2003)	This metric indicates the level of Dynamic Coupling occurring within a given set of classes
36	**DFC** (*DynamicFunction Coupling*) (APIWATTANAPONG; ORSO; HARROLD, 2005)	DFC can compute the impact sets between two functions or methods with adjustable precision and recall rates
37	**DKC** (*Dynamic Key Class*) (SINGH; SINGH, 2010)	The percentage of sum of calls sent out from the class and calls received by the class at runtime taken over the total number of static calls sent and received by all the classes
38	**DKCC** (*Dynamic Key Client Class*) (SINGH; SINGH, 2010)	The percentage of number of calls sent by a class at runtime to the total number of static calls sent by all the classes
39	**DKSC** (*Dynamic Key Server Class*) (SINGH; SINGH, 2010)	The percentage of number of calls sent to a class at runtime to the total number of static calls sent to all the classes
40	**DMMEC** (*Descendant class Method–Method Export Coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first characters indicate the type of coupling relationship between classes (D: descendents) The next two characters indicate the type of interaction: MM: There is a method–method interaction if class x calls method of another class y, or class x has a method of class y as a parameter. The last two characters indicate the locus of impact: EC: Export coupling, count number of other classes using class y.
41	**DOC** (*Dynamic Object Coupling b/w Objects*) (GUPTA, 2011)	DOC can be defined as “the weighted summation of the Dynamic inheritance Coupling, Dynamic aggregation coupling, Dynamic reference coupling and Dynamic invocation coupling
42	**Dynamic Afferent Coupling** (SINGH; SINGH, 2008)	—
43	**Dynamic Efferent Coupling** (SINGH; SINGH, 2008)	—

44	**EC** (*Export Coupling*) (BRIAND; MORASCA; BASILI, 1994)	Export coupling is the extent to which the data declarations of a software part affect the data declarations of the other software parts in the system
45	**EC_CC** (*Export Coupling at the Class level in distinct Classes*) (ARISHOLM; BRIAND; FOYEN, 2004) (ARISHOLM, 2002)	It counts the number of distinct client classes that are being used in all objects of a given class
46	**EC_CD** (*Export Coupling at the Class level in Dynamic messages*) (ARISHOLM; BRIAND; FOYEN, 2004) (ARISHOLM, 2002)	It counts the total number of messages received by all methods of all objects of a class
47	**EC_CM** (*Export Coupling at the Class level in distinct Methods*) (ARISHOLM; BRIAND; FOYEN, 2004) (ARISHOLM, 2002)	It counts the number of distinct methods received by all methods of all objects of a class
48	**EC_OC** (*Export Coupling at the Object level in distinct Classes*) (ARISHOLM; BRIAND; FOYEN, 2004) (ARISHOLM, 2002)	It counts the number of distinct client classes that are being used in a given object
49	**EC_OD** (*Export Coupling at the Object level in Dynamic messages*) (ARISHOLM; BRIAND; FOYEN, 2004) (ARISHOLM, 2002)	It counts the total number of messages received by one object from other objects
50	**EC_OM** (*Export Coupling at the Object level in distinct Methods*) (ARISHOLM; BRIAND; FOYEN, 2004) (ARISHOLM, 2002)	It counts the number of distinct methods received by an object
51	**EOCx** (*Export Object Coupling*) (YACOUB; AMMAR; ROBINSON, 1999)	EOCx(oi,oj) for an object oi with respect to an object oj, is defined as the percentage of the number of messages sent from oi to oj with respect to the total number of messages exchanged during the execution of a scenario x
52	**FCAEC** (*Friends Class-Attribute Export Coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first characters indicate the type of coupling relationship between classes (F: Friend classes)The next two characters indicate the type of interaction: CA: There is a class–attribute interaction if class x has an attribute of type class y. The last two characters indicate the locus of impact: EC: Export coupling, count number of other classes using class y.
53	**FCMEC** (*Friends Class-Method Export Coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first characters indicate the type of coupling relationship between classes (F: Friend classes)The next two characters indicate the type of interaction: CM: There is a cass–method interaction if class x consists of a method that has parameter of type class y. The last two characters indicate the locus of impact: EC: Export coupling, count number of other classes using class y.

54	**FIN** (*Fan In*) (FENTON; BIEMAN, 2014)	Count of modules (classes) that call a given class, plus the number of global data elements
55	**FMMEC** (*Friends Method-Method* *Export Coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first characters indicate the type of coupling relationship between classes (F: Friend classes) The next two characters indicate the type of interaction: MM: There is a method–method interaction if class x calls method of another class y, or class x has a method of class y as a parameter. The last two characters indicate the locus of impact: EC: Export coupling, count number of other classes using class y.
56	**FOUT** (*Fan Out*) (CHIDAMBER; KEMERER, 1994)	Count of modules (classes) called by a given module plus the number of global data elements altered by the module (class)
57	**GCC** (*Global Coupling Counts*) (OFFUTT; ABDURAZIK; SCHACH, 2008)	Refers to variables that are defined in one class and used in others. Global Coupling: Method of one class can directly access parts of the internal structure, of another class method (friend).
58	**IC** (*Import Coupling*) (BRIAND; MORASCA; BASILI, 1994)	Import Coupling is the extent to which a software part depends on imported external data declarations
59	**IC** (*Inheritance-based* *Intermodule coupling index*) (SARKAR; KAK; RAMA, 2008)	The extent to which there are no inheritance-based inter-module dependencies between a module and other modules. The maximum and minimum IC(S) values are 1 and 0, respectively. An IC(S) value of 1 is indicative of a system that does not have any inheritance-based couplings. Conversely, an IC(S) value of 0 is indicative of a system that has maximum inheritance-based couplings.
60	**IC** (*Inheritance coupling*) (TANG; KAO; CHEN, 1999)	It is measured by counting the parent classes to which the current class is coupled.The class is coupled to its parent class if one the following case occurs: 1st Inherited method invokes a redefined method; 2nd Inherited method uses at least one attribute, belonging to another method which is new or redefined; 3rd Inherited method is invoked by another method which is new or redefined.
61	**IC_CC** (*Import Coupling at* *the Class level in* *distinct Classes*) (ARISHOLM, 2002) (ARISHOLM; BRIAND; FOYEN, 2004)	It counts the number of distinct server classes used by all methods of all objects of a class
62	**IC_CD** (*Import Coupling at* *the Class level in* *Dynamic messages*) (ARISHOLM, 2002) (ARISHOLM; BRIAND; FOYEN, 2004)	It counts the total number of messages sent by all methods in all objects of a class
63	**IC_CM** (*Import Coupling at* *the Class level in* *distinct Methods*) (ARISHOLM, 2002) (ARISHOLM; BRIAND; FOYEN, 2004)	It counts the number of distinct methods invoked by all methods in all the objects of a class
64	**IC_OC** (*Import Coupling at* *the Object level in* *distinct Classes*) (ARISHOLM, 2002) (ARISHOLM; BRIAND; FOYEN, 2004)	It counts the number of distinct server classes used by the methods of the given object

65	**IC_OD** (*Import Coupling at the Object level in Dynamic messages*) (ARISHOLM, 2002) (ARISHOLM; BRIAND; FOYEN, 2004)	It counts the total number of messages sent from one object to other objects
66	**IC_OM** (*Import Coupling at the Object level in distinct Methods*) (ARISHOLM, 2002) (ARISHOLM; BRIAND; FOYEN, 2004)	It counts the number of distinct methods invoked from one object to other objects
67	**ICC** (*Inheritance coupling counts*) (OFFUTT; ABDURAZIK; SCHACH, 2008)	Inheritance coupling occurs when one class is a subclass or descendant of another. Inheritance Coupling: One class is a superclass or subclass of another class.
68	**ICP** (*Information-flow-based coupling*) (LEE et al., 1995)	The number of method invocations in a class weighted by the number of parameters of the invoked methods
69	**IFCAIC** (*Inverse Friend Class-Attribute Import Coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first two characters indicate the type of coupling relationship between classes. (IF: Inverse Friends (classes that declare a given class a as their friend). The next two characters indicate the type of interaction: CA: There is a class–attribute interaction if class x has an attribute of type class y. The last two characters indicate the locus of impact: IC: Import coupling, counts the number of other classes called by class x.
70	**IFCMIC** (*Inverse Friend Class-Method Import Coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first two characters indicate the type of coupling relationship between classes. (IF: Inverse Friends (classes that declare a given class a as their friend). The next two characters indicate the type of interaction: CM: There is a cass–method interaction if class x consists of a method that has parameter of type class y. The last two characters indicate the locus of impact: IC: Import coupling, counts the number of other classes called by class x.
71	**IFMMIC** (*Inverse Friend Method-Method Import Coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first two characters indicate the type of coupling relationship between classes. (IF: Inverse Friends (classes that declare a given class a as their friend). The next two characters indicate the type of interaction: MM: There is a method–method interaction if class x calls method of another class y, or class x has a method of class y as a parameter.The last two characters indicate the locus of impact: IC: Import coupling, counts the number of other classes called by class x.
72	**IH-ICP** (*Inheritance Information-flow-based coupling*) (LEE et al., 1995)	Same as ICP, but only counts methods invocations of ancestors of classes.

73	**IOCx** (*Import Object Coupling*) (YACOUB; AMMAR; ROBINSON, 1999)	IOCx(oi,oj) for an object oi with respect to an object oj, is "the percentage of the number of messages received by object oi that were sent by object oj with respect to the total number of messages exchanged during the execution of a scenario x."
74	**LCC** (*Loose Class Coupling*) (SHARMA; CHUG, 2015) (KAUR; MAINI, 2016)	This metric calculates the low dependency between object-structure at run-time
75	**MAC** (*Method-Attribute Coupling*) (BOWMAN; BRIAND; LABICHE, 2010)	For a given class $cl \in C$, MAC(cl) counts all MAI (Method - Attribute Interactions) from class cl to classes that are not ancestors or descendents of cl.
76	**MAEC** (*Method–Attribute Export Coupling*) (ENGLISH; CAHILL; BUCKLEY, 2012)	$MAEC = O\_FMAEC + O\_DMAEC + O\_OMAEC$
77	**MAF** (*Members Accessed by Friends*) (ENGLISH; CAHILL; BUCKLEY, 2012)	The MAF metric counts the number of hidden members in a class declaring friends, which are accessed by classes which are declared friends of the class. This measure provides information about the use of the friend construct, within a class declaring friends.
78	**MAIC** (*Method–Attribute Import Coupling*) (ENGLISH; CAHILL; BUCKLEY, 2012)	$MAIC = O\_IFMAIC + O\_AMAIC + O\_OMAIC$
79	**MGC** (*Method-Generalization Coupling*) (BOWMAN; BRIAND; LABICHE, 2010)	MGC (cl) counts all MAI (Method - Attribute Interactions) and MMI (Method-Method Interactions) from class cl to classes that are descendants or siblings of cl.
80	**MII** (*Module interaction index*) (SARKAR; KAK; RAMA, 2008)	The extent to which all external calls made to a module are routed though the APIs of the module. The MII(S) value ranges from 0 to 1. A max MII(S) value of 1 indicates an ideal system where all intermodule interaction is only through the officially designated S-API methods. Amin MII(S) value of 0 is indicative of a system with very bad intermodule interaction.
81	**MMC** (*Method-Method Coupling*) (BOWMAN; BRIAND; LABICHE, 2010)	MMC(cl) counts all MMI (Method-Method Interactions) from class cl to classes that are not ancestors or descendents of cl.
82	**MMEC** (*Method–member interactions*) (ENGLISH; CAHILL; BUCKLEY, 2012)	$MMEC = O\_FMMEC + O\_DMMEC + O\_OMMEC$
83	**MMIC** (*Method–member interactions*) (ENGLISH; CAHILL; BUCKLEY, 2012)	$MMIC = O\_IFMMIC + O\_AMMIC + O\_OMMIC$
84	**MPC'** (ZHOU et al., 2012)	A variation on MPC metric
85	**MPC** (*Message Passing Coupling*) (LI; HENRY, 1993b)	The number of messages passed among objects of a class
86	**MPEC** (*Method–Procedure Export Coupling*) (ENGLISH; CAHILL; BUCKLEY, 2012)	$MPEC = O\_FMPEC + O\_DMPEC + O\_OMPEC$
87	**MPIC** (*Method–Procedure Import Coupling*) (ENGLISH; CAHILL; BUCKLEY, 2012)	$MPIC = O\_IFMPIC + O\_AMPIC + O\_OMPIC$
88	**MQFS** (*Method Request For Service*) (SINGH; SINGH, 2008)	MQFS is defined at method level, it considers all the methods including those exhibiting polymorphism

89	**NC** (*Non-API method closedness index*) (SARKAR; KAK; RAMA, 2008)	The extent to which non-API public methods in a module are not called by other modules. What the value of NC(p) should be under this condition depends entirely on the nature of the software system. If it was the intent of the software designers that all of the public methods for all the classes in a module constitute that module's API, then NC(p) must be set to 1 when both the numerator and the denominator go to 0. In the absence of division by zero, NC for a module p becomes 1 (best case) when all of the public methods of a module p that are not "Declared API methods of p" are actual non-API methods, i.e., they never participate in intermodule call traffic.
90	**NIH-ICP** (*Noninheritance Information-flow-based coupling*) (LEE et al., 1995)	Same as ICP, but only counts methods invocations of classes not related through inheritance.
91	**O_AMPIC** (ENGLISH; CAHILL; BUCKLEY, 2012)	Metric based on method–method (MP) interactions
92	**O_DMPEC** (ENGLISH; CAHILL; BUCKLEY, 2012)	Metric based on method–method (MP) interactions
93	**O_FMPEC** (ENGLISH; CAHILL; BUCKLEY, 2012)	Metric based on method–method (MP) interactions
94	**O_IFMPIC** (ENGLISH; CAHILL; BUCKLEY, 2012)	Metric based on method–method (MP) interactions
95	**O_OMPEC** (ENGLISH; CAHILL; BUCKLEY, 2012)	Metric based on method–method (MP) interactions
96	**O_OMPIC** (ENGLISH; CAHILL; BUCKLEY, 2012)	Metric based on method–method (MP) interactions
97	**OCAEC** (*Others Class-Attribute Export Coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first characters indicate the type of coupling relationship between classes. (O: others, i.e. none of the other relationships).The next two characters indicate the type of interaction: CA: There is a class–attribute interaction if class x has an attribute of type class y. The last two characters indicate the locus of impact: EC: Export coupling, count number of other classes using class y.
98	**OCAIC** (*Others Class-Attribute Import Coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first characters indicate the type of coupling relationship between classes. (O: others, i.e. none of the other relationships).The next two characters indicate the type of interaction: CA: There is a class–attribute interaction if class x has an attribute of type class y. The last two characters indicate the locus of impact: IC: Import coupling, counts the number of other classes called by class x.

99	**OCMEC** (*Others Class-Method Export Coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first characters indicate the type of coupling relationship between classes. (O: others, i.e. none of the other relationships). The next two characters indicate the type of interaction: CM: There is a cass–method interaction if class x consists of a method that has parameter of type class y. The last two characters indicate the locus of impact: EC: Export coupling, count number of other classes using class y.
100	**OCMIC** (*Others Class-Method Import Coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first characters indicate the type of coupling relationship between classes. (O: others, i.e. none of the other relationships).The next two characters indicate the type of interaction: CM: There is a cass–method interaction if class x consists of a method that has parameter of type class y. The last two characters indicate the locus of impact: IC: Import coupling, counts the number of other classes called by class x.
101	**OMMEC** (*Other class Method–Method Export Coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first characters indicate the type of coupling relationship between classes. (O: others, i.e. none of the other relationships). The next two characters indicate the type of interaction: MM: There is a method–method interaction if class x calls method of another class y, or class x has a method of class y as a parameter. The last two characters indicate the locus of impact: EC: Export coupling, count number of other classes using class y.
102	**OMMIC** (*Other class Method–Method Import Coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first characters indicate the type of coupling relationship between classes. (O: others, i.e. none of the other relationships). The next two characters indicate the type of interaction: MM: There is a method–method interaction if class x calls method of another class y, or class x has a method of class y as a parameter. The last two characters indicate the locus of impact: IC: Import coupling, counts the number of other classes called by class x.
103	**OPFS** (*Object Response for Service*) (YACOUB; AMMAR; ROBINSON, 1999)	Percentage of total number of messages sent to the object oi from all other objects in the application during the execution of a specific scenario x
104	**OQFS** (*Object Request For Service*) (YACOUB; AMMAR; ROBINSON, 1999)	Percentage of the total number of messages sent by the object oi to all other objects in the design
105	**PAC** (*Percentage Active Classes*) (SINGH; SINGH, 2010)	The percentage of number of classes sending or receiving at least one method calls from/to another class at runtime to the total number of classes

106	**PACK** (*Number of imported packages*) (FOWLER et al., 1999)	PACK is the number of imported packages
107	**PCC** (*Parameter Coupling Counts*) (OFFUTT; ABDURAZIK; SCHACH, 2008)	Method of one class invokes method or passes parameter or passes a message to the method of another class then parameter coupling count of invoking class is incremented by one
108	**PIM** (BRIAND; WUST; LOUNIS, 1999)	PIM is the number of method invocations in C of methods in D
109	**RDE** (*Runtime export degree of coupling*) (MITCHELL; POWER, 2004a)	RDE is the number of accesses made to a class as a proportion of the total number of accesses
110	**RDI** (*Runtime import degree of coupling*) (MITCHELL; POWER, 2004a)	RDI is the number of accesses a class makes as a proportion of the total number of accesses
111	**RE** (*Runtime export coupling between objects*) (MITCHELL; POWER, 2004a)	Number of classes that access methods or instance variables from a given class
112	**RFC** (*Response for a Class*) (CHIDAMBER; KEMERER, 1991)	RFC is defined as set of methods that can be executed in response and messages received a message by the object of that class
113	**RFC1** (*Response for a Class 1*) (CHIDAMBER; KEMERER, 1994)	It is same as RFC except that it does not counts methods indirectly invoked by methods
114	**RI** (*Runtime import coupling between objects*) (MITCHELL; POWER, 2004a)	RI is the number of classes from which a class accesses methods or instance variables at run-time
115	**TCC** (*Tight Class Coupling*) (SHARMA; CHUG, 2015) (KAUR; MAINI, 2016)	It measures the tight class coupling is the high dependency between object-structure at run time
116	**TCC** (*Total coupling counts*) (OFFUTT; ABDURAZIK; SCHACH, 2008)	It is the total coupling counts
117	**TDOC** (*Total Dynamic Object Coupling*) (GUPTA, 2011)	The sum of coupling values of the object with all other objects belonging to other classes at run-time
118	**UCL** (ZHOU et al., 2012)	—
119	**Change propagation probability** (XENOS et al., 2000)	Each element of the Change propagation probability $CP = [cpij]$ for an architecture is the conditional probability that a change originating in component Ci requires changes to be made to component Cj .
120	Dep_in (LORENZ; KIDD, 1994)	The number of elements that depend on this class
121	**Dep_Out** (LORENZ; KIDD, 1994)	The number of elements on which this class depends
122	**MsgRecv** (BRIAND; DEVANBU; MELO, 1997)	The number of messages received. Counts the number of messages that instances of this class receive from instances of other classes or unclassified instances.
123	**MsgSent** (BRIAND; DEVANBU; MELO, 1997)	The number of messages sent. Counts the number of messages that instances of this class send to instances of other classes, or unclassified instances.
124	**EC_Attr** (BRIAND; DEVANBU; MELO, 1997)	The number of times the class is externally used as attribute type
125	**EC_Par** (BRIAND; DEVANBU; MELO, 1997)	The number of times the class is externally used as parameter type
126	**IC_Attr** (BRIAND; DEVANBU; MELO, 1997)	The number of attributes in the class having another class or interface as their type
127	**IC_Par** (BRIAND; DEVANBU; MELO, 1997)	The number of parameters in the class having another class or interface as their type

128	**Size Of Change** (ABDELMOEZ et al. 2006)	Each element of the Size of Change $SC = [scij]$ matrix is defined as the ratio between the number of affected methods of the receiving component caused by the changes in the interface elements of the providing components and the total number of methods in the receiving component (ABDELMOEZ; GOSEVA-POPSTOJANOVA; AMMAR, 2006)
129	**NASC** (*Number of Associations*) (GENERO; PIATTINI; CALERO, 2002)	The number of association relationships that a specific class has in a class diagram. And NAssoc is a generalization of NASC metric and represents the total number of association relationships within a class diagram for all entities
130	**NDepIn** (*Number of Dependencies In*) (GENERO; PIATTINI; CALERO, 2002)	The number of classes that depend on a specific class
131	**NDepOut** (*Number of Dependencies Out*) (GENERO; PIATTINI; CALERO, 2002)	The number of classes on which a specific class depends
132	**NAS** (*Number of associations*) (TANG; KAO; CHEN, 1999)	NAS count of the number of association lines emanating from a class in an OMT diagram
133	**DCMIC** (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first characters indicate the type of coupling relationship between classes. (D: descendents, The next two characters indicate the type of interaction: CM: There is a cass–method interaction if class x consists of a method that has parameter of type class y. The last two characters indicate the locus of impact: IC: Import coupling, counts the number of other classes called by class x.
134	**ACAEC** (*Ancestor class-attribute export coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first characters indicate the type of coupling relationship between classes (A: ancestor) The next two characters indicate the type of interaction: CA: There is a class–attribute interaction if class x has an attribute of type class y. The last two characters indicate the locus of impact: EC: Export coupling, count number of other classes using class y.
135	**ACMEC** (*Ancestor class-method export coupling*) (BRIAND; DEVANBU; MELO, 1997)	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes, different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what interactions are counted: The first characters indicate the type of coupling relationship between classes (A: ancestor). The next two characters indicate the type of interaction: CM: There is a cass–method interaction if class x consists of a method that has parameter of type class y. The last two characters indicate the locus of impact: EC: Export coupling, count number of other classes using class y.
136	**CBI** (*Coupling based on inheritance*) (**??**)	The product of the number of descendent classes of a class and the sum of internal method complexity of all methods implemented in the class.
137	**CCM** (*Conceptual Coupling Between Methods*) (POSHYVANYK et al., 2009)	The conceptual coupling between two methods $mk \in M(C)$ and $mj \in M(C)$, CCM(mk, mj), is computed as the cosine between the vectors vmk and vmj, corresponding to mk and mj in the semantic space constructed by LSI (Latent Semantic Indexing).

138	**CCMC** (*Conceptual Coupling Between a Method and a Class*) (POSHYVANYK et al., 2009)	Let $ck \in C$ and $cj \in C$ be two distinct $(ck \neq cj)$ classes in the system. Each class has a set of methods $M(ck) = mk1, \ldots, mkr$, where $r = \|M(ck)\|$ and $M(cj) = mj1, \ldots, mjt$, where $t = \|M(cj)\|$. Between every pair of methods (mk, mj) there is a CCM(mk, mj).
139	**CCBC** (*Conceptual Coupling Between two Classes*) (POSHYVANYK et al., 2009)	Is the conceptual coupling between two classes $ck \in$ C and $cj \in C$
140	**ICC** (*Individual class coupling*) (ALGHAMDI; ELISH; AHMED, 2002)	Represents the inheritance coupling between a class and all classes in the system
141	**CTC** (*Class-to-class coupling*) (ALGHAMDI; ELISH; AHMED, 2002)	Measures the inheritance coupling between any two classes in the system
142	**OSC** (*Overall system coupling*) (ALGHAMDI; ELISH; AHMED, 2002)	The inheritance coupling of the whole object-oriented systems.
143	**SIMAS** (BRIAND; WUST; LOUNIS, 1999)	The number of method invocations in C, which directly or indirectly invoke methods of D. Takes static invocations into account only
144	**PIMAS** (BRIAND; WUST; LOUNIS, 1999)	The number of method invocations in C, which directly or indirectly invoke methods of D, taking polymorphism into account.
145	**INAG** (BRIAND; WUST; LOUNIS, 1999)	'Indirect aggregation coupling'. Takes the transitive closure of the 'C has an attribute of type D' relation into ac-count. E.g. C has attribute of type D, D has an attribute of type E => C and E are indirectly coupled. The measure yields one if a class pair is directly or indirectly coupled in that manner, else zero.